

Permalink URI:

vscode://joelabenhaim.aicode-ja/chat?sessionId=b445e6364f52498c9ff3e75cbecf5c9b-Extraction-et-refont

<USER>

Analyse cette demande en détail. Fais un rapport complet de compréhension et de faisabilité. - le systeme qui permet d'afficher une session en cours coté front dans easy agents - c'est quoi? systeme complexe qui distingue msg en cours de msg passé, plus complexe à gérer, qui marque un temps d'arret sur le streaming, qui a un systeme pour ne pas perdre de morceaux face aux races, etc, étudier en détail ==> extrêmement complexe et piégeux, pas un truc à regarder vite fait. on va commencer par étudier finement, puis extraire et créer une programming doc sur ce sujet dans le nouveau module dédié qu'on placera quelque part dans orch/ (je crois). Extraire = déplacer ce code en le rendant générique, pret à etre utilisable pour le use case easy agents ET pour le use case session normale. Extraire avec les systeme associés propres, et convenablement, pas comme un porc. Extraction réussie à cette étape = le code est séparé, rebranché à easyagents mais sans qu'easy agents contienne le code, avec même fonctionnalité apparente, le changement étant invisible à l'écran, mais réel dans le code. - faire quoi? l'extraire de easy agents, le déplacer, le rendre générique - à quelle finalité? doit marcher sur les sessions aicode normales - résultat? on peut fermer une vue session aicode normale, ca ne kill plus le process orchestrateur, ca reste dans la vue sidebar activité - pourquoi? on a parfois un bug qui rend la vue grise (memory overflow electron sous vscode), et c'est chiant, on ne voit plus rien. on doit laisser finir à l'aveugle jusqu'à ce que la vue activité dise que c'est fini, ou bien kill la requete mais on a perdu le travail fait et il faut relancer le dernier message user de zéro. Avec ce nouveau systeme, si le bug arrive, on pourra fermer la vue, la rouvrir en cliquant depuis activité. En la rouvrant, ca rétablit l'affichage. - adaptation ux nécessaire: comme le process n'est pas killé en fermant, il faut un moyen de le kill. Dans la vue activité, un item a deux lignes (titre et ellapsed). Ajouter une troisième ligne avec un bouton stop (icone stop, taille small, style danger). il apparait seulement quand on est dans le cas vue invisible mais process tourne encore. si le process finit normalement ou que la vue est réaffichée, il n'apparait pas ni la troisième ligne. il est disabled après clic pour éviter double clic accidentel et bug. si on le clique, ca kill le process, de la meme facon qu'actuellement avant modif quand on ferme la fenetre ca kill.

</USER>

<ASSISTANT>

I'm thinking...

</ASSISTANT>

<ASSISTANT>

  reading project:  g ["Docs Easy Agents"](#),  g ["Docs front Easy Agents"](#),  g ["Docs front prompt"](#),  g ["Docs bus streaming"](#)

  scanning recent files

  reading project:  [easyAgents.programming-doc-readme.md](#),  [prompt-stream-dispatch-lock.programming-doc-readme.md](#),  [isomorphism-invariant-rule-between-streaming-and-rehydration-of-chat-sessions.programming-doc-readme.md](#),  [thinking-bubble-and-hourglass-streaming-rules-invariants.programming-doc-readme.md](#),  h ["Observateur live Batch"](#),  h ["Prompt et activité live"](#),  e ["Release live viewer"](#),  e ["Mode batch observer"](#),  e ["Cleanup thread Prompt"](#),  ["Cœur observateur live"](#),  ["Branchement runtime Batch"](#),  [usePromptBatchLiveViewer.hook.ts](#),  [PromptRuntimeMode.type.ts](#),  [preparePromptStreamingTurn.util.ts](#),  [AiStreamStartArgs.type.ts](#),  [SidebarActivityQuery.tsx](#),  [PromptThreadCleanup.manager.ts](#),  [iaStreamHandlers.ts](#),  [SidebarQueryRunningBridge.ts](#),  [SidebarStore.ts](#),  [openPanel.util.ts](#) [L1-L130],  [openPanel.util.ts](#) [L430-L590],  [PromptQueryDoneSound.manager.ts](#),  [openPanel.util.ts](#) [L590-L642],  e ["Mode ouverture Prompt"](#),  e ["Enregistrement thread"](#),  ["Binding session thread"](#),  e ["RPC cleanup Prompt"](#),  ["Cycle vie Prompt"](#),  e ["Propriété thread cleanup"](#),  [Prompt.tsx](#) [L360-L450],  [usePromptRunController.hook.ts](#) [L80-L180],  [usePromptRunController.hook.ts](#) [L470-L607],  [useAiStream.hook.ts](#) [L1-L240], [useAiStream.hook.ts](#) [L500-L713], [useAiStreamQueue.hook.ts](#), [usePromptStreamDispatchGateQueue.hook.ts](#), [useStreamListener.hook.ts](#), [frontMessageBus.util.ts](#), [extMessageBus.util.ts](#), [extTypedMessageBus.util.ts](#), [useAiStream.hook.ts](#) [L400-L520], [EasyAgentsBatchRunChildExecution.service.ts](#) [L220-L430], [EasyAgentsBatchRunExecution.service.ts](#) [L300-L430], [EasyAgentsBatchRunExecution.service.ts](#) [L130-L230], e ["Usage streamTerminal"](#), [VscButton.tsx](#), [VscButton.module.css](#), h ["Boutons stop danger"](#), e ["Icône stop"](#), h ["Noms icônes arrêt"](#), [VscIcon.tsx](#) [L1-L180], e ["Bouton verrou après clic"](#), [SidebarStore.queryWindow.listeners.manager.ts](#), [SidebarStore.queryWindow.mutations.manager.ts](#),

[SidebarStore.core.manager.ts](#),
[SidebarActivity.tsx](#),  [useSidebarState.hook.ts](#),
[SidebarStore.rehydrate.manager.ts](#),
[open.endpoint.ts](#),
[usePromptSessionHydration.hook.ts](#), [applyHydratedPromptSessionState.util.ts](#),  h "Ouverture session live", [stream.endpoint.ts](#), [OrchAsk.ts](#),  e "Barrière fallback persisté",  h "Finalisation stream Orch", [fallback-persist-and-title.programming-doc-readme.md](#),
[EasyAgentsBatchLiveObserver.service.test.ts](#),
[EasyAgentsBatchLastViewerStop.service.test.ts](#),
[useAiStream.hook.easyAgentsBatchLiveObserver.test.ts](#),
[Prompt.batchLiveViewer.handoff.test.tsx](#),
[usePromptSessionLocator.hook.ts](#),
[easyAgentsBatchQueryString.util.ts](#), [useEasyAgentBatchRunController.hook.ts](#) [L430-L520], [useEasyAgentsBatchController.hook.ts](#) [L360-L440],  e "UI mode live Batch", [usePromptInputViewerMode.hook.ts](#), [PromptInput.tsx](#) [L1900-L2030], [EasyAgentsBatchRunManager.service.ts](#) [L1-L330],  e "Réfs observateur Batch",  e "Réfs ouverture live",  e "Réfs batchLive front",  e "Réfs flux live Batch",  e "Réfs cleanup Prompt", [MessageRegistry.ts](#) [L1-L80],
[registerAllBusHandlers.ts](#),
[easyAgentsBatchLiveStreamHandlers.test.ts](#),
[EasyAgentsBatchLiveOpen.service.test.ts](#),
[callLocalApi.util.ts](#), [abort.endpoint.ts](#), [awaitStop.endpoint.ts](#), [clearThreadMemory.endpoint.ts](#),  e "Réfs QueryWindowItem",  e "Réfs queryState windows",  e "Réfs addQueryWindow",  e "Réfs removeQueryWindow",  e "Réfs setQuery running", [EasyAgentsBatchRunRetry.service.ts](#) [L260-L325],  l "Modules orchestrateur",
[streamLiveChild.endpoint.test.ts](#),
[Prompt.batchLiveViewer.hydration.test.tsx](#),
[SidebarQueryRunningBridge.race.test.ts](#),
[AiStreamMergedQueue.test.ts](#), [PromptStreamDispatchGateQueue.test.ts](#),  h "État exécution Orch", [OrchAskStream.ts](#),  e "createReadable finalisation",  "Durée de vie stream", [OrchAskStreamTurnRuntime.ts](#) [L860-L960],  e "Tests terminal observer",  h "Tests fermeture Prompt", [SidebarStore.manager.queryRunningUpsert.test.ts](#), [SidebarStore.rehydrate.manager.test.ts](#),  h "Tests activité sidebar", [SidebarActivityQuery.test.tsx](#),  "Tests cycle de vie panel", [SidebarUiMessages.ts](#), [SidebarStore.handlers.manager.ts](#), [uiGenericHandlers.ts](#),  e "Handler sidebar store", [UiMessages.ts](#),  e "Désenregistrement webview",  "Dispose webview",  e "Taille buffer raw live", [PromptConfigConst.const.ts](#) [L105-L135],  [ChatSessionLocator.type.ts](#),
[EasyAgentsBatchUI.port.ts](#),
[EasyAgentsBatchRunBindings.service.ts](#), [EasyAgentsBatchViewBindings.manager.ts](#), [delete.endpoint.ts](#) [L1-L80],  g "Docs sidebar extension",  g "Docs sidebar front",  g "Docs panels Prompt",  g "Docs live orch", [EasyAgentsBatchRunStore.service.ts](#)

Verdict

Joel, tu as raison de qualifier ce système d'**extrêmement complexe et piégeux** : ici, le streaming possède davantage de sés que qu'un sous-marin 🐡. Ta demande est néanmoins très bien cadrée.

Conclusion : la fonctionnalité est réalisable et je ne vois aucun blocage architectural. Le point particulièrement favorable est que **l'exécution de la requête peut déjà continuer dans l'extension après la disparition du webview**. Ce qui tue aujourd'hui la requête n'est pas une contrainte fondamentale du transport : c'est une politique explicite de nettoyage exécutée lors de la fermeture du panneau.

En revanche :

- ce n'est **pas** un simple déplacement de deux fichiers ;
- le système actuel n'est pas un bloc monolithique : c'est une chorégraphie répartie entre EasyAgents, l'orchestrateur, le transport de streaming, l'hydratation Prompt, le cycle de vie des panneaux et la sidebar ;
- une extraction propre doit séparer ;
- le **mécanisme générique de reprise d'une session live** ;

- les **politiques métier EasyAgents** ;
- la **politique des sessions normales**, qui doit continuer sans vue ;
- je recommande fortement **deux tranches successives**, avec une première extraction strictement invisible fonctionnellement, puis l'adoption par les sessions normales.

Documentation et code étudiés

J'ai lu les programming docs pertinentes avant de suivre les chemins d'exécution :

- [architecture EasyAgents](#)
- [invariants du verrou de streaming Prompt](#)
- [invariant d'isomorphisme streaming / réhydratation](#)
- [persistance de secours avant le premier chunk](#)
- [règles de rendu thinking / hourglass](#)

J'ai également suivi les implémentations backend, frontend, cycle de vie des panneaux, sidebar, transports et tests associés.

Ce qu'est réellement le système EasyAgents actuel

Le nom le plus juste est le **Live Prompt watcher des enfants EasyAgents Batch**.

Ce n'est pas uniquement un système « côté front ». Il traverse quatre couches :

Couche	Responsabilité
Exécution Batch	Lance un enfant Orch indépendamment du panneau Prompt.
Observateur live backend	Conserve tous les chunks, expose le replay et poursuit avec le flux live.
Prompt frontend	Réhydrate l'historique passé, recrée le tour courant, puis rejoue les chunks dans le pipeline Prompt normal.
Handoff terminal	Attend que toutes les écritures et convergences soient terminées avant de repasser en session classique.

L'implémentation centrale actuelle se trouve notamment dans :

- [EasyAgentsBatchLiveObserver.service.ts](#)
- [EasyAgentsBatchLiveOpen.service.ts](#)
- [EasyAgentsBatchRunExecution.service.ts](#)
- [usePromptBatchLiveViewer.hook.ts](#)
- [useAiStream.hook.ts](#)

Les cinq états importants du watcher existant

Le système ne raisonne pas seulement en « lancé / terminé ». Il possède plusieurs barrières distinctes :

État	Signification	Conséquence
Record créé	Le child et son <code>turnId</code> sont connus.	Les chunks peuvent commencer à être conservés.
Durable	Le chunk <code>fallback_persisted</code> a été reçu.	Le prompt utilisateur et la session de secours existent sur disque.
Watchable	La session est durable et au moins un token, thinking token ou raw-live chunk existe.	EasyAgents peut afficher l'action « Watch live ».
Stream terminal	La génération n'émet plus et son résultat est connu.	Cela ne signifie pas encore que le chat classique est prêt à être rechargé.
Handoff ready	Git, métadonnées child, état parent et publication UI ont convergé.	Le flux d'observation peut se fermer et le Prompt peut charger la session classique.

Cette séparation entre **stream terminal** et **handoff ready** est l'une des parties les plus importantes du système.

Le « temps d'arrêt » que tu évoques

Il ne s'agit pas principalement d'un délai arbitraire. Le flux replay/tail reste volontairement ouvert après les derniers chunks, jusqu'à `handoffReady`.

[AICode/src/extension/src/aiTools/easyAgents/batch/run/EasyAgentsBatchLiveObserver.service.ts](#)

Copy

```
while (emittedIndex < record.bufferedChunks.length) {  
  const nextChunk = record.bufferedChunks[emittedIndex];
```

```

        emittedIndex += 1;
        yield nextChunk;
    }

    if (record.handoffReady) {
        return;
    }

```

Ce maintien du stream évite que le frontend recharge trop tôt une session dont :

- le dernier message n'est pas encore totalement persisté ;
- les métadonnées terminales ne sont pas encore écrites ;
- le commit Git EasyAgents n'a pas encore convergé ;
- l'état du parent Batch n'est pas encore publié.

Comment le système distingue les messages passés du message en cours

Lorsqu'une vue live s'ouvre :

1. Le backend charge le snapshot persistant du child.
2. Il retire **uniquement les messages assistant appartenant au tour live courant**.
3. Il conserve :
 - tous les tours historiques ;
 - le message utilisateur du tour courant ;
 - les éventuels résultats d'outils côté utilisateur.
4. Le frontend réhydrate ce snapshot historique.
5. Il crée les placeholders assistant canoniques du tour courant.
6. Il rejoue tous les chunks conservés depuis le début.
7. Il continue ensuite avec les nouveaux chunks.

Cette logique se trouve dans [EasyAgentsBatchLiveOpen.service.ts](#).

Le premier snapshot live autorise temporairement les résultats d'outils orphelins, parce que les `tool_use` assistant du tour courant viennent précisément d'être retirés. Le handoff terminal repasse ensuite en hydratation stricte.

C'est ce mécanisme qui permet d'obtenir :

- **historique passé par hydratation** ;
- **tour courant par replay du stream** ;
- sans duplication du message assistant en cours.

Comment les chunks ne se perdent pas

Il y a deux systèmes complémentaires.

1. Le replay backend

Chaque chunk est ajouté à un tableau ordonné. Chaque attachement possède son propre index de lecture :

- un nouvel observateur repart à l'index 0 ;
- il rejoue tout le backlog ;
- il poursuit ensuite avec les nouveaux éléments ;
- un `EventEmitter` réveille les observateurs lorsqu'un nouveau chunk arrive.

Le flux d'un observateur n'enlève pas les chunks du buffer : plusieurs ouvertures successives peuvent donc reconstruire la même session.

2. Les deux files frontend Prompt

Une fois rejoués, les chunks traversent le pipeline Prompt normal :

- **Queue A**, dans [useAiStreamQueue.hook.ts](#) :
 - ordre total ;
 - fusion des tokens adjacents ;
 - fusion des snapshots `tool_calls` ;
 - barrières de commit React ;
- **Queue B**, dans [usePromptStreamDispatchGateQueue.hook.ts](#) :
 - verrouillage pendant les codeframes ;
 - mise en attente des chunks incompatibles ;
 - replay FIFO après déverrouillage ;
 - protection contre les deadlocks liés aux `tool_calls`.

Ces deux files sont déjà génériques. Elles ne doivent pas être déplacées dans le futur module Orch. Le nouveau système doit continuer à les utiliser telles quelles.

Analyse d'exécution LLM-DEBUG-1 : fermeture d'une session normale
Chemin actuel confirmé

Pour une session normale S, avec un tour T en cours :

1. Le frontend démarre le stream dans [useAiStream.hook.ts](#).
2. Le message atteint [iaStreamHandlers.ts](#).
3. L'extension consomme le endpoint de stream Orch.
4. [SidebarQueryRunningBridge.ts](#) maintient l'état Running et le temps écoulé.
5. Lorsque le panneau est fermé, [openPanel.util.ts](#) exécute le nettoyage du Prompt.
6. [PromptThreadCleanup.manager.ts](#) applique explicitement :
 - Abort ;
 - AwaitStop ;
 - ClearMemory.
7. Le même dispose retire l'item de la vue Activité.

[AICode/src/extension/src/extension/panels/prompt/PromptThreadCleanup.manager.ts](#)

Copy

```
if (hasThreadId) {
  // Abort → AwaitStop → ClearMemory sequence with strict ordering.
  // Abort is idempotent and returns immediately even if nothing is running.
  await callLocalApi('/api/ai/orch/query/ask/abort', {
    threadId,
    storageScopeRelPath,
  });
  // ...
}
```

Diagnostic

Hypothèse	Statut	Conclusion
Fermer le panneau tue la requête parce que le webview possède intrinsèquement le processus	Écartée	La consommation du stream se déroule dans l'extension.
Fermer le panneau tue la requête à cause d'une politique explicite	Confirmée	Le dispose appelle Abort → AwaitStop → ClearMemory.
Le stream extension peut continuer si le webview disparaît	Confirmée par l'architecture actuelle	Les erreurs de <code>postMessage</code> vers un webview détruit sont absorbées ; aucun mécanisme de cancellation du stream source n'est lié au démontage React.
Une réouverture classique peut être faite pendant le run	Écartée	Le endpoint classique réinitialise et réhydrate la mémoire Orch ; il est dangereux tant que la requête tourne.
Le principal verrou n'est donc pas de rendre l'exécution indépendante du panneau : elle l'est déjà suffisamment. Il faut : • supprimer la politique de kill automatique pour les sessions normales actives ; • conserver les chunks afin de pouvoir rattacher une nouvelle vue ; • ne lancer le endpoint classique d'ouverture qu'après le handoff terminal.		

Pourquoi l'ouverture classique actuelle ne peut pas servir à la reconnexion

Le endpoint [open.endpoint.ts](#) fait notamment :

1. chargement du snapshot persistant ;
2. `resetThreadInMemory(sessionId)` ;
3. réhydratation de la mémoire Orch ;

4. réinjection du storage scope.

C'est correct pour une session inactive, mais **interdit pendant une génération active** : cela pourrait effacer ou remplacer la mémoire que le stream utilise encore.

Le mécanisme live doit donc conserver le modèle EasyAgents :

1. ouverture live sans toucher la mémoire Orch ;
2. replay et tail ;
3. attente de la fin réelle ;
4. seulement ensuite, ouverture classique et réhydratation stricte.

Pourquoi l'objectif est faisable sans refaire tout le transport

Le comportement actuel de l'extension est favorable :

- le stream source est consommé dans le processus extension ;
- le frontend n'est qu'un destinataire ;
- le premier chunk normal est le marqueur de persistance `fallback_persisted` ;
- le prompt utilisateur est donc durable avant le début réel de l'affichage ;
- le stream Orch ne se termine qu'après sa phase de finalisation et de nettoyage des registres actifs.

Par conséquent, pour une session normale :

1. le handler source peut créer un record générique avant son premier `await` ;
2. chaque chunk peut être ajouté au buffer **avant** son émission vers le panneau d'origine ;
3. si le panneau disparaît, le handler continue à consommer et persister ;
4. une nouvelle vue peut rejoindre le record ;
5. la fin du handler source devient la barrière `handoffReady`.

Il n'est pas nécessaire, dans cette première version, de transformer toute requête normale en job de fond ou d'introduire un worker supplémentaire.

Frontière correcte de l'extraction

Tout ne doit pas être déplacé aveuglément dans le module Orch.

Élément	Destination recommandée
Registre des sessions live	Nouveau sous-module générique sous l'orchestrateur
Buffer ordonné des chunks	Nouveau module générique
Replay backlog + tail	Nouveau module générique
Barrières durable / watchable / terminal / handoff	Nouveau module générique
Ouverture live sans mutation de mémoire	Nouveau module générique
Gestion des viewers par lease	Nouveau module générique
Hook frontend live	Reste dans Prompt, mais devient générique et n'est plus nommé Batch
Queue A / Queue B	Reste dans le transport Prompt actuel
Hydratation et isomorphisme	Restent dans le sous-système Prompt actuel
Politique « zéro viewer = kill »	Reste exclusivement dans EasyAgents
Git, parent Batch, numéro de ligne, retry	Restent exclusivement dans EasyAgents
Politique « zéro panneau = continuer »	Nouvelle politique pour les sessions Prompt normales
Affichage Activité et bouton Stop	Reste dans la sidebar

Emplacement

Un sous-module `liveSession` sous le module orchestrateur est le meilleur niveau architectural :

- il dépend de l'exécution Orch et de la persistance des chats ;
- il est indépendant des composants React ;
- il est partagé entre EasyAgents et les sessions normales ;
- il ne doit pas vivre sous EasyAgents ;
- il ne doit pas non plus être enfoui dans le gestionnaire de panneaux VS Code.

La nouvelle programming doc doit être placée à la racine de ce sous-module. Il faudra également mettre à jour [la programming doc EasyAgents existante](#), qui affirme actuellement que le live observer appartient à EasyAgents.

Architecture générique recommandée

Identité d'un run live

Le record générique doit être corrélé par :

- le locator complet ;
- `sessionId` ;
- `storageScopeRelPath` ;
- le `turnId`.

Le locator seul n'est pas suffisant.

Le service actuel est indexé uniquement par child locator. Cela fonctionne pour un child Batch qui ne possède qu'un unique run, mais pas pour une session normale qui reçoit plusieurs tours successifs.

Tous les appels de mutation doivent inclure le `turnId`, notamment :

- ajout de chunk ;
- passage durable ;
- terminal source ;
- handoff ready ;
- release ;
- éviction.

Cela empêche la fin tardive d'un ancien tour de terminer ou d'effacer le nouveau.

Record générique minimal

Le record doit conserver au minimum :

- locator ;
- `turnId` ;
- chunks ordonnés ;
- durable ou non ;
- watchable ou non ;
- terminal source observé ou non ;
- handoff ready ou non ;
- viewers attachés ;
- leases d'ouverture ;
- dates de rétention ;
- signal de changement ;
- éventuel `stopRequested` pour l'idempotence de l'arrêt détaché.

Les informations suivantes **ne doivent pas appartenir au cœur générique** :

- parent Batch ;
- numéro de ligne ;
- Git ;
- statut de ligne ;
- EasyAgent id.

Le numéro de ligne peut être transporté comme contexte d'affichage lors de l'ouverture du panneau EasyAgents, sans contaminer le registre générique.

Nécessité de remplacer le booléen viewer par des leases

Le service actuel conserve seulement `viewerAttached: boolean`.

Cela présente un piège lors d'un enchaînement rapide :

1. panneau A fermé ;
2. panneau B rouvert immédiatement sur la même session ;
3. B s'attache ;
4. une release tardive provenant de A arrive ;
5. le booléen de B peut être remis à `false`.

Dans EasyAgents, cette race peut même déclencher le kill du Batch alors qu'une nouvelle vue existe.

L'extraction doit introduire :

- un identifiant de cycle de vie du panneau ;
- un identifiant de lease du viewer ;
- un set de leases attachées, et non un booléen ;
- une release portant exactement sur la lease qu'elle ferme.

C'est également important pour :

- les ouvertures abandonnées avant attachement du stream ;
- les réouvertures multiples ;
- l'expiration de la lease d'ouverture ;

- les fermetures et handoffs simultanés.

Différence fondamentale entre EasyAgents et une session normale

Le mécanisme de stream peut être partagé, mais la politique de possession est opposée.

Cas	Zéro vue	Comportement attendu
EasyAgents Batch	Aucun item, historique ou watcher Prompt	Le Batch demande un hard stop.
Session normale	Panneau Prompt fermé	La requête continue.
Session normale rouverte	Un watcher Prompt se rattache	Replay puis tail, sans redémarrer la requête.
Session normale terminée sans panneau	Plus de vue et plus de processus	Nettoyage mémoire puis disparition de l'item Activité détaché.

Il ne faut donc surtout pas déplacer `EasyAgentsBatchLastViewerStopService` dans le cœur générique.

La politique doit rester dans `EasyAgentsBatchLastViewerStop.service.ts`.

`EasyAgentsBatchViewerPresence.service.ts` peut devenir un adaptateur qui consulte :

- les bindings item/history EasyAgents ;
- les leases du child actif dans le service générique.

Cycle de vie cible des sessions normales

Panneau	Processus	État Activité	Action
Ouvert	Inactif	Item Idle, deux lignes	Ouverture classique
Ouvert	Actif	Item Running, deux lignes	Panneau interactif d'origine
Fermé	Actif	Item Running, trois lignes	Reconnexion live + bouton Stop
Rouvert	Actif	Item Running, deux lignes	Viewer live read-only avec Stop dans le Prompt
Fermé	Terminé	Aucun item détaché	Session disponible dans l'historique classique
Rouvert pendant la fin	Transition live → handoff	Deux lignes	Recharge terminale stricte
Fermé, Stop demandé	Actif puis terminal	Bouton désactivé, puis item retiré	Abort protégé contre les doubles clics

Interprétation de « vue invisible »

Je comprends ici **panneau réellement fermé**, et non simplement :

- panneau en arrière-plan ;
- onglet non actif ;
- panneau masqué par un autre onglet.

Tant que le panneau existe dans le registre VS Code, la troisième ligne ne doit pas apparaître.

Adaptation du modèle Activité

Le modèle actuel représente explicitement une liste de « fenêtres Prompt ». Or, après ce changement, un item peut exister **sans fenêtre**.

Le contrat actuel dans `SidebarStore.ts` devra donc évoluer.

Je recommande un modèle sémantique de type « activité de session » plutôt qu'un simple `WindowItem`, avec au minimum :

- locator ;
- titre ;
- `isRunning` ;
- `activeTurnId` ;
- temps live / dernier temps ;
- `isPromptPanelOpen` ;
- `continuesWhenPanelClosed` ;
- `stopRequested`.

L'`activeTurnId` est important : le bouton Stop ne doit pas pouvoir interrompre un tour ultérieur à cause d'un clic devenu obsolète.

Transitions de la sidebar

Les gestionnaires actuels concernés sont :

- [SidebarStore.queryWindow.listeners.manager.ts](#)
- [SidebarStore.queryWindow.mutations.manager.ts](#)
- [SidebarStore.rehydrate.manager.ts](#)
- [SidebarQueryRunningBridge.ts](#)

La fermeture doit devenir atomique :

- item actif et détachable : marquer le panneau fermé, conserver l’item ;
- item inactif : retirer l’item ;
- observateur EasyAgents viewer-owned : retirer l’item et laisser la politique EasyAgents décider du stop.

À la fin du source stream :

- panneau ouvert : conserver l’item, passer Idle ;
- panneau fermé : retirer l’item entièrement.

Conserver un item Idle sans panneau créerait un « fantôme » dans Activité ; je recommande donc sa suppression complète à la fin.

Reconnexion depuis l’item Activité

Le composant actuel [SidebarActivityQuery.tsx](#) ouvre toujours une session classique.

Le futur comportement doit être :

- panneau déjà ouvert : révéler le panneau existant ;
- session inactive : ouvrir normalement ;
- session active mais détachée : ouvrir avec le mode live générique.

Le frontend live doit alors :

1. demander un snapshot live-safe ;
2. hydrater l’historique passé avec tolérance temporaire des résultats d’outils orphelins ;
3. créer les placeholders assistant canoniques ;
4. lancer l’attachement au stream générique ;
5. rejouer le backlog ;
6. suivre le tail ;
7. attendre :
 - un terminal meta ;
 - la fermeture réelle du stream observateur ;
8. recharger le snapshot terminal strict ;
9. repasser en mode normal et réactiver le composer.

Le hook actuel [usePromptBatchLiveViewer.hook.ts](#) doit donc devenir générique, de même que :

- [PromptRuntimeMode.type.ts](#)
- [AiStreamStartArgs.type.ts](#)
- les états et libellés viewer de [usePromptInputViewerMode.hook.ts](#).

Les libellés Batch actuels devront rester identiques pour que la tranche d’extraction soit visuellement invisible.

Pour une session normale rouverte, le composer doit être **read-only tant que le stream source tourne**, avec un bouton Stop dans le Prompt. Il redevient interactif après le handoff terminal.

Faisabilité exacte du bouton Stop dans Activité

Cette partie est relativement simple.

Le design system possède déjà tout ce qui est demandé :

- [VscButton.tsx](#) supporte :
 - `size="small"` ;
 - `variant="danger"` ;
 - `disableAfterClick` ;
 - icône + label ;
- [VscIcon.tsx](#) expose déjà l’icône `SquareStop`.

La troisième ligne peut donc utiliser un bouton compact :

- icône Stop ;
- label `Stop` ;
- taille `small` ;
- variante `danger` ;
- désactivation immédiate après clic.

Aucune modification du design system ne paraît nécessaire.

Protection backend obligatoire

La désactivation frontend ne suffit pas. L'action doit transmettre :

- le locator exact ;
- le `turnId` attendu.

L'extension doit accepter l'arrêt uniquement si :

1. la session possède toujours ce tour actif ;
2. il s'agit d'une requête normale détachable ;
3. aucun panneau Prompt n'a été rouvert entre-temps ;
4. aucun arrêt n'a déjà été accepté.

Ainsi, un vieux clic ou deux sidebars concurrentes ne peuvent pas tuer un tour plus récent.

Je recommande une commande typée sidebar → extension, déclarée dans [SidebarUiMessages.ts](#), plutôt qu'un appel frontend connaissant directement les détails internes de l'orchestrateur.

Arrêt détaché : comportement attendu

L'arrêt depuis Activité doit réutiliser la séquence actuelle :

1. abort du thread ;
2. attente de la convergence réelle ;
3. clear de la mémoire Orch ;
4. conservation du snapshot persistant terminal ;
5. retrait de l'item Activité lorsque le source stream s'arrête.

Cette séquence doit être extraite de [PromptThreadCleanup.manager.ts](#) dans une primitive réutilisable par :

- l'ancien nettoyage d'un panneau inactif ;
- le bouton Stop détaché ;
- les éventuels nettoyages terminaux sans panneau.

Pièges majeurs à traiter explicitement

1. Plusieurs tours successifs dans la même session

Le service actuel est pratiquement mono-tour par locator. Une session normale ne l'est pas.

Un ancien record terminal ne doit pas empêcher le tour suivant, et la finalisation tardive d'un ancien tour ne doit jamais affecter le nouveau.

2. Release tardive d'un ancien panneau

Un booléen viewer n'est pas suffisant. Il faut des leases liées au cycle de vie exact du panneau.

3. Double comptage Activité lors d'une reconnexion

La requête normale est déjà suivie par son handler source. Le stream de replay ne doit pas appeler une seconde fois le démarrage de l'activité et remettre l'elapsed à zéro.

À l'inverse, le watcher EasyAgents a actuellement besoin de posséder l'activité de son panneau, car son exécution Batch source n'est pas suivie comme une requête Prompt normale.

4. Ouverture classique pendant le run

C'est interdit : elle réinitialise la mémoire. Seule l'ouverture live-safe peut être utilisée.

5. Terminal meta différent de handoff ready

Le terminal meta ne suffit jamais à déclencher le rechargement classique. Il faut aussi que le stream observateur se ferme après la convergence backend.

6. Fin sans terminal meta

Le frontend exige actuellement un terminal `stop_reason` ou `error` avant le handoff. Une erreur basse couche qui ferme la source sans terminal explicite pourrait laisser le viewer bloqué.

Le module générique doit garantir qu'une fin anormale fournit un terminal cohérent au replay, ou propage une erreur de stream permettant au frontend de produire son meta d'erreur canonique.

7. Fermeture juste avant le démarrage du handler source

Le record doit être créé tout au début du handler extension, avant son premier `await`. Ainsi, fermeture et démarrage sont ordonnés dans le même processus sans fenêtre dangereuse.

8. Stop détaché devenu obsolète

Un arrêt par simple `sessionId` pourrait tuer le tour suivant. La validation du `turnId` attendu est indispensable.

9. Fin du run pendant la réouverture

Les deux ordres doivent fonctionner :

- fin puis création du panneau : ouverture terminale classique ;
- création du panneau puis fin : replay, fermeture du stream et handoff.

10. Nettoyage mémoire sans panneau

Si le run se termine alors que la vue reste fermée, la mémoire Orch ne doit pas rester indéfiniment. Elle peut être nettoyée après la persistance terminale ; une réouverture ultérieure réhydratera le snapshot.

11. Conservation mémoire du buffer live

Le système retient une copie de chaque `AiStreamedChunk`. Étendu à toutes les sessions normales, ce coût devient significatif.

Il faut au minimum :

- un seul record actif par tour ;
- un remplacement propre lors du tour suivant ;
- une expiration terminale réellement exécutée ;
- une éviction explicite sur suppression/retry ;
- une rétention courte après handoff ;
- aucun record terminal conservé indéfiniment.

12. `raw_live_received`

Le buffer EasyAgents conserve actuellement aussi les événements raw-live, tandis que le frontend n'en garde qu'un ring buffer limité. Sur de très longues générations, les raw events peuvent être la partie la plus volumineuse.

Il ne faut pas modifier cette sémantique pendant l'extraction sans décision explicite, mais la programming doc doit signaler ce coût.

13. Reset de la sidebar

La réhydratation actuelle reconstruit uniquement les panneaux ouverts. Après l'introduction de requêtes détachées, elle devra aussi relire les runs normaux actifs dans le registre générique, sinon un reset de la sidebar pourrait faire disparaître un processus encore vivant de la vue Activité.

14. Suppression d'un chat

Le endpoint `delete.endpoint.ts` bloque déjà la suppression lorsqu'un item est Running. Il devra continuer à bloquer les runs détachés, idéalement en consultant le registre runtime générique plutôt qu'en dépendant uniquement de la projection UI.

Impact mémoire et limite fonctionnelle

Cette fonctionnalité améliore la **résilience au crash du renderer**, mais elle ne corrige pas la cause du renderer gris.

Elle permettra :

- de fermer le renderer malade ;
- de libérer sa mémoire ;
- de laisser l'extension et Orch finir ;
- de reconstruire un nouveau frontend depuis le backlog.

Cependant, si le problème vient de la taille intrinsèque d'un unique tour gigantesque, rejouer tout ce tour dans un nouveau renderer peut reproduire le problème.

La promesse raisonnable est donc :

La fermeture/réouverture devient une stratégie de récupération fiable lorsque le renderer est corrompu, bloqué ou a accumulé trop d'état ; elle ne garantit pas qu'un transcript individuellement trop grand deviendra affichable sans pagination ou limitation supplémentaire.

Le buffer live est process-local :

- il survit à la fermeture du panneau ;
- il ne survit pas à un reload de l'extension ou à un arrêt de VS Code ;
- cela correspond à la demande actuelle, puisque le processus Orch disparaîtrait lui aussi dans ces cas.

Plan recommandé en deux tranches

Tranche 1 — Extraction générique, zéro changement visible

Objectif

Extraire le watcher EasyAgents dans le nouveau module Orch et reconnecter EasyAgents dessus sans changer le comportement à l'écran.

Travaux

- créer le registre générique locator + turn ;
- créer les leases de viewers ;
- déplacer l'ouverture live-safe ;
- déplacer le replay backlog + tail ;
- déplacer les barrières durable/watchable/terminal/handoff ;
- déplacer les endpoints live sous le domaine Orch ;
- rendre génériques :
 - le canal de stream ;
 - le type de démarrage frontend ;

- les runtime modes ;
- le hook viewer ;
- garder les libellés Batch identiques ;
- adapter EasyAgents :
- execution sink ;
- viewer presence ;
- last-viewer stop ;
- retry eviction ;
- handoff ordering ;
- supprimer les implémentations EasyAgents devenues obsolètes ;
- créer la nouvelle programming doc ;
- mettre à jour la programming doc EasyAgents.

Critères d'acceptation

- EasyAgents présente exactement le même écran ;
- `Watch live` apparaît au même moment ;
- replay et tail donnent le même transcript ;
- fermeture du dernier viewer EasyAgents conserve le hard-stop actuel ;
- Git et handoff conservent strictement leur ordre ;
- retry détruit toujours le bon record ;
- aucun fichier `*.generated.*` n'est modifié manuellement.

Cette tranche est indispensable pour prouver que l'extraction est correcte avant d'ajouter le nouveau comportement.

Tranche 2 — Sessions normales détachables et UX Activité

Objectif

Brancher le même mécanisme sur les requêtes Prompt normales.

Travaux

- enregistrer le source run dans le handler normal ;
- bufferiser chaque chunk avant son émission au panneau ;
- marquer la persistance à `fallback_persisted` ;
- marquer le handoff après la fin réelle du handler source ;
- scinder le nettoyage de panneau :
- détacher la vue ;
- ne pas abort la requête normale active ;
- conserver les nettoyages sans danger ;
- retenir l'item Activité lorsque le panneau est fermé ;
- ajouter la troisième ligne Stop ;
- ouvrir le Prompt en mode live depuis l'item détaché ;
- faire disparaître la troisième ligne dès que le panneau est recréé ;
- retirer l'item quand le run termine sans panneau ;
- ajouter l'arrêt protégé par locator + turn ;
- nettoyer la mémoire Orch après convergence d'un run terminé sans vue ;
- adapter suppression, rehydrate sidebar et tests de races.

Critères d'acceptation

- fermer une vue normale en cours n'interrompt plus la génération ;
- la requête continue d'apparaître dans Activité ;
- cliquer le titre restaure le transcript déjà reçu et poursuit le stream ;
- aucun chunk n'est doublé ou perdu ;
- le Prompt rouvert reste non interactif jusqu'au handoff ;
- le bouton Stop détaché n'apparaît que sans panneau ;
- il se désactive immédiatement ;
- il ne peut pas interrompre un tour plus récent ;
- après fin ou arrêt, l'item détaché disparaît ;
- les sessions EasyAgents conservent leur comportement historique.

Surface de code estimée

Le changement dépassera probablement **35 fichiers**, tests compris, et pourrait atteindre **45 à 50 fichiers** si les renommages génériques sont menés proprement.

Zone	Principaux hotspots confirmés
Cœur live	
EasyAgents actuel	observer , ouverture live
Wiring EasyAgents	manager Batch , exécution Batch , retry Batch
Source normal	handler de stream normal , OrchAskStream
Transport live	registre des messages streamés , handler live Batch actuel
Cycle de vie du panneau	openPanel.util.ts , PromptThreadCleanup.manager.ts
Prompt live frontend	hook live Batch , modes runtime Prompt , transport AI stream
Activité	contrat SidebarStore , projection Query
Tests	Tests observer, handoff, endpoint stream, panel dispose, sidebar store, activity UI, stale stop et courses fermeture/réouverture
Aucune dépendance npm supplémentaire et aucune migration de stockage durable ne paraissent nécessaires.	

Niveau de risque

Stratégie	Risque
Un seul gros patch extraction + sessions normales + UX	Très élevé
Tranche 1 seule, comportement EasyAgents inchangé	Moyen à élevé , mais contrôlable
Tranche 2 après validation de la tranche 1	Élevé , mais isolable
Extraction superficielle conservant booléens et noms Batch	Élevé à long terme , avec dette et races probables

Le principal risque n'est pas la quantité de code. C'est l'ordre des événements :

- chunk source ;
- persistance de secours ;
- ouverture du viewer ;
- fermeture du panneau ;
- terminal meta ;
- finalisation Orch ;
- handoff ;
- rechargement classique ;
- arrêt utilisateur ;
- démarrage éventuel d'un nouveau tour.

Une erreur de barrière peut produire soit une perte de chunks, soit un double affichage, soit un reset mémoire pendant la génération.

Clarifications que j'ai déduites

Je pars des invariants suivants pour la future spécification :

- « vue invisible » signifie **panneau fermé**, pas simplement non actif ;
- lorsqu'une requête détachée termine, son item Activité disparaît entièrement ;
- la session reste naturellement disponible dans l'historique Chat ;
- une session normale rouverte pendant son run est temporairement en mode viewer read-only ;
- elle possède un bouton Stop dans le Prompt lui-même ;
- le comportement EasyAgents « aucun viewer = hard stop » reste inchangé ;
- le bouton Activité ne peut arrêter que le `turnId` qu'il affiche ;
- une réouverture intervenue juste avant le clic rend le clic Stop obsolète et non destructeur ;
- le travail ne cherche pas à corriger la cause profonde du renderer gris ;
- les fichiers générés restent totalement intouchés : seuls leurs fichiers sources sont modifiés, puis la génération normale du projet produit les contrats.

Conclusion de faisabilité

Oui, la demande est faisable proprement.

Le point décisif est que le stream normal est déjà consommé côté extension : il peut survivre à la destruction du panneau dès lors que le dispose cesse de l'aborder. Le watcher EasyAgents fournit déjà presque toutes les briques nécessaires :

- snapshot live-safe ;

- distinction passé / tour courant ;
- buffer complet ;
- replay + tail ;
- attente de handoff ;
- réhydratation terminale stricte ;
- réutilisation du pipeline Prompt ;
- protections d'ordre frontend.

Mais l'extraction doit corriger les hypothèses trop spécifiques à EasyAgents :

- un seul tour par locator ;
- un seul viewer booléen ;
- ouverture uniquement après watchabilité ;
- activité possédée par le viewer ;
- hard-stop lorsque le dernier viewer disparaît.

L'emplacement Orch est pertinent, à condition de ne pas y déplacer les politiques EasyAgents ni les composants frontend.

TL-DR

- Le système actuel est le **Live Prompt watcher EasyAgents Batch**.
- Il sépare historique hydraté et tour courant rejoué.
- Il conserve les chunks, rejoue le backlog, suit le tail et attend une barrière de handoff distincte de la fin du stream.
- La fermeture d'une session normale tue aujourd'hui la requête uniquement parce que le dispose exécute explicitement Abort → AwaitStop → ClearMemory.
- Le transport extension peut continuer sans webview : la fonctionnalité est donc architecturalement possible.
- Il faut d'abord extraire le cœur live dans un module Orch générique, rebrancher EasyAgents sans changement visible, puis adopter ce cœur pour les sessions normales.
- Les politiques EasyAgents et session normale doivent rester opposées et séparées.
- Le bouton Stop demandé est déjà supporté par le design system.
- Les principaux pièges sont les tours successifs, les viewer leases, les stops obsolètes, le double comptage Activité, le handoff terminal et la rétention mémoire.
- Je recommande une spécification très détaillée avant codage, puis deux tranches séparées.



Continuer l'étude des races et états



Rédiger la spécification détaillée



Coder directement par étapes

</ASSISTANT>

<USER>

Orchestrated comprehensive technical feasibility analysis for live session extraction Reçu. Extraction du watcher live EasyAgents (observer backend + buffer replay + queues front A/B) vers un module générique orch/liveSession, réutilisable par les sessions AICode normales en plus des children Batch. Diagnostic clé posé par AICode : fermer le panel ne tue pas la requête à cause d'une contrainte de transport, c'est une politique explicite (Abort → AwaitStop → ClearMemory dans PromptThreadCleanup) ; le stream continue déjà côté extension sans webview, donc l'objectif est faisable sans refaire le transport. Points d'architecture retenus : identité par locator+turnId et non locator seul (les sessions normales sont multi-tours, contrairement aux children Batch mono-tour, donc un tour terminé tardivement ne doit pas affecter le suivant) ; leases de viewer à la place du booléen viewerAttached (race fermeture rapide/réouverture qui remettrait le booléen à false) ; distinction stream terminal / handoff ready conservée comme barrière séparée ; politique EasyAgents (zéro vue = hard stop) et politique session normale (continue) qui restent opposées et ne doivent pas fusionner. Bouton Stop en troisième ligne dans Activité, protégé par le turnId attendu côté backend pour qu'un clic obsolète ne tue pas un tour plus récent. Plan en deux tranches : extraction générique à comportement EasyAgents strictement inchangé d'abord, adoption par les sessions normales + UX Activité ensuite. Surface estimée 35 à 50 fichiers. 14 pièges listés, les plus lourds étant le multi-tour par locator, les leases de viewer, le double comptage Activité lors d'une reconnexion, et la rétention mémoire du buffer live étendue à toutes les sessions. Verdict AICode : faisable, aucun blocage architectural, recommande spec détaillée avant code. J'attends la suite. le mode read only est pour contourner une difficulté ou juste comme ça pour le fun ? Démêlé contraintes techniques et inférences architecturales du mode lecture seule Démêlé contraintes techniques et inférences architecturales du mode lecture seule Ni l'un ni l'autre de façon certaine, le texte ne le justifie pas explicitement. Même défaut que pour le bouton Stop : la phrase "le composer doit être read-only tant que le stream source tourne" est posée directement, sans phrase de justification accolée. Mais contrairement au bouton Stop, il y a un passage ailleurs dans le rapport qui pointe vers une vraie contrainte technique, sans qu'AICode fasse lui-même le lien explicite avec le read-only.

Section "Pourquoi l'ouverture classique actuelle ne peut pas servir à la reconnexion" : Le endpoint `open.endpoint.ts` fait notamment : chargement du snapshot persistant ; `resetThreadInMemory(sessionId)` ; réhydratation de la mémoire Orch ; réinjection du storage scope. C'est correct pour une session inactive, mais interdit pendant une génération active : cela pourrait effacer ou remplacer la mémoire que le stream utilise encore. Donc rendre un composer interactif, capable d'envoyer un nouveau message, pendant qu'un tour tourne encore sur ce même thread, ouvrirait une question qu'AI Code ne traite nulle part dans les 14 pièges : que se passe-t-il si un nouveau message arrive sur une session dont la mémoire Orch est activement utilisée par une génération en cours ? Rien dans le document ne décrit ce chemin. Le viewer live tel que décrit ne fait que rejouer et suivre le tour existant, il n'est jamais présenté comme capable d'accepter une nouvelle entrée. Le lien logique probable est celui-là : lecture seule = éviter d'avoir à concevoir un chemin d'envoi concurrent sur un thread déjà en cours de génération, qui toucherait à la même mémoire Orch que le rapport dit explicitement dangereuse à modifier pendant un stream actif. Mais c'est ma reconstruction, pas une phrase d'AI Code. Le rapport ne fait jamais ce lien lui-même. Autre chose que je ne peux pas confirmer avec ce texte : si le composer est déjà verrouillé pendant un tour normal, panneau resté ouvert du début à la fin, sans passer par la fermeture/réouverture. Le rapport ne parle jamais de ce cas de base, donc impossible de dire si le read-only pour la reconnexion est une continuité du comportement existant ou une règle nouvelle propre à ce mécanisme. oui il a raison. il faut verrouiller pendant ce passage. et il verrouille seulement pendant le rattrapage streaming ou toute la réponse jusqu'à end turn ? Démêla la durée du verrouillage du compositeur après reconnexion Démêla la durée du verrouillage du compositeur après reconnexion Toute la réponse jusqu'à end turn, pas seulement le rattrapage. Le texte est explicite à deux endroits. La phrase de définition elle-même : "le composer doit être read-only tant que le stream source tourne... Il redevient interactif après le handoff terminal." "Tant que le stream tourne" couvre toute la durée de la génération, pas juste le temps de rejouer les chunks déjà bufferisés. La séquence d'étapes du frontend live, dans l'ordre donné par le rapport : rejouer le backlog (le rattrapage), suivre le tail (le live qui vient après, les nouveaux chunks au fur et à mesure), attendre un terminal meta, attendre la fermeture réelle du stream observateur, recharger le snapshot terminal strict, et en toute dernière étape seulement : "repasser en mode normal et réactiver le composer." La réactivation est explicitement la dernière ligne de la liste, après le tail, pas après le rattrapage. Ça recoupe aussi la distinction que le rapport pose lui-même entre deux barrières différentes, stream terminal (la génération n'émet plus) et handoff ready (Git, métadonnées, état parent et publication UI ont convergé, le Prompt peut charger la session classique). Le "handoff terminal" cité dans la phrase de réactivation du composer semble fusionner les deux en un raccourci, mais rien dans le texte ne suggère une réactivation anticipée dès la fin du seul rattrapage, avant que le tail ait fini de suivre et que la convergence backend soit passée. je dois l'interroger sur la nécessité de bloquer autant. bloquer pendant un état instable, seulement si absolument nécessaire, serait plus judicieux. même zéro blocage et revenir à l'état invariant d'une session normale le plus tôt possible, immédiatement si possible, sinon au plus tôt. Je dois aussi lui demander l'intérêt du second bouton qu'il a ajouté. Il doit y avoir une raison ? sinon c'est nul. a voir. Démêla les contradictions entre barrières de verrouillage et déverrouillage réel Démêla les contradictions entre barrières de verrouillage et déverrouillage réel Sur le blocage. Le texte a une contradiction interne exploitable directement. La phrase de définition dit "le composer doit être read-only tant que le stream source tourne". "Stream source", dans le vocabulaire du rapport, c'est la génération elle-même, celle qui s'arrête à l'état stream terminal ("la génération n'émet plus"). Ça pointerait vers un déverrouillage dès stream terminal. Mais la séquence concrète de reconnexion attend autre chose avant de réactiver le composer : rejouer le backlog, suivre le tail, attendre un terminal meta ET la fermeture réelle du stream observateur, recharger le snapshot terminal strict, et seulement là réactiver. Le rapport a montré plus haut, code à l'appui, que ce stream observateur ne se ferme (return) que quand `record.handoffReady` est vrai, un état séparé et plus tardif, défini comme "Git, métadonnées child, état parent et publication UI ont convergé". Le rapport dit lui-même que la séparation stream terminal / handoff ready est "l'une des parties les plus importantes du système", et pourtant utilise les deux comme s'ils étaient interchangeables pour décider du déverrouillage du composer. Question précise à poser plutôt que la question générale : lequel des deux est la vraie frontière du danger, et pourquoi l'étape de reconnexion attend l'autre si ce n'est pas lui le risque réel. Deuxième chose qui va dans ton sens : handoff ready est défini uniquement en langage EasyAgents, Git plus métadonnées child plus état parent Batch plus publication UI. Une session normale n'a rien de tout ça. Le rapport générise la structure de la barrière (elle est listée parmi les éléments à déplacer dans le module générique) mais ne dit jamais ce qu'elle recouvre concrètement hors du cas Batch. Si pour une session normale handoff ready se réduit à presque rien, proche de la persistance terminale elle-même, resserrer la fenêtre suffit, pas besoin de repenser la logique de verrouillage. Ton "même zéro blocage" est une demande d'une autre nature que "resserrer la fenêtre". Resserrer, c'est déverrouiller à stream terminal au lieu de handoff ready, ça reste dans ce que le rapport décrit déjà. Zéro blocage, c'est accepter un nouveau message pendant qu'un tour précédent génère encore sur le même thread, un scénario absent des 14 pièges listés. Le rapport ne dit nulle part ce qui arriverait à un message envoyé pendant une génération active sur la même session : mis en file, rejeté, ou traité comme un tour parallèle. Les deux questions valent d'être posées séparément, la deuxième est structurellement plus lourde que la première. Sur le deuxième bouton. Le rapport ne donne pas de raison, même trou que pour le read-only. Mais ta demande initiale contient la pièce manquante. Tu avais toi-même écrit que la troisième ligne d'Activité, celle avec le bouton stop, n'apparaît pas si "la

vue est réaffichée". Au moment précis où le panneau se rouvre et où AICode active son état read-only, le bouton stop d'Activité vient de disparaître par ta propre règle. Sans bouton dans le Prompt, il n'y aurait plus aucun moyen d'interrompre la génération pendant toute la fenêtre de rattrapage et de suivi, jusqu'au déverrouillage. C'est une reconstruction en croisant ton texte et le rapport, pas une justification qu'AICode a écrite. Mais elle répond à "il doit y avoir une raison" : oui probablement celle-là, à confirmer avec lui puisqu'il ne l'a jamais formulée lui-même. si il y a un bouton pour stopper dans la fenetre normale. peut etre qu'il parlait juste de le réactiver alors et que tu as mal compris par ignorance de l'interface. je vais montrer notre conversation à AICode, puis lire sa réponse, et produire une liste de remarques. ensuite à lui de ré-analyser. Reconnut l'erreur d'interprétation sur le bouton Stop existant Reconnut l'erreur d'interprétation sur le bouton Stop existant Oui, ça change la lecture, et probablement dans le bon sens. Si la fenêtre Prompt normale a déjà un bouton pour stopper une génération en cours, indépendamment de ce nouveau mécanisme, juste le comportement standard pendant qu'une réponse streamée, alors "avec un bouton Stop dans le Prompt" ne décrit probablement pas un second bouton nouveau. Juste ce bouton existant qui reste visible dans la vue reconnectée parce qu'une génération y tourne, exactement comme il le serait pour n'importe quelle session normale en cours de génération. Mon erreur : j'ai traité ça comme un ajout, au même titre que le read-only, parce que ta demande initiale à toi ne mentionnait qu'un bouton dans Activité. Mais "pas mentionné dans ta demande" et "n'existe pas dans l'interface" ne sont pas la même chose. Je n'ai aucune visibilité sur l'interface réelle d'AICode, et j'ai comblé ce trou par supposition au lieu de le signaler comme angle mort. Si le bouton existe déjà normalement, il n'y a rien à justifier, c'est la règle habituelle (génération en cours = stop visible) qui continue de s'appliquer ici, point. Ça rend aussi ma reconstruction de tout à l'heure inutile, celle qui expliquait le bouton Prompt par la disparition de la troisième ligne d'Activité à la réouverture. Pas besoin de ce détour si c'est juste le bouton standard qui était déjà là. Le read-only est une question séparée, et le même doute s'applique en miroir : si le composer est déjà verrouillé pendant n'importe quelle génération normale, panneau resté ouvert du début à la fin sans fermeture, alors le read-only pour la reconnexion ne serait pas non plus une invention d'AICode, juste la même règle qui continue de s'appliquer. Ça, je ne sais pas non plus si c'est le cas côté interface actuelle. Ce que ça ne change pas : le passage sur le danger mémoire Orch (resetThreadInMemory pendant un stream actif) reste valable que le verrouillage soit nouveau ou déjà standard, il explique pourquoi on ne peut pas le lever trop tôt, pas s'il est neuf ou pas. Claude is AI and can make mistakes. Please double-check responses. ----- 1) Quand je dis qu'il faut faire un module extrait, ça comprend un répertoire par projet. Lui trouver un nom de rep clair et représentatif et pas trop long, avoir un de chaque dans orch, front, shared (comme d'habitude dans le projet). Le programming doc dans orch. Tout avec des sous dossiers. Le backend est en OOP avec composition, le front avec composants découpés et unitaires, hooks si nécessaires. Ne pas regrouper ce qui ne doit pas être regrouper, le faire intelligemment. tu as le project map, tu sais ce qu'on veut comme architecture. Pas des fichiers jetés à droite à gauche ! 2) Je suis d'accord pour les tranches. La première est l'extraction sans modif fonctionnelle. 3) Tout ce qui est programmé doit être testé. 4) Il y a déjà un programming doc qui explique ce truc, il faudra le déplacer et modifier, ou supprimer et recréer. 5) la conversation avec claudie mentionne le fait de rebrancher le cas normal le plus vite possible. A t on besoin de ce read only ? Si oui, combien de temps ? 6) cas du bouton stop en plus ? réel ou fantasmé par claudie ? si réel, pourquoi ? Je l'ai pas demandé. Explique pourquoi tu le juges nécessaire ? 7) Il n'y a pas de "vue invisible", tu as inventé ça. J'ai été clair sur ce que je voulais. Relis la demande initiale. Le panneau est ouvert ou fermé, rien d'autre. 8) Oui fais évoluer sidebar store, tu as raison. 9) le raw live buffer n'est pas si gros. ça marche pour easy agents. ça marchera pour aicode. c'est que de la mémoire string, pas du process lourd. 10) A toi de réanalyser en profondeur. Très en profondeur. Pas d'analyse superficielle. C'est un boulot risqué, mais un risque assumé, avec extra attention de ta part.

</USER>

<ASSISTANT>

I'm thinking...

</ASSISTANT>

<ASSISTANT>

🔧  reading project:  g ["Docs du watcher live"](#),  g ["Docs streaming Prompt"](#),  g ["Docs transport front"](#),  g ["Docs streaming Orch"](#),  h ["Documentation live actuelle"](#),  h ["Verrou et arrêt Prompt"](#),  h ["État activité session"](#),  [easyAgents.programming-doc-readme.md](#) [L1-L115],  [easyAgents.programming-doc-readme.md](#) [L930-L1205],  [prompt-stream-dispatch-lock.programming-doc-readme.md](#),  [isomorphism-invariant-rule-between-streaming-and-rehydration-of-chat-sessions.programming-doc-readme.md](#),  [fallback-persist-and-title.programming-doc-readme.md](#),  [thinking-bubble-and-hourglass-streaming-rules-invariants.programming-doc-readme.md](#),  [EasyAgentsBatchLiveObserver.service.ts](#),  [EasyAgentsBatchLiveOpen.service.ts](#),  [EasyAgentsBatchViewerPresence.service.ts](#),  [EasyAgentsBatchLastViewerStop.service.ts](#),  [EasyAgentsBatchRunManager.service.ts](#),

[EasyAgentsBatchRunStore.service.ts](#), [EasyAgentsBatchRunExecution.service.ts](#) [L300-L430], [EasyAgentsBatchRunExecution.service.ts](#) [L650-L940], [EasyAgentsBatchRunChildExecution.service.ts](#), [EasyAgentsBatchRunRetry.service.ts](#) [L260-L330], [iaStreamHandlers.ts](#), [OrchAskStream.ts](#), [OrchAskStreamAbort.ts](#), ["Contrôleur actif par thread"](#), [OrchAskStreamWiring.ts](#), [PromptRuntimeMode.type.ts](#), [usePromptBatchLiveViewer.hook.ts](#), [usePromptInputViewerMode.hook.ts](#), [Prompt.batchLiveViewer.handoff.test.tsx](#), [PromptInput.batchLiveViewerMode.test.tsx](#), [usePromptRunUiDerived.hook.ts](#), [usePromptCancelController.hook.ts](#), ["Composer et bouton Stop"](#), ["Textarea en cours de run"](#), ["Contrôle Send Stop textarea"](#), ["Invariant textarea normal"](#), [applyHydratedPromptSessionState.util.ts](#), ["Override local composer"](#), [Prompt.tsx](#) [L100-L240], [Prompt.tsx](#) [L240-L360], [Prompt.tsx](#) [L360-L560], ["Marquage saisie locale"](#), [Prompt.tsx](#) [L720-L830], ["onChange Prompt local"](#), ["Saisie prompt value"](#), ["Rendu PromptInput normal"](#), [Prompt.tsx](#) [L880-L1040], [AiStreamStartArgs.type.ts](#), [useAiStream.hook.ts](#) [L1-L220], [useAiStream.hook.ts](#) [L420-L714], [easyAgentsBatchLiveStreamHandlers.ts](#), [MessageRegistry.ts](#), [PromptThreadCleanup.manager.ts](#), [openPanel.util.ts](#) [L1-L180], [openPanel.util.ts](#) [L250-L450], [openPanel.util.ts](#) [L520-L642], [OpenedPanelsRegistry.util.ts](#), [closePromptPanel.util.ts](#), [openPanel.util.ts](#) [L430-L540], [SidebarStore.ts](#), [SidebarStore.queryWindow.listeners.manager.ts](#), [SidebarStore.queryWindow.mutations.manager.ts](#), [SidebarStore.rehydrate.manager.ts](#), [SidebarQueryRunningBridge.ts](#), [SidebarActivityQuery.tsx](#), [SidebarUiMessages.ts](#), ["Architecture InteractGate ext"](#), ["Architecture InteractGate front"](#), ["Contrats InteractGate"](#), ["Architecture Autocfg ext"](#), ["Contrats Autocfg"](#), ["Nom liveSession existant"](#), ["Nom sessionStream existant"](#), ["Nom liveTurn existant"](#), ["Symbole liveSession"](#), ["Tests live EasyAgents"](#), ["Tests viewer EasyAgents"](#), ["Tests endpoints live"](#), ["Tests Prompt live"](#), ["Tests bus live front"](#), ["Tests bus live extension"](#), ["Tests Sidebar activité"](#), ["Tests cycle panel Prompt"](#), ["Implémentation onStreamed"](#), ["Survie stream sans webview"](#), ["Client stream webview"](#), ["Transport stream sans vue"](#), ["Dispatch du stream"](#), [extTypedMessageBus.util.ts](#) [L100-L220], [open.endpoint.ts](#), [abort.endpoint.ts](#), [awaitStop.endpoint.ts](#), [clearThreadMemory.endpoint.ts](#), [usePromptSessionHydration.hook.ts](#), ["Fin runtime Orch"](#), ["Finalisation runtime Orch"](#), ["Ordre terminal Orch"](#), [stream.endpoint.ts](#), [OrchAskStreamTurnRuntime.ts](#) [L640-L935], [PromptQueryDoneSound.manager.ts](#), [easyAgentsBatchQueryString.util.ts](#), [usePromptSessionLocator.hook.ts](#), ["Ouverture watcher Batch"](#), [EasyAgentsBatchLiveObserver.service.test.ts](#), [EasyAgentsBatchLiveOpen.service.test.ts](#), [EasyAgentsBatchViewerPresence.service.test.ts](#), [EasyAgentsBatchLastViewerStop.service.test.ts](#), [useAiStream.hook.easyAgentsBatchLiveObserver.test.ts](#), [SidebarActivityQuery.test.tsx](#), ["Identité tours Prompt"](#), ["Persistance continue retry"](#), ["Continue response handler"](#), ["Retry query handler"](#), [usePromptRunCommands.hook.ts](#) [L120-L320], [usePromptRunCommands.hook.ts](#) [L330-L470], [usePromptRunCommands.hook.ts](#) [L520-L660], [usePromptRunCommands.hook.ts](#) [L740-L865], [preparePromptStreamingTurn.util.ts](#), [usePromptRunController.hook.ts](#) [L450-L535], [usePromptRunController.hook.ts](#) [L535-L585], [AiQuery.type.ts](#), ["Ack fallback"](#)

[continue](#)",  ["Bootstrap fallback par Ai"](#),  [PromptThreadCleanup.manager.attachmentsCleanup.test.ts](#),  [openPanel.util.test.ts](#) [L230-L430],  [SidebarStore.manager.queryRunningUpsert.test.ts](#),  [SidebarStore.rehydrate.manager.test.ts](#),  [SidebarQueryRunningBridge.race.test.ts](#),  [VscButton.tsx](#) [L20-L180],  ["Icône SquareStop"](#),  ["Usages viewer runtime"](#),  [interactGate.programming-doc-readme.md](#)

Réponse corrigée en deux phrases

Tu as bien fait de challenger le **read-only** et le supposé **second bouton Stop** : le premier rapport avait mélangé la politique propre au watcher EasyAgents avec le comportement normal d'un Prompt AICode.

- Pour une session normale ouverte, **aucun mode read-only spécial ne doit durer pendant toute la réponse**. Seul le verrou d'envoi normal doit subsister tant que le stream est actif ; le compositeur doit revenir à son interface habituelle dès que le snapshot live est installé et que le rattachement au stream est armé.
- **Aucun nouveau bouton Stop ne doit être ajouté dans le Prompt**. Le bouton Stop normal existe déjà. Le seul nouveau bouton demandé est celui de la troisième ligne dans Activité, quand le panneau est fermé et que le processus continue.

Relecture approfondie effectuée

J'ai relu les chemins backend, frontend, panneau, sidebar, transport et hydratation, ainsi que les programming docs concernées :

- [architecture EasyAgents et contrat actuel du live watcher](#)
- [invariant streaming / réhydratation](#)
- [verrou de dispatch Queue B](#)
- [barrière fallback persisted](#)
- [infrastructure interactGate](#)

Cette nouvelle lecture révèle aussi un piège majeur que le premier rapport n'avait pas identifié : **continueResponse ne peut pas être rejoué comme un nouveau tour standard**. Il faudra un mode de replay distinct, car la continuation réutilise le même `turnId` et complète un message assistant déjà existant.

1. Architecture des nouveaux modules

Nom recommandé

Je retiens **liveSession** :

- suffisamment court ;
- immédiatement compréhensible ;
- plus large que **observer**, car le module gèrera aussi :
- l'enregistrement du run source ;
- le snapshot de base ;
- le replay ;
- le tail ;
- les leases de panneau ;
- l'ouverture live ;
- la rétention terminale ;
- l'arrêt protégé.

Le nom n'est actuellement utilisé nulle part dans le projet.

Emplacements retenus

Projet	Emplacement parent vérifié	Nouveau module proposé	Responsabilité
Extension	orchestrateur backend	liveSession/	Registre des runs, buffer, replay, snapshots live-safe, leases et composition OOP.
Frontend	Prompt frontend	liveSession/	Contrôleur de rattachement, bootstrap de replay, présentation des états transitoires, intégration Prompt.
Shared	types partagés	liveSessionTypes/	Identités, protocoles d'ouverture, types de replay, identifiants de lease et contrats d'arrêt.

Cette disposition suit les précédents du projet, notamment les trois couches de l'infrastructure InteractGate :

- [InteractGate backend](#)
- [InteractGate frontend](#)

- [InteractGate shared](#)

Organisation interne

Le module ne doit pas être un répertoire plat avec quinze fichiers côte à côte.

Backend OOP avec composition

Le module backend doit être décomposé par responsabilités, par exemple :

Sous-domaine	Responsabilité
core/	Façade stable et composition des services.
registry/	Records par run, index actif par locator, transitions atomiques.
source/	Enregistrement du run source, append des chunks, source settled.
open/	Résolution live / terminal / not-ready et production du snapshot live-safe.
replay/	Replay complet du backlog puis tail ordonné.
panel/	Leases exactes liées au cycle de vie d'un panneau Prompt.
retention/	Expiration des ouvertures abandonnées et éviction terminale.
__test__/	Tests unitaires ciblés ou sous-répertoires de tests proches des services.

La façade générique composerait plusieurs petits services, sur le modèle déjà utilisé

par [EasyAgentsBatchRunManager.service.ts](#).

EasyAgents ne conserverait ensuite que ses **adaptateurs métier** :

- critère `watchable` propre au Batch ;
- association parent / child / line number ;
- présence des panneaux item/history ;
- politique « zéro viewer = hard stop » ;
- barrière Git et parent Batch avant `handoffReady`.

Le buffer, l'ouverture live-safe, les leases et le replay ne resteraient plus dans EasyAgents.

Frontend découpé

Le frontend doit également être structuré :

Sous-domaine	Responsabilité
core/	Contrôleur de rattachement et état de la session live.
bootstrap/	Préparation du Prompt selon le type de replay.
components/	Composants unitaires pour les rares états transitoires ou erreurs de rattachement.
adapters/	Politique EasyAgents et politique Prompt normal.
__test__/	Tests de hooks, composants et transitions.

Le bloc viewer actuellement embarqué directement dans `PromptInput.tsx` pourra devenir un composant dédié, avec son propre module CSS si nécessaire.

Les Queues A et B ne doivent pas être déplacées dans `liveSession`

C'est une correction importante par rapport au résumé de la conversation.

- Queue A, dans [useAiStreamQueue.hook.ts](#), est déjà utilisée par les streams normaux.
- Queue B, dans [usePromptStreamDispatchGateQueue.hook.ts](#), est déjà un verrou générique du pipeline Prompt.
- Le watcher EasyAgents les utilise parce qu'il entre dans [useAiStream.hook.ts](#).

Les déplacer sous `liveSession/` ferait croire que le transport Prompt normal dépend d'une fonctionnalité de reconnexion. Ce serait une mauvaise inversion de responsabilité.

La bonne extraction est donc :

- déplacer dans `liveSession/` **l'intégration qui rattache un replay au pipeline existant** ;
- garder Queue A et Queue B dans leur module de transport actuel ;
- documenter explicitement qu'elles sont des dépendances obligatoires du module `liveSession`.

C'est exactement l'application de ta règle : **ne pas regrouper ce qui ne doit pas être regroupé**.

2. Découpage en deux tranches

Je conserve totalement cette décision.

Tranche 1 — Extraction générique sans modification fonctionnelle

Cette tranche doit garantir une stricte invisibilité fonctionnelle :

- mêmes boutons EasyAgents ;
- mêmes textes ;
- même critère d'apparition de `Watch live` ;
- même panneau viewer ;
- même replay ;
- même terminal handoff ;

- même politique de hard stop au dernier viewer ;
- mêmes durées d'ouverture et de rétention ;
- mêmes transitions Batch/Git ;
- aucune modification de comportement pour les sessions normales ;
- aucune nouvelle troisième ligne dans Activité ;
- aucune modification fonctionnelle du nettoyage des panneaux normaux.

Les modifications internes suivantes sont compatibles avec cette contrainte :

- identité `locator + turnId` ;
- remplacement de `viewerAttached` par des leases ;
- déplacement des services dans le module générique ;
- remplacement des types et canaux nommés Batch par des contrats génériques ;
- création du module frontend et du module shared ;
- migration de la documentation ;
- nouveaux tests de courses et d'invariants.

EasyAgents est ensuite rebranché sur le nouveau module par des adaptateurs fins.

Tranche 2 — Sessions normales et Activité

Seulement après validation de la tranche 1 :

- enregistrement des streams Prompt normaux dans `liveSession` ;
- fermeture d'un panneau actif = détachement, plus abort automatique ;
- rattachement depuis Activité ;
- évolution du SidebarStore ;
- troisième ligne avec Stop ;
- arrêt protégé par `locator + turnId` ;
- nettoyage mémoire après fin si aucun panneau n'est ouvert ;
- réouverture normale immédiate dès que l'état frontend est sûr.

3. Tout le code doit être testé

Oui. Je retiens cette règle comme critère d'acceptation dur.

Il ne suffira pas de déplacer les tests actuels : plusieurs invariants importants ne sont actuellement pas testés.

Tests de la tranche 1

Zone	Tests obligatoires
Identité de run	Deux tours successifs sur le même locator ; un événement tardif de l'ancien tour ne modifie jamais le nouveau.
Registre générique	Création, réutilisation cohérente, collision de <code>turnId</code> , éviction, rétention terminale.
Buffer	Ordre exact des chunks, absence de duplication, conservation avant attachement.
Replay + tail	Backlog complet puis nouveaux chunks ; fermeture uniquement à <code>handoffReady</code> .
Leases de panneau	Deux leases indépendantes, release exacte, release obsolète, ouverture abandonnée, TTL.
Ouverture live-safe	Snapshot live, terminal, not-ready, not-found.
Replay de nouveau tour	Le message utilisateur est conservé ; seuls les assistants du tour courant sont reconstruits.
Replay de continuation	Le message assistant existant est conservé comme préfixe et reçoit uniquement la continuation.
EasyAgents adapter	Critère durable + token/thinking/raw-live inchangé.
EasyAgents policy	Dernier viewer perdu = même hard stop qu'aujourd'hui.
Handoff Batch	Git et état parent convergent toujours avant <code>handoffReady</code> .
Retry Batch	L'ancien record live est libéré puis évincé avant suppression du child.
Transport extension	Relay sans transformation, erreur, fin normale.
Transport frontend	Le replay générique traverse bien Queue A puis Queue B.
Prompt EasyAgents	Même banner, même Stop line, même handoff et même retry de handoff.
Documentation	Pas de test automatique du Markdown, mais aucune duplication des sources de vérité.

Tests de la tranche 2

Zone

Fermeture en cours de stream
Fermeture idle
Réouverture en cours
Réouverture après la fin
Fermeture puis réouverture puis fermeture

Fin pendant la réouverture

Activité

Panneau rouvert

Fin normale panneau fermé

Stop Activité

Stop obsolète

Stop ancien tour

Double source d'activité

Sidebar rehydrate

`continueResponse`

`interactGate`

Grouped file edits

Isomorphisme

Tests obligatoires

Aucun abort, aucun clear mémoire, run toujours actif.

Nettoyage historique inchangé.

Snapshot live-safe, replay du backlog, puis tail.

Ouverture classique, sans tentative de watcher.

Les leases successives sont indépendantes.

Soit live attach valide, soit fallback terminal valide, jamais un état intermédiaire incohérent.

Troisième ligne uniquement si panneau fermé + run actif.

Troisième ligne supprimée immédiatement.

Passage à Idle et disparition de la troisième ligne.

Désactivation immédiate, une seule requête acceptée.

Aucun abort si le panneau a été rouvert.

Aucun abort si un nouveau `turnId` a remplacé l'ancien.

Le viewer reconnecté ne redémarre ni le timer ni le son de fin.

Les runs détachés actifs sont restaurés depuis le registre live.

Préfixe assistant intact, aucun texte dupliqué.

Une gate active réapparaît après replay et reste résoluble.

Même état final entre stream original et session rattachée.

Résultat React final identique entre stream continu et fermeture/réouverture.

Les tests frontend devront suivre tes règles existantes :

- actions dans `act(...)` ;
- pas de sélection fragile par label ;
- test IDs ou rôles stables ;
- mocks Jest avec une seule signature fonctionnelle ;
- aucun `any`, aucun cast inutile.

4. Migration du programming doc existant

Tu as raison : le système est déjà documenté.

La documentation actuelle n'est cependant pas un fichier autonome consacré uniquement au watcher. C'est une grande section de [easyAgents.programming-doc-readme.md](#), intitulée **Live Prompt watcher contract**.

La bonne opération est donc :

1. créer `liveSession.programming-doc-readme.md` à la racine du futur module `liveSession/` sous [Orch](#) ;
 2. déplacer et généraliser dans ce document :
 - le record live ;
 - les identités de run ;
 - les snapshots live-safe ;
 - le replay/tail ;
 - les leases ;
 - les barrières source terminal / handoff ready ;
 - les politiques d'ouverture ;
 - l'intégration frontend avec le pipeline Prompt ;
 3. retirer de la grande doc EasyAgents la description du mécanisme générique ;
 4. y conserver uniquement :
 - le critère EasyAgents de watchabilité ;
 - l'association parent/child/ligne ;
 - la politique du dernier viewer ;
 - l'ordre Git/parent/handoff ;
 - une référence vers la nouvelle doc ;
 5. mettre à jour le titre de la doc EasyAgents pour ne plus prétendre être la source de vérité du mécanisme générique.
- Les docs suivantes restent à leur emplacement actuel :
- [Queue B](#)

- [isomorphisme hydration](#)

Elles documentent des sous-systèmes génériques déjà existants et ne doivent pas être absorbées artificiellement dans `liveSession`.

5. Le mode read-only : nécessité réelle et durée

Ce que fait réellement le Prompt normal aujourd'hui

Le Prompt normal bloque l'envoi pendant une génération :

[AICode/src/frontend/src/app/prompt/Prompt/usePromptRunUiDerived.hook.ts](#)

Copy

```
const isPromptLocked: boolean = isStreaming || uiPending;

// Combined running flag for the input (UI pending OR true streaming)
const isRunningForInput: boolean = uiPending || isStreaming;
```

Mais ce verrou n'empêche pas l'utilisateur d'écrire son prochain brouillon. Dans le composant actuel :

[AICode/src/frontend/src/app/prompt/PromptInput/PromptInput.tsx](#)

Copy

```
submitEnabled={
  !isLocked &&
  hasSubmitPayload &&
  !hasBlockingAttachmentWork &&
  !effectiveSubmitTxnPending
}
editEnabled={
  !indexPendingVisible && !effectiveSubmitTxnPending
}
```

`isLocked` désactive donc **Submit**, pas l'édition.

Pendant une génération normale :

- le compositeur reste affiché ;
- l'utilisateur peut préparer le prochain message ;
- le bouton Send devient le bouton Stop ;
- Ctrl/Cmd+Enter ne peut pas envoyer un second tour ;
- le backend ne reçoit pas deux streams concurrents sur le même thread.

Ce que fait le watcher EasyAgents

Le viewer EasyAgents applique une autre politique :

- il cache complètement le compositeur ;
- il affiche un banner ;
- il expose son bouton Stop line.

[AICode/src/frontend/src/app/prompt/PromptInput/usePromptInputViewerMode.hook.ts](#)

Copy

```
case 'batchLiveStreaming':
  return {
    isViewerMode: true,
    showComposer: false,
    showViewerStopAction: true,
    showViewerRetryAction: false,
```

Cette politique est correcte pour EasyAgents et doit rester inchangée en tranche 1.

Elle ne doit pas être copiée telle quelle vers le cas normal.

Politique corrigée pour une session normale rouverte

Phase	Compositeur	Envoi	Stop
Chargement du snapshot live-safe	Temporairement indisponible	Bloqué	Rattachement encore en préparation
Snapshot appliqué + listener de replay armé	Interface Prompt normale	Bloqué par <code>isStreaming</code>	Bouton Stop normal
Backlog en cours de rattrapage	Interface Prompt normale	Bloqué par <code>isStreaming</code>	Bouton Stop normal
Tail live	Interface Prompt normale	Bloqué par <code>isStreaming</code>	Bouton Stop normal
Source terminée et files A/B drainées	Interface Prompt normale	Réactivé	Send normal

Le retour à l'interface normale ne doit donc **pas attendre la fin de la réponse**.

Il peut avoir lieu dès que :

1. l'ouverture live-safe a retourné le snapshot et l'identité du run ;
2. le snapshot a été appliqué ;
3. les placeholders ou la cible de continuation ont été installés ;
4. le stream de replay est armé ;
5. `isStreaming` est actif, garantissant que l'envoi d'un nouveau tour reste impossible.

Il n'est pas nécessaire d'attendre que tout le backlog soit rejoué. Queue A et Queue B garantissent déjà l'ordre.

Verrou supplémentaire réellement nécessaire

Il reste un petit verrou transitoire nécessaire pendant le bootstrap.

Sans ce verrou, il pourrait exister un render où :

- le snapshot vient d'être installé ;
- le listener n'est pas encore actif ;
- `isStreaming` vaut encore `false` ;
- Submit redevient brièvement disponible.

Le module frontend devra donc avoir un état explicite du type `bootstrapPending`, intégré au verrou d'envoi jusqu'à l'activation du stream.

C'est un verrou de quelques transitions React, pas un mode read-only pendant toute la réponse.

Faut-il un rechargement terminal pour le cas normal ?

Le watcher EasyAgents en a besoin parce que la fin du stream Orch n'est pas la fin de la transaction Batch :

- Git doit converger ;
- les métadonnées child sont patchées ;
- le parent Batch est mis à jour ;
- l'état UI Batch est publié.

Pour une session normale, ces étapes externes n'existent pas.

Le runtime Orch :

1. termine le regroupement ;
2. persiste les métadonnées ;
3. émet le terminal meta en dernier ;
4. nettoie les intégrations ;
5. libère le contrôleur et la clé in-flight ;
6. ferme ensuite le stream source.

Par conséquent, **le rechargement final par le endpoint classique n'est pas intrinsèquement obligatoire pour une session normale**. La mémoire Orch existante n'a jamais été réinitialisée et reste déjà la mémoire correcte du thread.

La cible optimale est donc :

- replay complet ;
- attente de la vraie fermeture du source stream ;
- drainage Queue A/Queue B ;
- passage immédiat à l'état normal ;
- aucun `/chat/open` terminal supplémentaire.

Ce point devra être prouvé par des tests d'isomorphisme complets. Si ces tests révèlent qu'une structure frontend temporairement tolérante ne converge pas exactement vers la structure hydratée stricte, alors seulement il faudra conserver une courte phase de réconciliation terminale.

Même dans ce fallback :

- le compositeur resterait affiché ;
- seul Submit serait bloqué ;
- le brouillon local et les attachments seraient préservés ;
- le verrou serait levé immédiatement après la réconciliation.

Conclusion sur le read-only

- **EasyAgents** : read-only complet inchangé jusqu'au handoff, tranche 1 oblige.
- **Session normale ouverte** : pas de read-only complet.
- **Bootstrap normal** : verrou d'envoi transitoire nécessaire.
- **Pendant toute la réponse** : verrou d'envoi standard `isStreaming`, identique à une session normale restée ouverte.
- **Après la réponse** : déverrouillage dès la convergence réelle ; idéalement sans reload terminal.

6. Le supposé second bouton Stop

Le code actuel contient déjà deux présentations adaptées à deux UX existantes

1. Dans une session normale :
 - le bouton Send de [VscTextarea.tsx](#) devient le bouton Stop quand `running=true`.
 2. Dans le viewer EasyAgents :
 - [PromptInput.tsx](#) affiche déjà `Stop line` dans son banner.
- Le rapport précédent disait qu'une session normale reconnectée aurait « un bouton Stop dans le Prompt ». Cette formulation pouvait faire croire à un ajout.

Décision corrigée

Aucun nouveau bouton Stop ne doit être créé dans le Prompt normal.

Pour une session normale reconnectée :

- on réutilise le compositeur normal ;
- il reçoit `running=true` ;
- son bouton existant passe de Send à Stop ;
- son handler existant appelle l'arrêt Orch ;
- le bouton redevient Send après la fin.

Pour EasyAgents :

- le bouton `Stop line` existant reste inchangé en tranche 1.

Le seul bouton réellement nouveau est :

- troisième ligne de l'item Activité ;
- uniquement si le panneau est fermé et le run toujours actif ;
- `SquareStop` ;
- taille `small` ;
- variante `danger` ;
- `disableAfterClick` ;
- validation backend par `locator + turnId`.

[VscButton.tsx](#) supporte déjà `size="small"`, `variant="danger"` et `disableAfterClick`. [VscIcon.tsx](#) expose déjà `SquareStop`.

Aucun changement du design system n'est requis.

7. Le panneau est ouvert ou fermé

Tu as raison. Je retire les notions de :

- vue invisible ;
- vue cachée ;
- viewer non visible ;
- panneau en arrière-plan assimilé à fermé.

Le contrat ne possède que deux états métier :

État

open Le panneau Prompt existe dans le registre des panneaux VS Code.

closed Aucun panneau Prompt n'existe pour ce locator.

Le champ `panel.active` de VS Code sert uniquement à la sélection/focalisation. Il ne doit jamais décider :

- de la survie du processus ;
- de l'affichage du bouton Stop Activité ;
- de la possession du run.

Un panneau ouvert mais non sélectionné reste **ouvert**.

Transition retenue dans Activité

Je retire également l'inférence antérieure selon laquelle l'item devait forcément être supprimé lorsque le run terminait avec le panneau fermé.

La lecture la plus fidèle de ta demande est :

- panneau fermé + run actif ;
- trois lignes ;
- état Running ;
- elapsed ;
- bouton Stop ;
- run terminé ;
- retour à deux lignes ;
- état Idle ;
- Last query ;
- aucune troisième ligne ;
- panneau rouvert pendant le run ;
- deux lignes ;
- état Running ;
- aucune troisième ligne.

L'item n'est pas forcé de disparaître simplement parce que le panneau est fermé. Le SidebarStore devient précisément un store d'**activité de session**, plus seulement un miroir des fenêtres ouvertes.

8. Évolution du SidebarStore

Le contrat actuel est nommé `SidebarActivityQueryWindowItem`, et le tableau s'appelle `windows`. Ce vocabulaire devient incorrect dès qu'une session peut rester dans Activité sans panneau.

Je recommande une vraie évolution sémantique :

- `SidebarActivityQueryWindowItem` → `SidebarActivityQueryItem` ;
- `queryState.windows` → `queryState.items` ;
- ajout de `panelState: 'open' | 'closed'` ;
- ajout de `activeTurnId?: string` ;
- ajout de `stopRequested: boolean`.

Projection recommandée

Champ	Rôle
<code>locator</code>	Identité exacte de la session.
<code>title</code>	Titre affiché.
<code>isRunning</code>	Processus source actif.
<code>panelState</code>	Panneau ouvert ou fermé.
<code>activeTurnId</code>	Run exact actuellement actif.
<code>stopRequested</code>	Désactivation persistante du bouton après acceptation.
<code>liveQueryElapsedMs</code>	Temps courant.
<code>lastQueryElapsedMs</code>	Durée terminale.
<code>dates existantes</code>	Tri UI et stabilité.

La condition du bouton devient simplement :

- `isRunning === true`
- `panelState === 'closed'`

Et son état `enabled` :

- `stopRequested === false`

La sidebar n'est pas l'autorité de sécurité

Le SidebarStore n'est qu'une projection UI.

L'endpoint Stop doit valider dans le registre `liveSession` :

1. le `locator` existe ;
2. le `turnId` correspond encore au run actif ;
3. le run appartient à une session normale détachable ;
4. le panneau est toujours fermé ;
5. le run n'est pas déjà terminal ;
6. aucun stop n'a déjà été accepté.

Cette validation évite :

- un clic arrivé après la réouverture ;
- un clic arrivé après la fin ;
- un clic appartenant à un ancien tour ;
- un double clic depuis deux sidebars.

Ouverture depuis Activité

`SidebarActivityQuery.tsx` ne devrait pas choisir lui-même entre ouverture classique et ouverture live à partir d'un état potentiellement périmé.

Il doit envoyer le locator comme aujourd'hui.

L'extension décide ensuite :

- panneau déjà ouvert : reveal ;
- aucun run actif : ouverture classique ;
- run actif détaché : ajout automatique du mode live dans la query string.

La décision reste ainsi près de la source de vérité.

Pas de double comptage d'activité

Le stream normal d'origine possède déjà :

- le timer Activité ;
- l'elapsed ;
- le son de fin ;
- la transition Running → Idle.

Le stream de replay d'un panneau reconnecté ne doit surtout pas rappeler les mêmes hooks.

C'est un défaut potentiel du handler EasyAgents actuel s'il était réutilisé aveuglément : il appelle lui-même le bridge d'activité, parce que le child Batch source n'est pas un stream Prompt normal.

Le module générique doit donc séparer :

- **run source** : possède Activité et le son ;
- **consumer de replay** : reçoit uniquement les chunks.

Pour EasyAgents, l'adaptateur conservera la possession viewer actuelle afin de ne rien modifier en tranche 1.

Pour les sessions normales, seul le handler source reste propriétaire d'Activité.

9. Buffer raw-live

Je retire la taille du buffer raw-live de la liste des risques dimensionnants.

Tu as raison :

- ce sont essentiellement des chaînes ;
- le mécanisme fonctionne déjà pour EasyAgents ;
- le même modèle peut fonctionner pour les sessions normales ;
- il ne faut pas complexifier le système prématurément pour économiser ce buffer.

Je maintiens seulement les règles de cycle de vie, non comme optimisation mémoire, mais comme protection de cohérence :

- un ancien tour doit être évincé correctement ;
- un retry destructif ne doit pas laisser un buffer réouvrable ;
- une fin tardive d'un ancien tour ne doit pas toucher le nouveau ;
- une session supprimée ne doit pas laisser de record orphelin.

Ce sont des invariants de correction, pas une inquiétude sur le poids des strings.

10. Nouveaux pièges identifiés par la seconde analyse

10.1. `continueResponse` exige un replay différent

C'est le principal ajout par rapport au premier rapport.

Les différents types de requêtes n'ont pas tous le même bootstrap :

Type de requête	Snapshot de base	Reconstruction frontend
prompt	Snapshot durable après <code>fallback_persisted</code>	Conservé le user, retirer les assistants du tour, recréer think + assistant.
compactContextNow	Même principe	Même reconstruction de nouveau tour.
specifAction	Même principe, avec user structuré persistant	Même reconstruction de nouveau tour.
retryQuery	Snapshot persistant du tour rejoué	Conservé le user, retirer l'ancien assistant, recréer les placeholders.

Type de requête	Snapshot de base	Reconstruction frontend
<code>editLastPrompt</code>	Snapshot après persistance de l'édition	Conserver le nouveau user, retirer l'ancien assistant, recréer.
<code>continueResponse</code>	Snapshot figé avant le début de la continuation	Conserver l'assistant existant et ajouter les nouveaux chunks à son texte.

`continueResponse` :

- réutilise le même `turnId` ;
- ne produit pas `fallback_persisted` ;
- n'ajoute pas un nouveau user ;
- utilise le texte assistant existant comme `baseAssistantTextRef` ;
- cible le même assistant au lieu d'en créer un nouveau.

Le generic live open doit donc retourner un `replayKind` explicite, au minimum :

- `newTurn`
- `continueResponse`

Pour la continuation, le backend doit capturer un snapshot de base avant que le stream ne commence à modifier l'assistant persistant.

Sans cela, une réouverture pourrait :

- supprimer le préfixe assistant original ;
- dupliquer une continuation déjà partiellement persistée ;
- recréer un nouvel assistant au lieu de compléter l'existant.

10.2. durable et watchable ne sont pas universels

EasyAgents exige aujourd'hui :

- `fallback_persisted` ;
 - puis au moins un token, thinking token ou raw-live chunk.
- Cette règle est une politique Batch, pas le cœur générique.

Pour une session normale :

- un nouveau prompt peut être rouvert dès `fallback_persisted`, même si aucun token visible n'est encore arrivé ;
- une continuation peut être rouverte immédiatement après capture de son baseline, puisqu'elle est déjà fondée sur une session persistée.

Le cœur doit donc exposer des primitives telles que :

- `baseline ready` ;
- `chunk replayable reçu` ;
- `source active` ;
- `source settled` ;
- `handoff ready`.

Chaque propriétaire décide ensuite si `openLive` est autorisé.

10.3. source settled et handoffReady restent séparés

La séparation reste nécessaire dans le module générique, mais les politiques diffèrent :

Propriétaire	handoffReady
---------------------	---------------------

Session normale Peut coïncider avec la fin réelle du stream source, après finalisation Orch.

EasyAgents Arrive plus tard, après Git, patch child, parent Batch et publication UI.

Ainsi, la session normale ne doit jamais attendre artificiellement une barrière conçue pour Git/Batch.

10.4. Nettoyage du panneau à décomposer

[PromptThreadCleanup.manager.ts](#) mélange actuellement :

- suppression du mapping ;
 - Abort ;
 - AwaitStop ;
 - ClearMemory ;
 - nettoyage des attachments orphelins.
- [openPanel.util.ts](#) ajoute encore :
- release du viewer EasyAgents ;
 - suppression de l'item Activité ;
 - trivial Untitled cleanup ;
 - cleanup du son.

Pour un panneau fermé pendant un run normal, on ne peut pas simplement « sauter Abort » et exécuter aveuglément le reste.

La fermeture active doit devenir :

1. marquer le panneau fermé ;
2. libérer sa lease exacte ;
3. conserver le run et sa mémoire ;
4. conserver l'item Activité ;
5. différer les nettoyages susceptibles de courir contre la persistance active.

Après la fin :

- si aucun panneau ne s'est rouvert :
 - clear mémoire ;
 - nettoyage opportuniste des attachments ;
 - cleanup trivial si pertinent ;
 - si un panneau est rouvert :
 - conserver la mémoire Orch prête pour le prochain message.
- La course fin/réouverture doit être sérialisée par le registre ou par une petite file par locator.

10.5. Les leases doivent représenter le vrai panneau

Je recommande un `panelInstanceId` généré côté extension pour chaque création réelle de panneau Prompt.

Il doit être distinct :

- du locator ;
- du `turnId` ;
- du channel de stream.

Le même identifiant accompagne :

- la query string du panneau ;
- l'ouverture live ;
- l'attachement au replay ;
- le dispose du panneau.

Ainsi :

- le dispose du panneau A ne peut pas libérer le panneau B ;
- une ouverture abandonnée expire proprement ;
- une release tardive est sans effet ;
- l'adaptateur EasyAgents sait exactement quel panneau comptait comme viewer.

10.6. L'interact gate doit survivre à la fermeture

Un tour Orch peut être suspendu dans une gate sans timeout, notamment MCP.

Si le panneau est fermé :

- le processus continue d'exister ;
- le run reste Running dans Activité ;
- Stop Activité doit pouvoir l'aborder ;
- une réouverture doit rejouer `interact_gate_required` ;
- le nouveau frontend doit reconstruire l'UI inline ;
- la décision doit résoudre la gate backend déjà existante.

Le buffer complet des chunks permet ce comportement, mais il faut un test d'intégration dédié.

10.7. Le transport continue bien sans webview

La faisabilité fondamentale reste confirmée par le code :

- le handler de stream est exécuté côté extension ;
- le démontage React ne possède aucun mécanisme d'annulation du handler ;
- l'échec de `webview.postMessage` est capturé dans [extMessageBus.util.ts](#) ;
- le handler continue à consommer le endpoint ;
- le replay peut donc être alimenté après la fermeture.

Le kill vient bien de la politique explicite de [PromptThreadCleanup.manager.ts](#), pas du transport.

10.8. Le Stop Activité doit rester distinct du Stop Prompt

- Stop Prompt : contrôle normal du panneau ouvert.
- Stop Activité : contrôle exceptionnel d'un run dont le panneau est fermé.

Ils appellent le même mécanisme d'abort Orch, mais le Stop Activité possède des gardes supplémentaires :

- panneau toujours fermé ;
- `turnId` toujours actif ;
- requête détachable ;

- stop pas déjà demandé.

Machine d'état générique révisée

État	Signification
<code>sourceRegistered</code>	Run créé avant le premier <code>await</code> , avec <code>locator</code> , <code>turnId</code> et <code>replayKind</code> .
<code>baselinePending</code>	Le snapshot nécessaire au replay n'est pas encore sûr.
<code>baselineReady</code>	Snapshot live-safe disponible.
<code>sourceStreaming</code>	Chunks en cours d'ajout au buffer.
<code>sourceSettled</code>	Le stream Orch a complètement terminé, nettoyage compris.
<code>handoffReady</code>	Le propriétaire autorise la fin du replay et le retour à son état terminal.
<code>retained</code>	Record terminal conservé temporairement pour résoudre les ouvertures concurrentes.
<code>evicted</code>	Record supprimé.

Les leases de panneau sont orthogonales à ces états.

Politique normale

- fermeture du dernier panneau : aucune modification du run source ;
- `handoffReady` suit immédiatement `sourceSettled` ;
- panneau rouvert : surface Prompt normale dès le bootstrap ;
- envoi bloqué seulement par le run actif ;
- fin du run : déverrouillage immédiat après drainage frontend.

Politique EasyAgents

- fermeture du dernier viewer global : hard stop ;
 - `handoffReady` après convergence Batch externe ;
 - surface viewer read-only inchangée ;
 - rechargement terminal strict inchangé.
-

Révision de la surface du chantier

Avec :

- trois modules réellement organisés ;
- extraction du comportement `continueResponse` ;
- leases exactes ;
- tests de chaque service, hook et composant ;
- adaptation `SidebarStore` ;
- tests des courses ;
- migration documentaire ;

l'ancienne estimation de **35 à 50 fichiers pour l'ensemble** était probablement trop basse.

Estimation réaliste sans prétendre fixer la liste avant spécification :

Tranche	Surface estimée
Extraction générique sans changement fonctionnel	Environ 25 à 40 fichiers créés, déplacés ou modifiés, tests compris
Adoption normale + Activité	Environ 20 à 30 fichiers supplémentaires
Ensemble	Environ 45 à 70 fichiers

Les fichiers `*.generated.*` ne sont évidemment pas comptés comme fichiers à éditer. Seules les sources génératrices et leurs call sites sont concernés.

Risque réévalué

Le risque reste élevé, mais mieux localisé.

Risques principaux réels

1. `continueResponse` et son assistant de base.
2. Fin du source contre réouverture simultanée du panneau.
3. Ancien `turnId` contre nouveau tour sur le même `locator`.
4. Double possession du timer Activité et du son.
5. Détachement du panneau contre nettoyage mémoire.
6. Gate interactive active pendant la fermeture.
7. Reproduction exacte des stores frontend par replay.
8. Release tardive d'une ancienne instance de panneau.

9. Retry/edit qui réutilisent un `turnId`.
10. Handoff EasyAgents externe au stream Orch.

Point retiré des risques majeurs

- volume du buffer raw-live.

Contrôle du risque

- deux tranches strictes ;
- tranche 1 sans changement visuel ni comportemental ;
- machine d'état explicite ;
- identité `locator + turnId` ;
- leases exactes ;
- source d'activité unique ;
- tests d'isomorphisme ;
- tests de courses avec ordres inversés ;
- aucun raccourci ni fallback de compatibilité non demandé.

Corrections apportées au premier rapport

Sujet	Première analyse	Analyse corrigée
Architecture	Module principalement backend	Un module structuré dans extension, frontend et shared.
Queues A/B	Présentées comme faisant partie de l'extraction	Restent dans le transport générique ; seule leur intégration live est extraite.
Read-only normal	Maintien jusqu'au handoff terminal	Aucun read-only spécial pendant toute la réponse ; surface normale après bootstrap.
Compositeur normal	Considéré comme non interactif	Il reste éditable, seul Submit est verrouillé.
Stop Prompt	Formulation pouvant faire croire à un nouveau bouton	Aucun nouveau bouton ; réutilisation du Stop normal existant.
État du panneau	Terminologie floue de visibilité	Seulement <code>open</code> ou <code>closed</code> .
Item après fin	Suppression supposée	Retour à deux lignes Idle/Last query, sans troisième ligne.
Rechargement terminal normal	Considéré implicitement obligatoire	Probablement évitable ; doit être tranché par tests d'isomorphisme.
Types de run	Tous traités comme un nouveau tour	<code>continueResponse</code> exige un baseline et un replay spécifiques.
Buffer raw-live	Présenté comme un risque mémoire	Retiré des risques dimensionnants.
Documentation	Création nouvelle évoquée	Extraction réelle de la section existante puis redistribution des responsabilités.
Estimation	35–50 fichiers	45–70 fichiers pour les deux tranches avec architecture et tests complets.

TL-DR

- Le module s'appellera **liveSession**, avec un répertoire propre dans l'extension Orch, le frontend Prompt et les types shared.
- Le backend sera OOP et composé ; le frontend sera réparti entre hooks, bootstrap et composants unitaires.
- Queue A et Queue B restent dans le transport Prompt actuel : elles sont déjà génériques.
- La première tranche extrait le système sans aucune modification fonctionnelle EasyAgents.
- Tout comportement créé ou modifié devra être couvert par des tests.
- La documentation live actuellement contenue dans la grande doc EasyAgents sera extraite vers la nouvelle programming doc Orch.
- Une session normale rouverte ne doit pas rester dans le viewer read-only EasyAgents.
- Après le bootstrap, elle reprend immédiatement la surface Prompt normale :
 - compositeur visible et éditable ;
 - Submit bloqué par `isStreaming` ;

- bouton Stop normal existant.
- Aucun second bouton Stop n'est ajouté dans le Prompt.
- Le seul nouveau bouton est celui de la troisième ligne Activité lorsque le panneau est fermé et le run actif.
- Les seuls états du panneau sont `open` et `closed`.
- Le SidebarStore doit devenir un store d'activité de session et non plus un miroir exclusif des fenêtres.
- Le nouveau piège majeur est `continueResponse`, qui nécessite un replay par continuation du même assistant.
- Le buffer raw-live n'est plus considéré comme un risque dimensionnant.
- Le chantier reste risqué, mais les risques sont maintenant mieux identifiés et isolables par les deux tranches.

</ASSISTANT>

<USER>

La tranche 1 a été spécifiée et implémentée totalement. Ça a été fait dans une autre session et dans le commit f65adc8e5172cc1e4f9781bf54ffa6dd089d3d33. Je te montre le texte de la spécification. Vérifie la pour vérifier que ça correspond bien et que la tranche 2 peut débiter. Si non, dis ce qui ne vas pas. Si oui, analyse la faisabilité de la tranche 2 à partir de ce point. <SPECIF_EXTRAIRE_LE_LIVE_WATCHER_EASYAGENTS_TRANCHE_20260711_152702> ##

Spécification — Extraire le live watcher EasyAgents dans `liveSession` — tranche 1 ## Contexte Le live watcher permet d'ouvrir dans un panneau Prompt un child EasyAgents Batch dont la génération est déjà en cours. Il combine actuellement : conservation ordonnée des `AiStreamedChunk`, barrière `fallback\_persisted`, critère de watchabilité, snapshot live-safe sans reset mémoire Orch, retrait des messages assistant du tour actif, replay backlog + tail, séparation `sourceSettled` / `handoffReady`, présence viewer, handoff terminal strict et réutilisation de Queue A / Queue B. Son cœur doit devenir générique afin de pouvoir servir ultérieurement aux sessions AICode normales rouvertes pendant une génération. Cette tranche est exclusivement architecturale : extraire un module générique, rebrancher EasyAgents dessus et préserver le comportement observable. Elle ne modifie ni la fermeture des sessions normales, ni Activité, ni le SidebarStore, ni `PromptThreadCleanup`, ni les algorithmes Queue A / Queue B. La relecture fraîche du code confirme les invariants finaux suivants : - le cœur est extractible sans modifier l'orchestrateur de génération ; - Queue A, Queue B, `PromptStreamLockStore` et les sanitizers restent dans leur sous-système ; - un run est identifié partout par le locator complet + `turnId`, y compris dès le bootstrap Prompt ; - le Batch expose donc un `liveWatchTurnId` live-only strictement corrélé à `liveWatchLineNumber`, afin que le panneau transporte l'identité du run attendu au lieu de résoudre implicitement « le run actuellement actif » ; - `viewerAttached` est remplacé par des leases exactes ; - l'attente actuelle `EventEmitter.once('changed')` possède une fenêtre de réveil manqué ; - « aucun viewer restant = hard stop » demeure une policy EasyAgents ; - le bouton Stop existant reste le seul bouton Stop, mais son transport live doit devenir exact locator + turn + lease : l'abort thread-only actuel pourrait sinon arrêter un nouveau run du même locator à cause d'un ancien panneau ; - le buffer raw-live reste complet, ordonné, non compressé et non tronqué ; chaque consumer reçoit seulement une copie readonly de son suffixe non consommé afin d'éviter un replay quadratique ; - Slimbk impose des `ZodObject` pour `argsSchema` et `respSchema` : l'union canonique discriminée et l'objet de transport doivent rester distincts mais strictement corrélés ; - `sourceSettled` doit être signalé exactement une fois dès que `askStream(...)` a retourné un stream, y compris si l'enregistrement initial, l'itération, le parsing ou un callback lève ; - le `Readable` retourné par `askStream(...)` est lazy : le child executor doit armer son itérateur et conserver la première lecture pendant avant d'appeler le callback source-start synchrone, sans interpréter ni livrer le premier chunk avant l'enregistrement ; sinon un échec de source-start laisserait le générateur Orch non démarré et son `inFlightKey` sans finalisation ; - en cas d'échec après le retour de `askStream(...)` , l'itérateur/Readable armé doit être fermé ou détruit puis attendu best-effort, en plus de l'abort thread, avant de signaler le settlement ; la première lecture pendante et les erreurs de fermeture ne doivent jamais devenir des rejets non observés ; - seul `registerRun` crée un record ; `appendChunk`, `markBaselineReady`, `markSourceSettled` et `markHandoffReady` ne doivent jamais recréer implicitement un ancien run évincé ; - l'enregistrement du run a lieu immédiatement après le retour réussi de `askStream(...)` et avant l'interprétation ou la livraison du premier chunk ; si `askStream(...)` lève avant de retourner, aucun record ni settlement n'est créé ; - le callback source-start est synchrone et ne fait que l'enregistrement ; le sink EasyAgents conserve localement si cet enregistrement a réellement réussi, afin que le callback source-settled soit toujours invoqué une fois mais ne puisse jamais solder un record préexistant lorsque source-start a échoué ; - une lease acquise avant le snapshot reste `opening` si le snapshot manque ou échoue ; dispose, TTL ou handoff restent propriétaires de son retrait ; - après toute lecture asynchrone de snapshot, l'ouverture revalide le run exact, la lease, l'état d'éviction et une époque monotone du locator ; une bascule ABA « ancien run → nouveau run → plus de run » ne peut donc pas valider un résultat obsolète ; - après handoff, les leases actives restent corrélées et replay-authorized, ne comptent plus comme viewers et ne sont plus invalidées par le TTL `opening` ; - le dispose Prompt poursuit toujours son nettoyage même si la release EasyAgents échoue ; - le frontend exige un vrai terminal meta du tour exact puis `isStreaming=false` avant handoff ; `meta.stop\_reason` avec `reason=TOOL\_USE` est explicitement non terminal dans le pipeline Orch et ne doit donc ni activer `hasTerminalMeta`, ni armer le latch frontend ; - un snapshot terminal persistant n'est valide que si le dernier message assistant du `turnId` demandé porte un terminal hydratable : erreur persistée, ou

`stopReason` différent de `TOOL\_USE`, avec un `stopReasonTimeoutMs` valide pour `TIMEOUT` ; la simple présence d'un message du tour ne suffit pas ; - si le replay atteint `handoffReady` sans `meta.error` ni `meta.stop\_reason` dont la raison est différente de `TOOL\_USE`, il draine le buffer puis échoue ; `useAiStream.onError` injecte alors le `meta.error` terminal sans modifier le buffer source ; ce meta frontend autorise la tentative de handoff mais ne remplace pas l'exigence d'un snapshot terminal persistant exact ; - le résultat `terminal` de l'endpoint liveSession transporte directement le snapshot terminal exact, `threadId` et `turnId` ; le frontend hydrate ce snapshot strictement et ne lance pas un second `/api/ai/orch/chat/open`, ce qui supprime la fenêtre TOCTOU dans laquelle un nouveau run du même locator pourrait être relu à la place de l'ancien ; - la mémoire Orch du child source reste intacte pendant ce handoff direct ; les ouvertures classiques hors liveSession continuent seules à utiliser `/api/ai/orch/chat/open` ; - un run settled avant la baseline durable ne peut plus devenir replayable ni handoff-ready ; il doit être évincé immédiatement et exactement lors de `markSourceSettled`, faute de quoi il resterait retenu sans TTL et fuirait en mémoire ; - un ancien `turnId` ou une ancienne lease ne doivent jamais retomber sur le nouvel index actif du même locator : open, stream, abort et release utilisent tous le run ref exact et ne possèdent aucun fallback vers le run actif ; - un replay déjà attaché à un ancien record handoff-ready peut toujours drainer ce record exact, mais un nouvel open ou une revalidation de handoff de cet ancien run retourne `not\_found` lorsqu'un autre run est désormais actif sur le même locator ; - une configuration liveSession invalide signifie une configuration liveSession invalide avec un locator Prompt de base toujours canonique ; la stricte validation du locator Prompt existant ne doit pas être affaiblie. ## Objectifs décrits par l'utilisateur ### Besoin initial exprimé par l'utilisateur - Extraire et généraliser le système live actuellement possédé par EasyAgents dans un module `liveSession` organisé dans les trois projets : backend Orch, frontend Prompt et contrats shared. - Conserver un backend orienté objet composé de services spécialisés et d'une façade claire. - Conserver un frontend séparé en hook, adaptateur de présentation et composant unitaire avec CSS local. - Ne pas absorber Queue A, Queue B, locks, sanitizers ou politiques Batch dans le cœur générique. - Rebrancher EasyAgents sans conserver dans son domaine l'implémentation du buffer, de l'ouverture live-safe, du replay/tail ou des leases. - Préserver textes, boutons, test IDs, `Watch live`, hard stop au dernier viewer, handoff terminal, ordre Git/Batch, read-only, TTL 15 s / 30 s et buffer raw-live. - Conserver la séparation stricte `sourceSettled` / `handoffReady`. - Empêcher tout événement tardif d'un ancien tour ou d'une ancienne lease d'affecter un nouveau run du même locator. - Déplacer la documentation générique du watcher vers une programming doc Orch et laisser dans la doc EasyAgents uniquement l'adaptation métier. - Tester tout code créé, déplacé ou modifié, notamment courses, TTL, replay, endpoints, message bus, adaptateurs, Prompt, contrôleur et dispose panneau. - Ne modifier aucun `\*.generated.\*` et aucun fichier personnel du répertoire documentaire interdit. - Réaliser uniquement la tranche 1 : aucune continuation après fermeture d'une session normale, aucune évolution Activité/SidebarStore et aucun nouveau bouton Stop visible. ### Clarifications implicites ajoutées par AICode - Backend : `liveSession/{core,registry,open,replay}` sous Orch. Frontend : `liveSession/{core,adapters,components}` sous Prompt. Shared : `liveSessionTypes`. - Le cœur n'importe jamais EasyAgents. Le pont métier est un adaptateur fin sous `batch/run/liveSession`. - Le parent Batch devient un `viewerGroupId` opaque : `JSON.stringify(['easyAgentsBatch', sessionId, storageScopeRelPath])`. L'adaptateur le décode avec tuple Zod + `ZChatSessionLocator`. La ligne reste une donnée de présentation dans la query string. - Le Batch publie aussi `liveWatchTurnId` ; la paire `liveWatchLineNumber` / `liveWatchTurnId` est atomique, live-only et jamais persistée. - Routes : `/api/ai/orch/liveSession/open`, `/api/ai/orch/liveSession/abort` et `/api/ai/orch/liveSession/stream`. Canal : `proxylocalapi:liveSessionStream`. Variant frontend : `liveSessionObserver`. - La route `abort` est un transport interne du bouton Stop existant ; elle n'ajoute aucune nouvelle action UI. - Query params : `openMode=liveSession`, `liveSessionTurnId`, `liveSessionViewerLeaseId`, `liveSessionClosePolicy=stopOnLastViewerClose`, `easyAgentsBatchLineNumber=<entier positif>`. La policy partagée expose aussi `continueOnPanelClose`, non utilisée par un produit dans cette tranche et rejetée par l'adaptateur EasyAgents. - Ordre canonique de la query EasyAgents : locator, `openMode`, `liveSessionTurnId`, lease, close policy, ligne. - Deux index de run : exact par locator + turn, actif par locator. Un ancien run handoff-ready peut rester retenu sans bloquer un nouveau run. - Une époque monotone `locatorEpochByLocatorKey` complète les deux index. Elle change chaque fois que le pointeur actif d'un locator change ou est retiré et n'est jamais remise à zéro pendant la vie du service ; elle protège les ouvertures asynchrones contre les bascules ABA. - Un index global lease → run exact conserve les leases actives et les corrélations retired jusqu'à éviction. Même après cette éviction, un événement tardif reste isolé parce que son `turnId` est obligatoire et qu'aucune opération ne retombe sur l'index actif. - Seul `registerRun` crée un record. Il est appelé par `onSourceStarted` après le retour réussi de `askStream(...)` et avant le traitement du premier résultat de l'itérateur. - `registerRun` valide tous les conflits avant mutation ; une registration idempotente avec les mêmes métadonnées n'altère ni l'index actif ni l'époque locator. Un échec ne laisse jamais un record partiellement créé. - Le callback source-start est obligatoirement synchrone par son type de retour non-Promise. Le sink EasyAgents mémorise localement le succès de l'enregistrement ; le callback source-settled est toujours appelé une fois par le child executor, mais ne délègue à `markSourceSettled` que si ce même sink a achevé source-start avec succès. - Le child executor arme l'itérateur lazy immédiatement après `askStream(...)` en conservant la première promesse `next()`, appelle ensuite source-start sans `await`, puis consomme d'abord ce résultat et poursuit l'itération. En

cas d'échec source-start, il abort, ferme/détruit l'itérateur, observe la première promesse et attend le cleanup best-effort avant settlement. - Seules les leases actives `opening|attached` d'un record non handoff-ready comptent comme viewers. `opening` → `attached` est monotone. Un open idempotent ne repousse jamais la deadline initiale. - `markHandoffReady` conserve les leases actives, suspend leur TTL opening, fixe une seule fois la rétention terminale et réveille les consumers. - Release active pré-handoff : retourne le groupe libéré. Release répétée, expirée, inconnue, retired ou post-handoff : strict no-op sans groupe et sans fallback vers le run actif. - Une lease retired/expired ne peut être réacquise ni attachée avant éviction de sa corrélation. Une lease active d'un ancien record handoff-ready reste liée à cet ancien record et peut drainer son replay exact. - Locator/turn incohérent après résolution d'une lease connue : erreur explicite ; aucun fallback. - Un open inconnu/stale visant un ancien turn pendant qu'un autre run est actif sur le locator retourne `not\_found` et n'acquiert pas la lease. - L'ouverture acquiert la lease avant le snapshot. Snapshot absent/échoué : lease conservée `opening` tant qu'elle n'a pas été released/expired entre-temps. - L'ouverture utilise un permit opaque capturant run exact, lease, état initial et époque locator. Après le `await` du reader, le permit est revalidé ; une lease released/expired, un record évincé, un changement d'époque ou un nouveau run invalident le résultat au lieu de retourner un faux `live`/`terminal`. - `sourceSettled` est idempotent, ne ferme jamais le replay et interdit tout append ultérieur. - Sans record, `markSourceSettled` est un no-op afin que le `finally` du child executor reste sûr. Le guard local du sink évite en plus de solder un record préexistant lorsqu'un source-start conflictuel a échoué. Avec un record sans baseline, il marque le settlement puis évince immédiatement ce record exact. Avec baseline, il conserve le record pour le handoff exact. - `appendChunk`, `markBaselineReady` et `markHandoffReady` exigent un record exact déjà enregistré ; ils ne créent rien implicitement. - `markHandoffReady` exige baseline + source settled, mais pas terminal meta. - Un terminal live valide est `meta.error` ou `meta.stop\_reason` avec raison différente de `TOOL\_USE`. La règle de stop reason est centralisée dans un helper shared utilisé par le registre, l'ouverture persistée et le frontend. - Le résultat canonique `live` contient le snapshot filtré, `threadId` et `turnId`. Le résultat canonique `terminal` contient le snapshot terminal complet, `threadId` et `turnId`. `not\_ready` et `not\_found` ne contiennent aucun de ces champs. - Le snapshot terminal complet est celui lu et validé par l'appel Open exact ; le hook l'hydrate directement. Aucun second appel `/api/ai/orch/chat/open` n'est autorisé dans le chemin liveSession. - `appendChunk` et `markBaselineReady` retournent la première transition `becameReplayReady`. - Le handler vérifie ownership exact run + lease avant d'ouvrir Activité/son. EasyAgents utilise `observer` ; `source` est testé mais non activé fonctionnellement. - L'autorisation d'abort exige : run exact toujours actif pour le locator, ownership `observer`, source non settled, handoff non prêt, record non évincé et lease active `opening|attached`. Un ancien run, une ancienne lease, une lease retired, un run `source`, settled ou handoff-ready retourne `accepted:false` sans appeler l'abort Orch. - Le parseur frontend EasyAgents retourne `inactive|invalid|ready` ; `ready` transporte run ref + lease + adaptateur déjà lié à la ligne. Le hook générique ne reçoit aucune donnée Batch ni `unknown` à recaster. - Le hook possède bootstrap, stream, terminal latch, revalidation exacte du handoff, hydratation terminale stricte et view model. Pour le bootstrap terminal, le handoff automatique et Retry, l'unique appel Open exact doit retourner `kind=terminal` avec son snapshot complet ; `live/not\_ready/not\_found` devient un échec sûr. - Le meta d'erreur injecté par `useAiStream.onError` arme le latch du tour exact, mais ne transforme jamais à lui seul un snapshot persistant incomplet en résultat `terminal`. - Le Stop existant reste piloté par `!abortAckPending`, mais `useAiStream.stop()` appelle la route liveSession exacte pour `liveSessionObserver` et conserve la route d'abort historique pour `promptQuery`. - Config live invalide : aucun endpoint/stream, dialogue `Live watcher bootstrap failed`, puis fermeture. - `openPanel` parse best-effort le descripteur de release liveSession ; des paramètres liveSession invalides créent quand même le panneau afin que le frontend explique puis ferme. Le locator Prompt de base reste strictement requis et validé comme aujourd'hui. ## Vue d'ensemble de la méthode choisie | Couche | Module | Responsabilité | ---|---|---| Shared | `liveSessionTypes` | Run refs, leases, clés, politiques/query params, résultat d'ouverture, helpers terminaux communs. || Shared Batch | `AiBatchViewState` | Transport live-only atomique ligne + turn vers les surfaces EasyAgents. || Orch | `liveSession/core` | Façade composée et singleton partagé. || Orch | `liveSession/registry` | Records, index exacts, époque locator, permits d'ouverture, readiness, leases/tombstones, TTL, ownership, révision et autorisation d'abort. || Orch | `liveSession/open` | Classification exacte, snapshot live-safe, snapshot terminal atomique et revalidation post-I/O. || Orch | `liveSession/replay` | Subscription exacte, backlog + tail, curseur suffixe, erreur terminal manquant. || EasyAgents | `batch/run/liveSession` | Enregistrement source, readiness Batch, groupe parent, hard-stop policy et éviction destructive. || Endpoints | `liveSession/{open,abort,stream}` | Bootstrap exact, snapshot terminal exact, abort exact du bouton existant et replay exact. || Message bus | `liveSessionStreamHandlers` | Relay et ownership observer/source. || Frontend | `liveSession/core` | Bootstrap → stream → vrai terminal → Open exact retournant le snapshot terminal → handoff strict. || Frontend | `liveSession/adapters` | Query string, validation et présentation EasyAgents. || Frontend | `liveSession/components` | Banner générique autonome. | ### Contrats et records `ZLiveSessionOpenResult` est une union stricte `live|terminal|not\_ready|not\_found`. `live` contient `ZChatSessionFile`, `threadId` et `turnId` non vides ; son snapshot retire les assistants du tour actif. `terminal` contient également `ZChatSessionFile`, `threadId` et `turnId` non vides ; son snapshot est complet et terminal pour le tour exact. `ZLiveSessionOpenResultTransport` reste un `ZodObject` strict avec champs optionnels et `superRefine`, utilisant la

présence propre des propriétés pour rejeter les champs interdits même explicitement `undefined`. Le service et le frontend utilisent l'union canonique ; l'objet existe uniquement pour Slimbk. `AiBatchViewState` ajoute `liveWatchTurnId?: string`. Le schéma impose que `liveWatchLineNumber` et `liveWatchTurnId` soient soit tous deux présents, soit tous deux absents. Le projector n'expose la paire que si la ligne correspondante est encore `running`. Un record contient : run ref exact, groupe opaque, readiness, ownership `observer|source`, chunks ordonnés, baseline, replay-ready, source-settled, `hasTerminalMeta`, handoff-ready, leases actives, corrélations retired, révision, deadlines immuables, rétention terminale, signal et flag `evicted`. `hasTerminalMeta` est activé uniquement via le helper shared : `meta.error` ou `meta.stop\_reason` hors `TOOL\_USE`. Le registre maintient aussi `locatorEpochByLocatorKey`. Cette époque est monotone et change uniquement quand le run actif d'un locator change ou disparaît. Elle ne remplace pas la révision du record : la révision réveille les consumers du record exact, alors que l'époque protège les lectures asynchrones contre une bascule de run, y compris ABA. Le registre n'expose jamais le record mutable. Une subscription opaque est liée au record et à la lease exacts ; elle fournit `bufferedChunks.slice(cursor)`, prochain curseur, révision, handoff, eviction, terminal et état de lease. Elle reste lisible après retrait des index juste assez longtemps pour drainer puis terminer, sans autoriser de nouvelle opération. Un permit d'ouverture opaque capture le locator key, son époque, le run key exact, la lease et la classification initiale ; l'open service doit le revalider après chaque lecture asynchrone. ### Enregistrement source, leases, replay et révision - `askStream(...)` retourne d'abord le `Readable` ; le child executor fixe `didStartChildRun=true`, obtient son itérateur, lance et conserve la première promesse `next()` afin d'armer le runtime Orch lazy, appelle le callback source-start synchrone, puis seulement interprète le premier résultat et poursuit l'itération. - Le priming de l'itérateur n'est pas une livraison métier du premier chunk : aucun `JSON.parse`, append, callback chunk ou changement Batch n'a lieu avant la réussite de source-start. - `onSourceStarted` enregistre le run exact avec readiness `baselineAndObservableChunk`, groupe EasyAgents et ownership `observer`. Son contrat de retour non-Promise rend l'appel synchroniquement vérifiable. - Le sink EasyAgents conserve un booléen local placé à vrai uniquement après le retour réussi de `onSourceStarted`. Le callback `onSourceSettled` reste appelé exactement une fois par le child executor, mais ne délègue au registre que lorsque ce booléen prouve que ce sink a terminé l'enregistrement. - Le `finally` associé à tout stream retourné appelle `onSourceSettled` exactement une fois, même si source-start, la première lecture, l'itération, `JSON.parse`, onChunk, onDurable, onWatchable ou l'itérateur lève. - Si source-start ou un autre traitement post-retour échoue, le child executor tente l'abort, ferme/détruit l'itérateur armé, observe la première promesse `next()` et attend le cleanup best-effort avant settlement. Aucun rejet de la première lecture ne reste non observé. - Si l'erreur principale, la fermeture de l'itérateur et le settlement lèvent, l'erreur principale n'est pas silencieusement masquée ; les erreurs secondaires sont agrégées ou annexées de façon déterministe, dans l'ordre fermeture puis settlement. - Si `askStream(...)` lève avant de retourner, aucun itérateur n'est armé, aucun source-start et aucun settlement ne sont exécutés. - Acquisition seulement sur le run ref exact, actif et replay-ready. - Lease connue résolue avant tout index actif. - `opening` compte immédiatement ; TTL 15 s pré-handoff. `attached` sans TTL opening. - Release active pré-handoff retire seulement la lease ciblée, crée la corrélation retired, réveille et retourne le groupe. - Handoff exclut toutes les leases du comptage mais les conserve replay-authorized. - Éviction : marquer `evicted`, incrémenter/réveiller, puis retirer les index ; l'index actif locator n'est supprimé que s'il pointe encore sur ce run. Toute modification effective du pointeur actif incrémente l'époque du locator. - Rétention terminale 30 s non prolongeable. - Sweeps aux frontières publiques ; horloge injectée en test. - Aucun append/baseline/handoff ne crée un record absent. Un événement tardif visant un run évincé échoue ou no-op selon son contrat, sans résurrection. Chaque consumer : attache la lease, lit son suffixe, vérifie avant chaque nouvelle émission que la lease reste active/replay-authorized, émet sans transformation, puis après drainage échoue si handoff sans vrai terminal, sinon quitte sur handoff/éviction ; `sourceSettled` seul ne ferme jamais. L'attente installe d'abord le listener, relit révision/éviction sur le record exact, retire le listener si déjà avancé et résout. Les appels idempotents n'incrémentent ni révision record ni époque locator. ### Ouverture, revalidation et abort exact - Open reçoit toujours locator + `turnId` + lease. - Une lease connue doit correspondre exactement au run demandé. - Une lease inconnue ne peut être acquise que sur le run ref exact demandé, jamais sur « le run actif du locator » par défaut. - Si le run exact n'existe plus et qu'un autre run est actif sur le locator, Open retourne `not\_found` sans lire/hydrater ce nouveau run et sans enregistrer la lease. - Si aucun autre run n'est actif, un snapshot persistant ne peut être classé `terminal` qu'après sélection du dernier assistant du `turnId` demandé et validation d'un terminal hydratable : erreur persistée (`errorMessage` ou `errorDetails`) ou stop reason hors `TOOL\_USE` ; `TIMEOUT` exige un timeout strictement positif et fini. - La présence d'un user message, d'un assistant sans terminal, d'un usage seul ou d'un `stopReason=TOOL\_USE` ne suffit jamais. Sans record exact, ces états donnent `not\_found`. Avec un record exact handoff-ready mais un snapshot encore non terminal, ils donnent `not\_ready`. - Le résultat `terminal` retourne directement ce snapshot complet, `threadId` et le `turnId` demandé. Il n'existe aucun second read par `/api/ai/orch/chat/open` dans le chemin liveSession. - Après toute lecture async, le service revalide le permit : époque locator inchangée, absence de nouveau run, record exact non redirigé, lease toujours active/replay-authorized lorsque le résultat est live, et aucune éviction destructive intervenue. - Une lease devenue retired/expired pendant le read ne peut pas recevoir un faux résultat live ; l'ouverture échoue explicitement ou se reclasse uniquement vers un terminal exact

déjà validé, jamais vers le run actif courant. - L'époque locator détecte aussi le cas ABA où un nouveau run démarre puis disparaît avant la fin du reader. - Le hook utilise le snapshot terminal retourné par Open pour le bootstrap terminal, le handoff automatique et Retry. L'hydratation reste stricte (`allowOrphanToolResults=false`) et utilise respectivement `classicHydration` ou `terminalHandoff`. - Abort reçoit le même run ref + lease. `canAbortSource` et l'appel `orchAsk.stream.abortStream(...)` restent synchrones et sans `await` entre les deux, afin que le controller thread-only ne puisse pas basculer vers un nouveau run entre l'autorisation exacte et l'abort. - Il ne délègue à `orchAsk.stream.abortStream(sessionId, 'live_session_viewer_stop')` que si l'autorisation exacte du registre est encore vraie ; sinon il retourne `{ accepted:false }`. ### Ordre EasyAgents conservé 1. Création du `turnId`. 2. Retour réussi de `askStream(...)`. 3. Armement de l'itérateur lazy et conservation de la première lecture pendante, sans traitement métier du chunk. 4. `onSourceStarted` enregistre exactement le run avant l'interprétation du premier chunk. 5. Consommation unique par le parser Batch, d'abord du résultat primé puis du tail. 6. Append de chaque chunk au run exact. 7. `fallback_persisted` marque la baseline. 8. Readiness = baseline + token/thinking/raw-live ; transition unique `becameReplayReady`. 9. Publication unique et atomique de `liveWatchLineNumber` + `liveWatchTurnId`. 10. Open : run ref exact + lease unique, permit/époque, puis snapshot filtré et revalidation post-I/O. 11. Stream : attach exact, backlog puis tail. 12. `onSourceSettled` exactement une fois pour tout stream retourné, après fermeture/cleanup de l'itérateur en cas d'erreur. Si le sink n'a pas réussi source-start, il ne solde aucun record. Si le record enregistré n'a pas de baseline, éviction exacte immédiate ; sinon settlement persistant. 13. Convergence Git/métadonnées/parent dans l'ordre actuel. 14. Handoff exact locator + `turnId`. 15. Drainage ; si aucun vrai terminal source, erreur de protocole et injection frontend `meta.error`. 16. Frontend attend vrai terminal du tour exact + `isStreaming=false`, rappelle Open avec run ref + lease, exige `terminal` et hydrate directement le snapshot exact retourné. Un snapshot persistant non terminal laisse le panneau en handoff failed. ### Non-objectifs Aucune modification de `PromptThreadCleanup`, `SidebarStore`, `Activité`, continuation de session normale, bouton Stop supplémentaire, compositeur normal, `continueResponse`, Queue A/B, TTL, compression raw-live ou fichier généré. Les ouvertures Prompt classiques hors `openMode=liveSession` continuent d'utiliser `/api/ai/orch/chat/open` sans changement. La nouvelle route `abort` ne constitue pas un nouveau contrôle utilisateur : elle sécurise le bouton Stop déjà présent. ## Liste exacte des fichiers avec détails d'implémentation ### Shared ### 1) `AIcode/src/sharedlib/src/types/liveSessionTypes/LiveSessionRun.type.ts` — créé - Définir `ZLiveSessionRunRef`/type : locator complet strict + `turnId` trim/non vide. - Définir le schéma/type de lease trim/non vide. - Fournir les clés locator/run par `JSON.stringify` de tuples structurels explicites, sans concaténation ambiguë ni champ EasyAgents. - Le run key est la seule identité métier acceptée par append, baseline, settlement, handoff, open, replay, abort et release ; les permits/subscriptions internes ne servent qu'à prouver la fraîcheur d'une opération déjà liée à ce run. ### 2) `AIcode/src/sharedlib/src/types/liveSessionTypes/LiveSessionOpenResult.type.ts` — créé - Quatre branches strictes. - `live` contient `ZChatSessionFile`, `threadId` et `turnId` non vides ; le snapshot est live-safe et filtré. - `terminal` contient également `ZChatSessionFile`, `threadId` et `turnId` non vides ; le snapshot est complet, terminal et exact pour le run demandé. - `not_ready` et `not_found` ne contiennent aucun champ de session/thread/turn. - Union canonique et objet Slimbk strict corrélé par `superRefine` et contrôle `hasOwnProperty` des champs interdits. - Le transport doit rejeter un champ de branche interdit même si sa valeur est explicitement `undefined`. ### 3) `AIcode/src/sharedlib/src/types/liveSessionTypes/LiveSessionPanel.type.ts` — créé - Constantes pour `openMode`, valeur `liveSession`, paramètre `liveSessionTurnId`, lease et close policy. - Policy fermée `stopOnLastViewerClose` continueOnPanelClose ; seconde non activée dans cette tranche. - Exposer des schémas réutilisables par le frontend et `openPanel` sans dépendance EasyAgents. ### 4) `AIcode/src/sharedlib/src/types/liveSessionTypes/LiveSessionStream.util.ts` — créé - Exporter `isLiveSessionTerminalStopReason(reason)` et `isLiveSessionTerminalMetaChunk(chunk)`. - Le helper stop reason retourne vrai pour toute raison différente de `AiStopReason.TOOL_USE`. - Le helper chunk retourne vrai uniquement pour `meta.error` ou `meta.stop_reason` dont la raison est terminale selon le helper précédent. - Retourner faux pour token, thinking, autres metas et `stop_reason=TOOL_USE`. - Être l'unique définition de la sémantique `TOOL_USE`/non-terminal utilisée par le registre backend, le prédicat de snapshot persisté et le latch frontend. ### 5) `AIcode/src/sharedlib/src/types/liveSessionTypes/__test__/LiveSessionContracts.test.ts` — créé - Couvrir run refs, leases, locator complet, clés sans collision, quatre branches, transport Slimbk, champs interdits `undefined`, constantes/policies et paramètre turn. - Vérifier que `live` et `terminal` exigent session/thread/turn, tandis que `not_ready`/`not_found` les interdisent. - Couvrir les helpers terminaux : error, END_TURN/ABORTED/TIMEOUT/etc. vrais ; `TOOL_USE` et autres chunks faux. ### 6) `[AiBatchViewState.type.ts]`(<https://aicode.local/open-file?path=AIcode%2Fsrc%2Fsharedlib%2Fsrc%2Ftypes%2FAiBatchViewState.type.ts>) — modifié - Ajouter `liveWatchTurnId?: string` live-only, trim/non vide. - Corréler strictement `liveWatchLineNumber` et `liveWatchTurnId` : les deux présents ou les deux absents. - Documenter que cette paire identifie le run exact attendu par le bouton `Watch live` et n'est jamais persistée. ### 7) `AIcode/src/sharedlib/src/types/aiTypes/__test__/AiBatchViewState.liveWatchIdentity.test.ts` — créé - Valider l'absence

conjointe, la présence conjointe et le rejet des deux états partiels. - Valider les contraintes entier positif pour la ligne et string trim/non vide pour le turn. ### Backend Orch générique ### 8)

`AICode/src/extension/src/aiTools/orch/liveSession/liveSession.programming-doc-readme.md` — créé - Respecter le format programming doc ; devenir la source de vérité index/leases/TTL/readiness/ownership/open/replay/abort/handoff. - Documenter settlement vs handoff, source-start registration, priming du Readable Orch lazy, fermeture de l'itérateur sur échec source-start, guard local de settlement, leases post-handoff, snapshot manquant, snapshot terminal direct, absence de second `/chat/open`, époque locator anti-ABA, permits d'ouverture, revalidation lease post-I/O, éviction conditionnelle, subscription exacte, suffixe par curseur, buffer non borné inchangé, vrai terminal excluant TOOL_USE, prédicat terminal persistant, erreur après drainage, éviction immédiate d'un record settled sans baseline, rejet post-settlement, isolation ancien/nouveau turn, abort exact, Queue A/B externes et tranche 2 inactive. ### 9)

`AICode/src/extension/src/aiTools/orch/liveSession/registry/LiveSessionRecord.type.ts` — créé - Types internes record/readiness/ownership, lease active `opening|attached`, corrélation retired séparée. - `hasTerminalMeta` signifie vrai terminal selon helper shared, pas tout `stop\_reason`. - Snapshot atomique suffixe/cursor/révision/handoff/evicted/terminal/lease state et subscription opaque. - Définir un permit d'ouverture opaque et immutable capturant locator key, époque locator, run key exact, lease id et classification initiale ; aucun record mutable ne traverse la frontière du registre. - Inclure l'état nécessaire à l'autorisation d'abort : run actif, ownership, source settled, handoff et lease active. ### 10) [`EasyAgentsBatchLiveObserver.service.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2FEasyAgentsBatchLiveObserver.service.ts) → `AICode/src/extension/src/aiTools/orch/liveSession/registry/LiveSessionRegistry.service.ts` — déplacé/refondu - Index `recordsByRunKey`, `activeRunKeyByLocatorKey`, `runKeyByKnownLeaseId`, plus `locatorEpochByLocatorKey`; TTL 15/30 s ; horloge injectable. - `registerRun` est l'unique créateur de record ; toutes les validations/conflits ont lieu avant mutation. Registration idempotente pour les mêmes métadonnées, conflits explicites sinon. - Une registration idempotente ne réactive jamais un ancien record, ne remplace pas un nouveau run actif et n'incrémente ni révision ni époque. - Toute création, remplacement ou suppression effective du pointeur actif locator incrémente son époque monotone. L'époque reste disponible après retour à « aucun run actif » afin de détecter les ABA. - `appendChunk`, `markBaselineReady` et `markHandoffReady` exigent le record exact ; aucun get-or-create implicite. - Valider `chunk.turnId`, rejeter append settled/handoff/evicted. - Calculer `hasTerminalMeta` exclusivement via `isLiveSessionTerminalMetaChunk`. - Transition `becameReplayReady` unique depuis append/baseline. - Nouveau run autorisé après handoff/éviction de l'ancien actif. - Lease connue : exact d'abord ; active idempotente sans repousser deadline ni régression attached → opening ; retired/expired refusée ; incohérence locator/turn refusée. - Pour une lease inconnue, Open utilise le run ref demandé et exige qu'il soit le run actif/replay-ready ; aucun fallback vers l'index actif. - Exposer au service Open un permit opaque préparé avant I/O et une revalidation après I/O. La revalidation contrôle époque locator, exact run, lease active/replay-authorized, eviction et absence de redirection. - Attach crée une subscription exacte ; suffixe par curseur ; ancien record jamais redirigé. - Release pré-handoff exacte ; autres releases no-op. - Handoff exige baseline + source settled, conserve leases, suspend TTL, n'exige pas terminal meta. - `markSourceSettled` : no-op sans record ; premier settlement réveille et interdit tout append ; si `baselineReady=false`, marquer puis évincer immédiatement le run exact sans toucher un éventuel nouveau run actif ; si baseline vraie, conserver pour handoff. - Autorisation d'abort vraie uniquement pour le run ref encore actif, ownership observer, source non settled, handoff non prêt, record non évincé et lease active opening/attached. - Un ancien handoff record avec nouveau run actif reste replayable par subscription déjà exacte, mais ne peut plus autoriser Open classique ni abort. - Attente sans réveil perdu et retrait des listeners inutiles. - Éviction locator-level de tous les records, avec snapshot final pour subscriptions déjà attachées et mise à jour conditionnelle de l'époque active. ### 11) [`EasyAgentsBatchLiveObserver.service.test.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F__test__%2FEasyAgentsBatchLiveObserver.service.test.ts) → `AICode/src/extension/src/aiTools/orch/liveSession/registry/\_\_test\_\_/LiveSessionRegistry.service.test.ts` — déplacé/étendu - Couvrir ancien comportement durability/readiness/raw-live/TTL/handoff/éviction. - Vérifier que seul register crée ; append/baseline/handoff absents échouent et un événement tardif ne recrée pas un record évincé. - Vérifier l'atomicité de register : un conflit ne laisse aucune mutation partielle ; une registration idempotente d'un ancien run ne réactive pas son locator et ne change pas l'époque. - Leases A/B, release A, B présente ; idempotence sans TTL refresh ; opening expiration ; post-handoff replay authorization sans viewer/TTL ; no-op release ; handoff TTL non repoussé. - Deux runs successifs, ancien événement, ancien TTL n'effaçant pas nouveau run/leases ; ancienne lease et ancien turn ne basculant pas ; retired non réacquérable. - Époque locator : changement à chaque vraie transition active, aucune modification sur idempotence, détection d'un cycle ABA ancien → nouveau → aucun. - Turn mismatch, append post-settlement, handoff prématuré, readiness unique, réveil sans perte/listener cleanup, subscription ancien record, suffixe exact. - `stop\_reason=TOOL\_USE` ne positionne pas `hasTerminalMeta` ; vrai stop/error oui. - Settlement sans baseline évince immédiatement le record exact et n'efface pas un nouveau run actif ; settlement avec baseline conserve le record. - Autorisation d'abort vraie/faux pour opening, attached,

source, settled, handoff, retired, ancien run et nouveau run. ### 12)

`AICode/src/extension/src/aiTools/orch/liveSession/replay/LiveSessionReplay.service.ts` — créé - Async generator générique, validation exacte, attach monotone, curseur local, suffixe sans transformation. - Vérifier la lease avant chaque nouvelle émission du suffixe. - Quitter avant nouvelle émission si lease released/expired ; continuer après source settled ; drainer puis lever si handoff sans vrai terminal ; quitter sur handoff/éviction. ### 13)

`AICode/src/extension/src/aiTools/orch/liveSession/replay/\_\_\_test\_\_\_/LiveSessionReplay.service.test.ts` — créé - Backlog/tail/deux consumers/suffixes, source-settled non terminal, handoff avec terminal, handoff avec seulement TOOL_USE puis erreur après drainage, handoff sans terminal, handoff avant attach, éviction/release pendant attente, leases invalides et course snapshot → attente. - Vérifier qu'une release entre deux éléments d'un même suffixe empêche l'émission suivante. ### 14) [`EasyAgentsBatchLiveOpen.service.ts`](<https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2FEasyAgentsBatchLiveOpen.service.ts>) → `AICode/src/extension/src/aiTools/orch/liveSession/open/LiveSessionOpen.service.ts` — déplacé/généralisé - Recevoir run ref exact + lease ; registre + reader. - Résoudre une lease connue avant toute classification ; acquisition opening uniquement pour `live` et uniquement sur le run exact actif/replay-ready. - Obtenir un permit opaque avant le reader, puis le revalider après chaque `await` de snapshot. - Snapshot live après acquisition ; filtrer seulement les assistants du `turnId` ; aucun reset mémoire. - Snapshot absent : `not\_ready` ; erreur reader : propagation ; dans les deux cas ne pas libérer une lease encore active. - Pour `terminal`, sélectionner le dernier assistant du `turnId` demandé et exiger un terminal hydratable : `errorMessage` ou `errorDetails`, ou `stopReason` terminal selon le helper shared ; `TIMEOUT` exige `stopReasonTimeoutMs` fini et positif. - Un assistant usage-only, sans terminal, ou `stopReason=TOOL\_USE` ne suffit pas. - Le résultat `terminal` contient le snapshot complet exact, `threadId` et `turnId`. Le service ne déclenche aucun `/api/ai/orch/chat/open` et ne reset/rehydrate pas la mémoire Orch. - Sans record exact, ne retourner `terminal` que si aucun autre run n'est actif sur le locator, si l'époque locator est restée inchangée pendant le reader et si le snapshot persistant contient le terminal exact ; sinon `not\_found`. - Avec record exact handoff-ready mais snapshot non terminal, retourner `not\_ready` plutôt qu'un faux terminal. - Après lecture async, revalider : époque inchangée, aucun run concurrent même transitoire, record/lease exacts, lease toujours active/replay-authorized pour `live`, aucune eviction/redirection. - Ancienne lease handoff-ready : replay exact autorisé, mais Open retourne `not\_found` si un nouveau run est actif sur le même locator. ### 15) [`EasyAgentsBatchLiveOpen.service.test.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F___test___%2FEasyAgentsBatchLiveOpen.service.test.ts) →

`AICode/src/extension/src/aiTools/orch/liveSession/open/\_\_\_test\_\_\_/LiveSessionOpen.service.test.ts` — déplacé/étendu - Filtrage exact, aucune mutation mémoire, snapshot absent/erreur avec lease conservée, terminal complet exact, assistant terminal error/stop, turn absent=not_found, handoff/non-prêt/absent. - Rejeter comme non terminal : user-only, assistant sans terminal, usage seul, TOOL_USE seul et TIMEOUT sans timeout valide. - Lease retired/expired/incohérente, ancienne lease sur ancien handoff, ancien open pendant nouveau run=not_found. - Course reader bloqué puis nouveau run actif : ne jamais retourner le snapshot du mauvais run. - Course ABA reader bloqué puis nouveau run démarré et évincé avant résolution : l'époque invalide quand même le permit. - Course lease released/expired ou record évincé pendant le reader : ne jamais retourner un faux `live`. - Vérifier que `terminal` retourne exactement le snapshot lu/validé, sans second reader ni effet mémoire. ### 16) `AICode/src/extension/src/aiTools/orch/liveSession/core/LiveSessionService.ts` — créé - Composer registry/open/replay ; API stricte `registerRun`, `appendChunk`, `markBaselineReady`, `markSourceSettled`, `markHandoffReady`, `open`, `replay`, `releaseViewerLease`, `hasViewerForGroup`, `getActivityOwnership`, `canAbortSource`, `evictLocator`. - Propager `becameReplayReady` ; settlement sans record false ; settlement pré-baseline évince exactement ; aucun import EasyAgents. - Conserver `canAbortSource` et les opérations de registre synchrones ; seul Open/Replay dépend de chemins async. - Getter singleton prod unique avec reader réel, partagé par adaptateur/endpoints/handler/policy d'abort. ### 17)

`AICode/src/extension/src/aiTools/orch/liveSession/core/\_\_\_test\_\_\_/LiveSessionService.test.ts` — créé - Parcours complet, registration explicite, readiness unique, source settled ne ferme pas replay, ownership/release, erreurs, deux runs, aucun record vide, settlement pré-baseline évincé, handoff sans vrai terminal accepté puis erreur replay. - Vérifier Open exact sans fallback vers le nouveau run, snapshot terminal direct, époque anti-ABA et `canAbortSource` exact. ### Adaptation EasyAgents backend ### 18) `AICode/src/extension/src/aiTools/easyAgents/batch/run/liveSession/EasyAgentsBatchLiveSessionAdapter.service.ts` — créé - Seul pont Batch ; groupe JSON namespacé réversible, scope racine inclus. - `onSourceStarted` synchrone enregistre readiness `baselineAndObservableChunk`, ownership observer et groupe parent, puis retourne un acquittement non-Promise seulement après réussite complète. - Mapper append/baseline/settled/handoff exacts, présence groupe, release lease, autorisation d'abort via cœur, éviction locator. - Propager `becameReplayReady` ; absence de record au settlement no-op ; parent retourné seulement sur vraie release pré-handoff. - Ne conserver aucun état global de source-start dans l'adaptateur singleton : la preuve que le sink courant a réussi l'enregistrement reste une closure locale construite par l'exécution Batch. - Aucun état de buffer, lease, replay ou TTL ne

reste dans EasyAgents. ### 19) `AICode/src/extension/src/aiTools/easyAgents/batch/run/liveSession/__test__/
EasyAgentsBatchLiveSessionAdapter.service.test.ts` — créé - Groupe déterministe/réversible, readiness/ownership,
source-start registration synchrone, handoff/éviction, leases A/B, ancien handoff+nouveau run, settlement sans record,
settlement pré-baseline sans fuite. - Vérifier qu'un source-start conflictuel ne permet pas au settlement du sink en échec de
soldier le record préexistant. - Vérifier que l'autorisation d'abort et la release restent exactes et que le groupe n'est retourné
que pour une vraie release active pré-handoff. ### 20)

[`EasyAgentsBatchRunManager.service.ts`](https://aicode.local/open-file?
path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2FEasyAgentsBatchRunManag
er.service.ts) — modifié - Obtenir le singleton liveSession générique dans `createProd`, construire un adaptateur
EasyAgents unique puis l'injecter dans execution/retry/presence/last-viewer. - Retirer observer/open et méthodes
`openLiveChild`/`streamLiveChild`. - Release publique accepte run ref exact + lease ; reste de façade inchangé. ### 21)

[`EasyAgentsBatchRunExecution.service.ts`](https://aicode.local/open-file?
path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2FEasyAgentsBatchRunExecut
ion.service.ts) — modifié - Dépendre de l'adaptateur ; sink : source started, append, baseline, replay-ready, source settled. -
Construire un sink par exécution avec une closure `sourceRegistrationSucceeded`, initialement fausse, mise à vrai
uniquement après retour réussi du source-start synchrone ; source-settled est toujours appelé mais ne délègue au registre
que si cette closure est vraie. - À la première transition replay-ready, fixer atomiquement `entry.liveWatchLineNumber` et
`entry.liveWatchTurnId`, puis publier une seule fois. - Effacer/projeter les deux champs ensemble. - Propager `liveTurnId`
exact à tous les handoffs ; helper privé uniquement après narrowing `didStartChildRun:true`. - Préserver tous les
emplacements/ordres Git, commit landed, recovery, retry, pause, stop, done, inter-line. - Les résultats démarrés non
durables n'appellent pas handoff ; leur settlement pré-baseline est déjà évincé par le cœur. ### 22)

[`EasyAgentsBatchRunChildExecution.service.ts`](https://aicode.local/open-file?
path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2FEasyAgentsBatchRunChildE
xecution.service.ts) — modifié - Résultat démarré contient toujours `liveTurnId` ; non démarré ne l'expose jamais. - Après
retour de `askStream`, fixer `didStartChildRun=true`, obtenir l'itérateur async, lancer/conserver la première promesse
`next()`, appeler `onSourceStarted` exactement une fois sans `await`, puis traiter le résultat primé avant le tail. - Aucun
chunk n'est parsé, appendu ou livré au sink avant la réussite de source-start. - `onSourceSettled` remplace
`onStreamTerminal` et est appelé exactement une fois pour tout stream retourné. - Après retour de `askStream`, exécuter
source-start + consommation dans une structure garantissant settlement exactement une fois ; aucun source-start/settlement
si `askStream` lève avant retour. - Garantir settlement sur première lecture, itération, JSON, onChunk, onDurable,
onWatchable et source-start throw. - En cas d'échec post-retour, tenter l'abort child, fermer/détruire l'itérateur/Readable,
observer la première promesse pendante et attendre le cleanup best-effort avant settlement. - Ne pas masquer une erreur
principale par une erreur secondaire de fermeture ou settlement ; préserver les informations dans un ordre déterministe. -
Parsing/classification/barrière/cleanup mémoire inchangés. ### 23)

[`EasyAgentsBatchRunManager.types.ts`](https://aicode.local/open-file?
path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2FEasyAgentsBatchRunManag
er.types.ts) — modifié - `BatchLineExecutionResult` devient une union discriminée : démarré + `liveTurnId` obligatoire ;
non démarré + durable false + `liveTurnId?:never`. - Sink : ajouter `onSourceStarted`, remplacer terminal par
`onSourceSettled`, conserver les callbacks append/baseline/replay-ready. - Le type de retour de `onSourceStarted` doit être
non-Promise afin d'interdire une implémentation async qui laisserait le premier chunk dépasser l'enregistrement ;
`onSourceSettled` expose également un retour synchrone/no-op testable. - Ajouter `liveWatchTurnId` au `BatchRunEntry`,
strictement corrélé à la ligne. - Ajouter la dépendance liveSession/adaptateur nécessaire au wiring testable du manager sans
singleton caché dans les services unitaires. ### 24) [`EasyAgentsBatchRunStateProjector.service.ts`](https://aicode.local/
open-file?
path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2FEasyAgentsBatchRunStateP
rojector.service.ts) — modifié - Projeter `liveWatchLineNumber` et `liveWatchTurnId` ensemble uniquement si la ligne est
encore `running`. - Rejeter un état interne partiel ligne sans turn ou turn sans ligne au lieu de publier un contrat ambigu. -
Ne jamais recopier ces champs depuis un cold state vers un nouveau runtime. ### 25)

[`EasyAgentsBatchRunRetry.service.ts`](https://aicode.local/open-file?
path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2FEasyAgentsBatchRunRetry.s
ervice.ts) — modifié - Remplacer release+evict par éviction destructive locator via adaptateur. - Préserver l'ordre panneau
→ live eviction → Activité/query window → durable delete → cold rebuild. ### 26)

[`EasyAgentsBatchViewerPresence.service.ts`](https://aicode.local/open-file?
path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2FEasyAgentsBatchViewerPre
sence.service.ts) — modifié - Dépendance adaptateur ; Batch UI prioritaire puis live ; sweep intégré au comptage. ### 27)

[`EasyAgentsBatchLastViewerStop.service.ts`](https://aicode.local/open-file?

path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2FeasyAgentsBatchLastViewerStop.service.ts) — modifié - Dispose reçoit run ref exact + lease. - Release exacte ; continuer uniquement si le groupe parent a réellement été libéré ; recheck global puis kill ; post-handoff no-op. ### 28)

[`EasyAgentsBatchLastViewerStop.service.test.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F__test__%2FeasyAgentsBatchLastViewerStop.service.test.ts) — modifié - Harness run ref + lease/parent optionnel ; A/B release tardive ; ancien handoff+nouveau run ; cas UI/no viewer/bootstrap/post-handoff. ### 29)

[`EasyAgentsBatchRunChildExecution.service.test.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F__test__%2FeasyAgentsBatchRunChildExecution.service.test.ts) — modifié - `liveTurnId` présent/absent selon union. - Source-start avant le traitement du premier chunk ; settlement exactement une fois normal, première lecture throw, stream throw, source-start throw, callbacks throw et settlement throw ; zéro callback si askStream throw avant retour. - Utiliser un Readable/async iterable lazy instrumenté avec un `finally` testable : lorsque source-start échoue, vérifier que le premier `next()` a armé le générateur, que la lecture pendante est observée, que l'itérateur est fermé et que le cleanup du générateur s'exécute. - Vérifier qu'aucun chunk primé n'est livré lorsque source-start échoue. - Vérifier la priorité/agrégation des erreurs de lecture/fermeture/settlement et l'abort best-effort après échec post-retour. - Assertions existantes conservées. ### 30)

[`EasyAgentsBatchRunExecution.activeExecution.service.test.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F__test__%2FeasyAgentsBatchRunExecution.activeExecution.service.test.ts) — modifié - Fixtures démarrées avec turn, non démarrées sans ; assertions persistance inchangées. ### 31)

[`EasyAgentsBatchRunExecution.kill.service.test.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F__test__%2FeasyAgentsBatchRunExecution.kill.service.test.ts) — modifié - Fixtures démarrées avec turn ; au moins un handoff exact locator+turn ; invariants kill/await/finalize conservés. ### 32)

[`EasyAgentsBatchRunExecution.testHarness.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F__test__%2FeasyAgentsBatchRunExecution.testHarness.ts) — modifié - Remplacer le mock observer par un mock adaptateur liveSession correctement nommé. - Ajouter source-start synchrone, settlement, guard d'enregistrement local, union fixtures strictes, `liveWatchTurnId`, signatures Jest à un paramètre fonction ; scénarios partagés inchangés. ### 33)

[`EasyAgentsBatchRunExecution.live-activity-and-gating.service.test.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F__test__%2FeasyAgentsBatchRunExecution.live-activity-and-gating.service.test.ts) — modifié - Vérifier que ligne + turn sont publiés ensemble pendant `running` et disparaissent ensemble dès que la ligne n'est plus running. - Conserver les tests de gate inter-ligne et d'activité. ### 34)

[`EasyAgentsBatchRunExecution.commit-and-recovery-failures.service.test.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F__test__%2FeasyAgentsBatchRunExecution.commit-and-recovery-failures.service.test.ts) — modifié - Adapter le mock renommé et vérifier que les handoffs des branches commit/recovery transportent le run ref exact sans changer l'ordre de convergence existant. ### 35)

[`EasyAgentsBatchRunExecution.post-commit-persistence.service.test.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F__test__%2FeasyAgentsBatchRunExecution.post-commit-persistence.service.test.ts) — modifié - Adapter le mock renommé et vérifier les handoffs exacts des branches commit déjà landed / persistance failure. ### 36)

[`EasyAgentsBatchRunLifecycle.service.test.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F__test__%2FeasyAgentsBatchRunLifecycle.service.test.ts) — modifié - Mock adaptateur et settlement ; assertions lifecycle non live inchangées. ### 37)

[`EasyAgentsBatchRunRetry.testHarness.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F__test__%2FeasyAgentsBatchRunRetry.testHarness.ts) — modifié - Mock éviction destructive via adaptateur ; signatures Jest conformes. ### 38)

[`EasyAgentsBatchRunRetry.live-restart.service.test.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F__test__%2FeasyAgentsBatchRunRetry.live-restart.service.test.ts) — modifié - Ordre éviction ; aucun buffer/lease/corrélation survivant avant restart. ### 39)

[`EasyAgentsBatchViewerPresence.service.test.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F__test__%2FeasyAgentsBatchViewerPresence.service.test.ts) — modifié - Adaptateur, aucun sweep explicite externe, cas UI/live/aucun/lease expirée. ### 40)

[`easyAgents.programming-doc-readme.md`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2FeasyAgents.programming-doc-readme.md) — modifié - Remplacer le détail générique par adaptation/policy ; endpoints Orch open/abort/stream ; conserver readiness,

last viewer, Git/handoff, retry destructif. - Documenter la paire ligne/turn live-only et pointer protocole replay, priming du source Readable, époque locator, snapshot terminal direct, vrai terminal, exact abort et settlement pré-baseline vers la nouvelle doc. ### Endpoints et message bus ### 41) [`openLiveChild.endpoint.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2Fslimbk-local%2Fendpoints%2Fai%2FeasyAgents%2Fbatch%2Frun%2FopenLiveChild.endpoint.ts) → `AICode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/open.endpoint.ts` — déplacé/généralisé - Args plats stricts locator + turn + lease shared, sans regex EasyAgents 32-hex locale. - Singleton générique ; transport object ; parse union canonique avant retour. - `'live'` et `'terminal'` transportent tous deux session/threadId/turnId conformément au contrat ; les autres branches interdisent ces champs. - Ne jamais déclencher ``api/ai/orch/chat/open`` ni réinitialiser la mémoire dans cet endpoint. ### 42) [`openLiveChild.endpoint.test.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2Fslimbk-local%2Fendpoints%2Fai%2FeasyAgents%2Fbatch%2Frun%2F__test__%2FopenLiveChild.endpoint.test.ts) → `AICode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/__test__/open.endpoint.test.ts` — déplacé/adapté - Quatre branches, turn, lease, invalides/extras, singleton, transport y compris interdit `'undefined'`, chemins mocks relatifs au fichier de test. - Vérifier que les branches live et terminal transportent le snapshot exact et qu'aucun endpoint chat/open n'est impliqué. ### 43) `AICode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/abort.endpoint.ts` — créé - Args plats stricts locator + turn + lease ; response object strict `{ accepted: boolean }`. - Appeler ``canAbortSource`` sur le singleton ; si faux, retourner `{ accepted: false }` sans toucher Orch. - Si vrai, appeler synchroniquement ``orchAsk.stream.abortStream(sessionId, 'live_session_viewer_stop')`` dans la même pile, sans ``await`` entre check et abort, puis retourner `{ accepted: true }`. - Ne jamais fallback vers l'abort thread-only sans autorisation exacte. ### 44) `AICode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/__test__/abort.endpoint.test.ts` — créé - Autorisation opening/attached, refus stale/ancien run/retired/handoff/source/settled, invalides/extras. - Vérifier l'ordre synchrone autorisation → abort et que l'abort Orch reçoit le thread exact uniquement lorsque ``accepted=true`` ; mocks relatifs au test. ### 45) [`streamLiveChild.endpoint.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2Fslimbk-local%2Fendpoints%2Fai%2FeasyAgents%2Fbatch%2Frun%2FstreamLiveChild.endpoint.ts) → `AICode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/stream.endpoint.ts` — déplacé/généralisé - Exiger locator, turn, lease ; replay générique ; JSON chunks inchangés ; erreur protocole remonte. ### 46) [`streamLiveChild.endpoint.test.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2Fslimbk-local%2Fendpoints%2Fai%2FeasyAgents%2Fbatch%2Frun%2F__test__%2FstreamLiveChild.endpoint.test.ts) → `AICode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/__test__/stream.endpoint.test.ts` — déplacé/adapté - Ordre/multiples/erreurs/invalides ; lease+turn exacts ; erreur après émission partielle ; mocks relatifs. ### 47) [`easyAgentsBatchLiveStreamHandlers.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FmessageBus%2Fhandlers%2Fia%2FeasyAgentsBatchLiveStreamHandlers.ts) → `AICode/src/extension/src/messageBus/handlers/ia/liveSessionStreamHandlers.ts` — déplacé/généralisé - Canal/route génériques ; ownership exact run+lease avant ressources. - Observer : bridges/son actuels. Source : relay seul. - Propager erreurs ; cleanup symétrique avec flags ; ne pas appeler stop fictif ; tenter séparément son et Sidebar ; préserver erreur principale. ### 48) [`easyAgentsBatchLiveStreamHandlers.test.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FmessageBus%2Fhandlers%2Fia%2F__test__%2FeasyAgentsBatchLiveStreamHandlers.test.ts) → `AICode/src/extension/src/messageBus/handlers/ia/__test__/liveSessionStreamHandlers.test.ts` — déplacé/étendu - Canal/route/turn/lease/relay/erreur, observer/source, partiel, ouvertures partielles, deux cleanups tentés, mocks relatifs. ### 49) [`registerAllBusHandlers.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FmessageBus%2Fhandlers%2FregisterAllBusHandlers.ts) — modifié - Enregistrement générique sans changer l'ordre voisin. ### 50) [`MessageRegistry.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fsharedlib%2Fsrc%2Ftypes%2FbusTypes%2Fglobal%2FMessageRegistry.ts) — modifié - Canal générique avec locator scalars, ``turnId`` et lease shared ; ``aiStream`` inchangé. ### Cycle de vie panneau ### 51) [`openPanel.util.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2Fextension%2Fpanels%2Fcore%2FopenPanel.util.ts) — modifié - Safe-parse des paramètres liveSession run ref/lease/policy ; descripteur release optionnel capturé à la création. - La validation du locator Prompt servant à la clé de panneau reste stricte et inchangée. - Une configuration liveSession invalide n'empêche pas la création/reveal du panneau si le locator Prompt est valide. - Pour EasyAgents stop-on-close valide, release exacte run ref + lease via manager ; ``continueOnPanelClose`` n'appelle aucune release dans cette tranche. - Policy avant cleanup Prompt. - Erreurs isolées : release échouée journalisée, cleanup Prompt toujours tenté, cleanup son toujours tenté ; registry/removeQueryWindow/sidebar/trivial cleanup non déplacés. ### 52) [`openPanel.util.test.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2Fextension%2Fpanels%2Fcore%2F__test__%2FopenPanel.util.test.ts) —

modifié - Locator+turn+lease exacts, ordre, erreurs release/Prompt, normal/continue/live invalide, panneau live invalide créé avec locator valide, locator Prompt invalide toujours rejeté, tests existants conservés. ### Frontend liveSession ### 53) [`PromptRuntimeMode.type.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2FPrompt%2FPrompt%2FPromptRuntimeMode.type.ts) → `AICode/src/frontend/src/app/prompt/liveSession/core/PromptLiveSession.types.ts` — déplacé/généralisé - Modes normal/opening/streaming/handoff pending/failed ; bootstrap inactive/invalid/ready. - `ready` = run ref exact + lease + adaptateur lié. - View model générique et port présentation/erreurs ; aucun texte/line/type Batch dans le cœur générique. ### 54) [`usePromptBatchLiveViewer.hook.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2FPrompt%2FPrompt%2FusePromptBatchLiveViewer.hook.ts) → `AICode/src/frontend/src/app/prompt/liveSession/core/usePromptLiveSession.hook.ts` — déplacé/généralisé - Endpoint générique + union shared + run ref + lease ; owns mode/failure/pending turn/terminal latch/single-flight. - Invalid : zéro réseau, callback dialogue/close. - Snapshot live orphan-tolerant ; stream observer locator+turn+lease. - Latch via `isLiveSessionTerminalMetaChunk` : TOOL_USE ne l'arme jamais. - Handoff seulement vrai terminal exact + streaming false. - Pour bootstrap terminal, handoff automatique et Retry, appeler Open avec le même run ref + lease et exiger `terminal` avec session/threadId/turnId exacts ; `live/not\_ready/not\_found` devient un échec sûr. - Hydrater directement `response.session` avec `allowOrphanToolResults=false`. Supprimer tout appel et tout type importé de `api/ai/orch/chat/open` dans ce hook. - Fallback terminal bootstrap `classicHydration`, handoff/retry `terminalHandoff`, tous stricts. - Un `meta.error` injecté par le frontend permet la tentative mais ne contourne jamais la validation du snapshot terminal persistant. - View model projeté ; callbacks/objets React via `useCallback`/`useMemo`. ### 55) [`Prompt.batchLiveViewer.hydratation.test.tsx`](https://aicode.local/open-file?path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2FPrompt%2FPrompt%2F__test__%2FPrompt.batchLiveViewer.hydratation.test.tsx) → `AICode/src/frontend/src/app/prompt/liveSession/core/\_\_test\_\_/usePromptLiveSession.hydratation.test.tsx` — déplacé/adapté - Route/lease/run ref/union/snapshot/orphans/turn/canal ; branches ; terminal direct strict ; invalid sans réseau ; ancien turn + nouveau actif refusé ; mocks relatifs/act/pas d'import expect. - Vérifier que la branche terminal hydrate le snapshot retourné par Open et n'appelle jamais `api/ai/orch/chat/open`. ### 56) [`Prompt.batchLiveViewer.handoff.test.tsx`](https://aicode.local/open-file?path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2FPrompt%2FPrompt%2F__test__%2FPrompt.batchLiveViewer.handoff.test.tsx) → `AICode/src/frontend/src/app/prompt/liveSession/core/\_\_test\_\_/usePromptLiveSession.handoff.test.tsx` — déplacé/adapté - Vrai terminal + fin, revalidation Open terminal retournant le snapshot exact, pending/reload/failure/retry single-flight ; terminalHandoff ; stream error meta ; source settled absent du frontend. - TOOL_USE seul ne déclenche pas le handoff et conduit, à la fermeture en erreur protocole, au meta.error injecté. - Si le snapshot persistant reste non terminal malgré le meta.error frontend, Open ne retourne pas terminal et le panneau reste en handoff failed. - Revalidation qui voit un autre run/non-terminal garde le panneau en handoff failed et ne lance aucune ouverture classique. - Vérifier l'absence totale d'appel `api/ai/orch/chat/open` dans handoff et Retry. - Présentation/enabled adaptateur ; mocks relatifs. ### 57) [`usePromptInputViewerMode.hook.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2FPrompt%2FPromptInput%2FusePromptInputViewerMode.hook.ts) → `AICode/src/frontend/src/app/prompt/liveSession/adapters/EasyAgentsBatchLiveSession.adapter.ts` — déplacé/étendu - Constante line ; builder recevant run ref + line et générant une nouvelle lease via getter `uidGenerator.newUniqueId` à chaque clic. - Ordre query : locator/openMode/turn/lease/policy/line. - Parse strict inactive/invalid/ready ; `continueOnPanelClose` est invalid pour EasyAgents dans cette tranche ; ready crée adaptateur lié. - Textes exacts à préserver : `Opening live Batch chat...`, `Watching live Batch line <N>...`, `Watching live Batch line...`, `Reloading terminal chat snapshot...`, `Live watcher handoff failed.`, `Stop line`, `Reload chat`. - Conserver les dialogues actuels `Live watcher unavailable` et `Live watcher bootstrap failed`, la fermeture identique et aucun lifecycle stream dans l'adaptateur. ### 58) `AICode/src/frontend/src/app/prompt/liveSession/adapters/\_\_test\_\_/EasyAgentsBatchLiveSession.adapter.test.ts` — créé - Query ordre/turn/leases distinctes/policy/line/scopes ; mock relatif uid ; invalides ; `continueOnPanelClose` rejeté ; adaptateur lié ; view models/textes/variants/enabled/dialogues. ### 59) `AICode/src/frontend/src/app/prompt/liveSession/components/PromptLiveSessionBanner.tsx` — créé - View model générique + callbacks ; aucun texte Batch ; test IDs/variants/enabled identiques ; handlers useCallback ; CSS propre uniquement. ### 60) `AICode/src/frontend/src/app/prompt/liveSession/components/PromptLiveSessionBanner.module.css` — créé - Déplacer `viewerBanner` sans changement visuel ; variables VS Code, rem/em ; hauteur locale équivalente `calc(2rem \* 1.5)`. ### 61) `AICode/src/frontend/src/app/prompt/liveSession/components/\_\_test\_\_/PromptLiveSessionBanner.test.tsx` — créé - Statut/détails/test IDs, Stop/Reload, enabled/callback unique, pas de sélecteurs par labels fragiles, act. ### 62) [`AiStreamStartArgs.type.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Ffrontend%2Fsrc%2FmessageBus%2FAiStreamStartArgs.type.ts) — modifié - Remplacer `batchLiveObserver` par `liveSessionObserver` ; ajouter lease ; conserver run turn et `promptQuery` inchangé. ### 63)

[`useAiStream.hook.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Ffrontend%2Fsrc%2FmessageBus%2FuseAiStream.hook.ts) — modifié - Canal générique + locator/turn/lease ; Queue A/B/raw-live/sanitizers/error/drainage inchangés ; raw logs forcés pour liveSessionObserver. - `onError` injecte meta.error avant streaming false comme aujourd'hui. - `stop()` branche sur `activeStream.kind` : route historique pour `promptQuery`, nouvelle route exact run+lease pour `liveSessionObserver`. - Un `accepted:false` live reste un no-op sûr et ne retombe jamais sur l'abort thread-only. ### 64)

[`useAiStream.hook.easyAgentsBatchLiveObserver.test.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Ffrontend%2Fsrc%2FmessageBus%2F__test__%2FuseAiStream.hook.easyAgentsBatchLiveObserver.test.ts) → `AICode/src/frontend/src/messageBus/\_\_test\_\_/useAiStream.hook.liveSessionObserver.test.ts` — déplacé/adapté - Canal/lease/locator/turn, absence stream normal, reset/drain/raw-live, erreur → meta.error terminal. - Stop live appelle uniquement la nouvelle route exacte ; Stop prompt continue d'utiliser la route historique. ### 65)

[`usePromptRunController.hook.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Fprompt%2FPrompt%2FusePromptRunController.hook.ts) — modifié - Nom public `startLiveObserver`; args run ref + lease ; placeholders puis `liveSessionObserver`; Send/Retry/Continue/Edit inchangés. ### 66)

`AICode/src/frontend/src/app/prompt/Prompt/\_\_test\_\_/usePromptRunController.liveSessionObserver.test.tsx` — créé - Vrai controller, placeholders exacts, start exact locator/turn/lease, aucun promptQuery, act/mocks relatifs/pas d'import expect. ### 67)

[`Prompt.tsx`](https://aicode.local/open-file?path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Fprompt%2FPrompt%2FPrompt.tsx) — modifié - Figer openMode/turn/lease/policy/line au premier render ; parser EasyAgents → sélection ; memo paramètres hook. - Déléguer machine/view model ; passer `!abortAckPending`; hydratation classique désactivée pour tout `openMode=liveSession` même invalide ; ownsThread false hors normal ; aucun state Batch parallèle. - Le passage à `normal` après résultat terminal utilise le snapshot exact déjà retourné par Open ; aucun second bootstrap chat/open n'est lancé. - Le locator Prompt canonique continue d'être lu par le hook existant et reste une précondition distincte de la config liveSession. ### 68)

`AICode/src/frontend/src/app/prompt/Prompt/\_\_test\_\_/Prompt.liveSessionBootstrap.test.tsx` — créé - Query valide figée/adaptateur/hydratation off ; mutation query ignorée ; invalid/hydratation off/pas de start ; normal inchangé ; résultats hook transmis sans state parallèle. - Couvrir turn manquant/malformé et close policy EasyAgents non supportée. - Vérifier que le chemin liveSession ne déclenche pas le hook d'hydratation classique ni `/api/ai/orch/chat/open`. ### 69)

[`PromptInput.tsx`](https://aicode.local/open-file?path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Fprompt%2FPromptInput%2FPromptInput.tsx) — modifié - Props view model générique + retry ; banner extrait ; composer/attachments cachés en viewer ; contrôles partagés. - Stop garde le chemin existant vers le controller, Reload garde retry ; aucun switch/texte Batch dans PromptInput. ### 70)

[`PromptInput.module.css`](https://aicode.local/open-file?path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Fprompt%2FPromptInput%2FPromptInput.module.css) — modifié - Retirer seulement `viewerBanner\*`. ### 71)

[`PromptInput.batchLiveViewerMode.test.tsx`](https://aicode.local/open-file?path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Fprompt%2FPromptInput%2F__test__%2FPromptInput.batchLiveViewerMode.test.tsx) — modifié - Harness view model, composer/controls/Stop/Reload/normal, enabled du VM, clics act. ### 72)

[`useEasyAgentsBatchController.hook.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Feasy-agents%2Feasy-agents-batch%2Fcore%2FuseEasyAgentsBatchController.hook.ts) — modifié - Lire la paire line/turn depuis l'état ; refuser un row live incomplet. - Utiliser le builder complet ; nouvelle lease par clic ; retirer anciennes constantes runtime. ### 73)

[`EasyAgentsBatch.liveWatch.test.tsx`](https://aicode.local/open-file?path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Feasy-agents%2Feasy-agents-batch%2Fcore%2F__test__%2FEasyAgentsBatch.liveWatch.test.tsx) — modifié - CTA, locator, turn, query complète/ordre/lease/policy/line ; inspecter query envoyée. ### 74)

[`useEasyAgentBatchRunController.hook.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Feasy-agents%2Feasy-agents-item%2Fcore%2Fbatch%2FuseEasyAgentBatchRunController.hook.ts) — modifié - Même builder, run turn exact, lease unique, ligne, gardes inchangées. ### 75)

[`EasyAgentBatchSurface.test.tsx`](https://aicode.local/open-file?path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Feasy-agents%2Feasy-agents-item%2Fcore%2Fbatch%2F__test__%2FEasyAgentBatchSurface.test.tsx) — modifié - Ajouter le cas item query complète réelle avec turn exact ; autres scénarios non reformatés. ## Récapitulatif exact des chemins de fichiers ### Créés -

- `AICode/src/sharedlib/src/types/liveSessionTypes/LiveSessionRun.type.ts` -
- `AICode/src/sharedlib/src/types/liveSessionTypes/LiveSessionOpenResult.type.ts` -
- `AICode/src/sharedlib/src/types/liveSessionTypes/LiveSessionPanel.type.ts` -
- `AICode/src/sharedlib/src/types/liveSessionTypes/LiveSessionStream.util.ts` -

`AICode/src/sharedlib/src/types/liveSessionTypes/\_\_test\_\_/LiveSessionContracts.test.ts` -
`AICode/src/sharedlib/src/types/aiTypes/\_\_test\_\_/AiBatchViewState.liveWatchIdentity.test.ts` -
`AICode/src/extension/src/aiTools/orch/liveSession/liveSession.programming-doc-readme.md` -
`AICode/src/extension/src/aiTools/orch/liveSession/registry/LiveSessionRecord.type.ts` -
`AICode/src/extension/src/aiTools/orch/liveSession/replay/LiveSessionReplay.service.ts` -
`AICode/src/extension/src/aiTools/orch/liveSession/replay/\_\_test\_\_/LiveSessionReplay.service.test.ts` -
`AICode/src/extension/src/aiTools/orch/liveSession/core/LiveSessionService.ts` -
`AICode/src/extension/src/aiTools/orch/liveSession/core/\_\_test\_\_/LiveSessionService.test.ts` -
`AICode/src/extension/src/aiTools/easyAgents/batch/run/liveSession/EasyAgentsBatchLiveSessionAdapter.service.ts` -
`AICode/src/extension/src/aiTools/easyAgents/batch/run/liveSession/\_\_test\_\_/EasyAgentsBatchLiveSessionAdapter.service.test.ts` -
`AICode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/abort.endpoint.ts` -
`AICode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/\_\_test\_\_/abort.endpoint.test.ts` -
`AICode/src/frontend/src/app/prompt/liveSession/adapters/\_\_test\_\_/EasyAgentsBatchLiveSession.adapter.test.ts` -
`AICode/src/frontend/src/app/prompt/liveSession/components/PromptLiveSessionBanner.tsx` -
`AICode/src/frontend/src/app/prompt/liveSession/components/PromptLiveSessionBanner.module.css` -
`AICode/src/frontend/src/app/prompt/liveSession/components/\_\_test\_\_/PromptLiveSessionBanner.test.tsx` -
`AICode/src/frontend/src/app/prompt/Prompt/\_\_test\_\_/usePromptRunController.liveSessionObserver.test.tsx` -
`AICode/src/frontend/src/app/prompt/Prompt/\_\_test\_\_/Prompt.liveSessionBootstrap.test.tsx` ### Modifiés -
[`AICode/src/sharedlib/src/types/aiTypes/AiBatchViewState.type.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fsharedlib%2Fsrc%2Ftypes%2FAiTypes%2FAiBatchViewState.type.ts) -
[`AICode/src/extension/src/aiTools/easyAgents/easyAgents.programming-doc-readme.md`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FAiTools%2FeasyAgents%2FeasyAgents.programming-doc-readme.md) -
[`AICode/src/extension/src/aiTools/easyAgents/batch/run/EasyAgentsBatchRunManager.service.ts`](https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FAiTools%2FeasyAgents%2Fbatch%2Frun%2FEasyAgentsBatchRunManager.service.ts) - [`AICode/src/extension/src/aiTools/easyAgents/batch/run/EasyAgentsBatchRunExecution.service.ts`] (https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FAiTools%2FeasyAgents%2Fbatch%2Frun%2FEasyAgentsBatchRunExecution.service.ts) - [`AICode/src/extension/src/aiTools/easyAgents/batch/run/EasyAgentsBatchRunChildExecution.service.ts`] (https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FAiTools%2FeasyAgents%2Fbatch%2Frun%2FEasyAgentsBatchRunChildExecution.service.ts) - [`AICode/src/extension/src/aiTools/easyAgents/batch/run/EasyAgentsBatchRunManager.types.ts`] (https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FAiTools%2FeasyAgents%2Fbatch%2Frun%2FEasyAgentsBatchRunManager.types.ts) - [`AICode/src/extension/src/aiTools/easyAgents/batch/run/EasyAgentsBatchRunStateProjector.service.ts`] (https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FAiTools%2FeasyAgents%2Fbatch%2Frun%2FEasyAgentsBatchRunStateProjector.service.ts) - [`AICode/src/extension/src/aiTools/easyAgents/batch/run/EasyAgentsBatchRunRetry.service.ts`] (https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FAiTools%2FeasyAgents%2Fbatch%2Frun%2FEasyAgentsBatchRunRetry.service.ts) - [`AICode/src/extension/src/aiTools/easyAgents/batch/run/EasyAgentsBatchViewerPresence.service.ts`] (https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FAiTools%2FeasyAgents%2Fbatch%2Frun%2FEasyAgentsBatchViewerPresence.service.ts) - [`AICode/src/extension/src/aiTools/easyAgents/batch/run/EasyAgentsBatchLastViewerStop.service.ts`] (https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FAiTools%2FeasyAgents%2Fbatch%2Frun%2FEasyAgentsBatchLastViewerStop.service.ts) -
[`AICode/src/extension/src/aiTools/easyAgents/batch/run/\_\_test\_\_/EasyAgentsBatchLastViewerStop.service.test.ts`] (https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FAiTools%2FeasyAgents%2Fbatch%2Frun%2F__test__%2FEasyAgentsBatchLastViewerStop.service.test.ts) -
[`AICode/src/extension/src/aiTools/easyAgents/batch/run/\_\_test\_\_/EasyAgentsBatchRunChildExecution.service.test.ts`] (https://aicode.local/open-file?path=AICode%2Fsrc%2Fextension%2Fsrc%2FAiTools%2FeasyAgents%2Fbatch%2Frun%2F__test__%2FEasyAgentsBatchRunChildExecution.service.test.ts) -

[`AICode/src/extension/src/aiTools/easyAgents/batch/run/\_\_\_test\_\_\_/EasyAgentsBatchRunExecution.activeExecution.service.test.ts`](https://aicode.local/open-file?
path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F___test___%2FEasyAgentsBatchRunExecution.activeExecution.service.test.ts) -

[`AICode/src/extension/src/aiTools/easyAgents/batch/run/\_\_\_test\_\_\_/EasyAgentsBatchRunExecution.kill.service.test.ts`]
(https://aicode.local/open-file?
path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F___test___%2FEasyAgentsBatchRunExecution.kill.service.test.ts) -

[`AICode/src/extension/src/aiTools/easyAgents/batch/run/\_\_\_test\_\_\_/EasyAgentsBatchRunExecution.testHarness.ts`]
(https://aicode.local/open-file?
path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F___test___%2FEasyAgentsBatchRunExecution.testHarness.ts) -

[`AICode/src/extension/src/aiTools/easyAgents/batch/run/\_\_\_test\_\_\_/EasyAgentsBatchRunExecution.live-activity-and-gating.service.test.ts`](https://aicode.local/open-file?
path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F___test___%2FEasyAgentsBatchRunExecution.live-activity-and-gating.service.test.ts) -

[`AICode/src/extension/src/aiTools/easyAgents/batch/run/\_\_\_test\_\_\_/EasyAgentsBatchRunExecution.commit-and-recovery-failures.service.test.ts`](https://aicode.local/open-file?
path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F___test___%2FEasyAgentsBatchRunExecution.commit-and-recovery-failures.service.test.ts) -

[`AICode/src/extension/src/aiTools/easyAgents/batch/run/\_\_\_test\_\_\_/EasyAgentsBatchRunExecution.post-commit-persistence.service.test.ts`](https://aicode.local/open-file?
path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F___test___%2FEasyAgentsBatchRunExecution.post-commit-persistence.service.test.ts) -

[`AICode/src/extension/src/aiTools/easyAgents/batch/run/\_\_\_test\_\_\_/EasyAgentsBatchRunLifecycle.service.test.ts`](https://aicode.local/open-file?
path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F___test___%2FEasyAgentsBatchRunLifecycle.service.test.ts) -

[`AICode/src/extension/src/aiTools/easyAgents/batch/run/\_\_\_test\_\_\_/EasyAgentsBatchRunRetry.testHarness.ts`](https://aicode.local/open-file?
path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F___test___%2FEasyAgentsBatchRunRetry.testHarness.ts) -

[`AICode/src/extension/src/aiTools/easyAgents/batch/run/\_\_\_test\_\_\_/EasyAgentsBatchRunRetry.live-restart.service.test.ts`]
(https://aicode.local/open-file?
path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F___test___%2FEasyAgentsBatchRunRetry.live-restart.service.test.ts) -

[`AICode/src/extension/src/aiTools/easyAgents/batch/run/\_\_\_test\_\_\_/EasyAgentsBatchViewerPresence.service.test.ts`]
(https://aicode.local/open-file?
path=AICode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F___test___%2FEasyAgentsBatchViewerPresence.service.test.ts) - [`AICode/src/extension/src/messageBus/handlers/registerAllBusHandlers.ts`](https://aicode.local/open-file?
path=AICode%2Fsrc%2Fextension%2Fsrc%2FmessageBus%2Fhandlers%2FregisterAllBusHandlers.ts) -

[`AICode/src/sharedlib/src/types/busTypes/global/MessageRegistry.ts`](https://aicode.local/open-file?
path=AICode%2Fsrc%2Fsharedlib%2Fsrc%2Ftypes%2FbusTypes%2Fglobal%2FMessageRegistry.ts) -

[`AICode/src/extension/src/extension/panels/core/openPanel.util.ts`](https://aicode.local/open-file?
path=AICode%2Fsrc%2Fextension%2Fsrc%2Fextension%2Fpanels%2Fcore%2FopenPanel.util.ts) -

[`AICode/src/extension/src/extension/panels/core/\_\_\_test\_\_\_/openPanel.util.test.ts`](https://aicode.local/open-file?
path=AICode%2Fsrc%2Fextension%2Fsrc%2Fextension%2Fpanels%2Fcore%2F___test___%2FopenPanel.util.test.ts) -

[`AICode/src/frontend/src/messageBus/AiStreamStartArgs.type.ts`](https://aicode.local/open-file?
path=AICode%2Fsrc%2Ffrontend%2Fsrc%2FmessageBus%2FAiStreamStartArgs.type.ts) -

[`AICode/src/frontend/src/messageBus/useAiStream.hook.ts`](https://aicode.local/open-file?
path=AICode%2Fsrc%2Ffrontend%2Fsrc%2FmessageBus%2FuseAiStream.hook.ts) -

[`AICode/src/frontend/src/app/prompt/Prompt/usePromptRunController.hook.ts`](https://aicode.local/open-file?
path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2FPrompt%2FPrompt%2FusePromptRunController.hook.ts) -

[`AICode/src/frontend/src/app/prompt/Prompt/Prompt.tsx`](https://aicode.local/open-file?
path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2FPrompt%2FPrompt%2FPrompt.tsx) -

[`AIcode/src/frontend/src/app/prompt/PromptInput/PromptInput.tsx`](https://aicode.local/open-file?path=AIcode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Fprompt%2FPromptInput%2FPromptInput.tsx) -
[`AIcode/src/frontend/src/app/prompt/PromptInput/PromptInput.module.css`](https://aicode.local/open-file?path=AIcode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Fprompt%2FPromptInput%2FPromptInput.module.css) -
[`AIcode/src/frontend/src/app/prompt/PromptInput/\_\_\_test\_\_\_/PromptInput.batchLiveViewerMode.test.tsx`](https://aicode.local/open-file?path=AIcode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Fprompt%2FPromptInput%2F___test___%2FPromptInput.batchLiveViewerMode.test.tsx) -
[`AIcode/src/frontend/src/app/easy-agents/easy-agents-batch/core/useEasyAgentsBatchController.hook.ts`](https://aicode.local/open-file?path=AIcode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Feasy-agents%2Feasy-agents-batch%2Fcore%2FuseEasyAgentsBatchController.hook.ts) - [AIcode/src/frontend/src/app/easy-agents/easy-agents-batch/core/___test___/EasyAgentsBatch.liveWatch.test.tsx`](https://aicode.local/open-file?path=AIcode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Feasy-agents%2Feasy-agents-batch%2Fcore%2F\_\_\_test\_\_\_%2FEasyAgentsBatch.liveWatch.test.tsx) - [AIcode/src/frontend/src/app/easy-agents/easy-agents-item/core/batch/useEasyAgentBatchRunController.hook.ts`](https://aicode.local/open-file?path=AIcode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Feasy-agents%2Feasy-agents-item%2Fcore%2Fbatch%2FuseEasyAgentBatchRunController.hook.ts) - [AIcode/src/frontend/src/app/easy-agents/easy-agents-item/core/batch/___test___/EasyAgentBatchSurface.test.tsx`](https://aicode.local/open-file?path=AIcode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Feasy-agents%2Feasy-agents-item%2Fcore%2Fbatch%2F___test___%2FEasyAgentBatchSurface.test.tsx) ### Déplacés et refundus -
[AIcode/src/extension/src/aiTools/easyAgents/batch/run/EasyAgentsBatchLiveObserver.service.ts`](https://aicode.local/open-file?path=AIcode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2FEasyAgentsBatchLiveObserver.service.ts) → `AIcode/src/extension/src/aiTools/orch/liveSession/registry/LiveSessionRegistry.service.ts` -
[AIcode/src/extension/src/aiTools/easyAgents/batch/run/___test___/EasyAgentsBatchLiveObserver.service.test.ts`](https://aicode.local/open-file?path=AIcode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F\_\_\_test\_\_\_%2FEasyAgentsBatchLiveObserver.service.test.ts) → `AIcode/src/extension/src/aiTools/orch/liveSession/registry/___test___/LiveSessionRegistry.service.test.ts` -
[AIcode/src/extension/src/aiTools/easyAgents/batch/run/EasyAgentsBatchLiveOpen.service.ts`](https://aicode.local/open-file?path=AIcode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2FEasyAgentsBatchLiveOpen.service.ts) → `AIcode/src/extension/src/aiTools/orch/liveSession/open/LiveSessionOpen.service.ts` -
[AIcode/src/extension/src/aiTools/easyAgents/batch/run/___test___/EasyAgentsBatchLiveOpen.service.test.ts`](https://aicode.local/open-file?path=AIcode%2Fsrc%2Fextension%2Fsrc%2FaiTools%2FeasyAgents%2Fbatch%2Frun%2F___test___%2FEasyAgentsBatchLiveOpen.service.test.ts) →
`AIcode/src/extension/src/aiTools/orch/liveSession/open/\_\_\_test\_\_\_/LiveSessionOpen.service.test.ts` -
[AIcode/src/extension/src/slimbk-local/endpoints/ai/easyAgents/batch/run/openLiveChild.endpoint.ts`](https://aicode.local/open-file?path=AIcode%2Fsrc%2Fextension%2Fsrc%2Fslimbk-local%2Fendpoints%2Fai%2FeasyAgents%2Fbatch%2Frun%2FopenLiveChild.endpoint.ts) →
`AIcode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/open.endpoint.ts` -
[AIcode/src/extension/src/slimbk-local/endpoints/ai/easyAgents/batch/run/___test___/openLiveChild.endpoint.test.ts`](https://aicode.local/open-file?path=AIcode%2Fsrc%2Fextension%2Fsrc%2Fslimbk-local%2Fendpoints%2Fai%2FeasyAgents%2Fbatch%2Frun%2F___test___%2FopenLiveChild.endpoint.test.ts) →
`AIcode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/\_\_\_test\_\_\_/open.endpoint.test.ts` -
[AIcode/src/extension/src/slimbk-local/endpoints/ai/easyAgents/batch/run/streamLiveChild.endpoint.ts`](https://aicode.local/open-file?path=AIcode%2Fsrc%2Fextension%2Fsrc%2Fslimbk-local%2Fendpoints%2Fai%2FeasyAgents%2Fbatch%2Frun%2FstreamLiveChild.endpoint.ts) →
`AIcode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/stream.endpoint.ts` -
[AIcode/src/extension/src/slimbk-local/endpoints/ai/easyAgents/batch/run/___test___/streamLiveChild.endpoint.test.ts`](https://aicode.local/open-file?path=AIcode%2Fsrc%2Fextension%2Fsrc%2Fslimbk-local%2Fendpoints%2Fai%2FeasyAgents%2Fbatch%2Frun%2F___test___%2FstreamLiveChild.endpoint.test.ts) →
`AIcode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/\_\_\_test\_\_\_/stream.endpoint.test.ts` -
[AIcode/src/extension/src/messageBus/handlers/ia/easyAgentsBatchLiveStreamHandlers.ts`](https://aicode.local/open-file?path=AIcode%2Fsrc%2Fextension%2Fsrc%2FmessageBus%2Fhandlers%2Fia%2FeasyAgentsBatchLiveStreamHandlers.ts)

path=AICode%2Fsrc%2Fextension%2Fsrc%2FmessageBus%2Fhandlers%2Fia%2FeasyAgentsBatchLiveStreamHandlers.ts) → `AICode/src/extension/src/messageBus/handlers/ia/liveSessionStreamHandlers.ts` -
 [`AICode/src/extension/src/messageBus/handlers/ia/\_\_test\_\_/easyAgentsBatchLiveStreamHandlers.test.ts`](https://aicode.local/open-file?
 path=AICode%2Fsrc%2Fextension%2Fsrc%2FmessageBus%2Fhandlers%2Fia%2F__test__%2FeasyAgentsBatchLiveStreamHandlers.test.ts) → `AICode/src/extension/src/messageBus/handlers/ia/\_\_test\_\_/liveSessionStreamHandlers.test.ts` -
 [`AICode/src/frontend/src/app/prompt/Prompt/PromptRuntimeMode.type.ts`](https://aicode.local/open-file?
 path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Fprompt%2FPrompt%2FPromptRuntimeMode.type.ts) →
 `AICode/src/frontend/src/app/prompt/liveSession/core/PromptLiveSession.types.ts` -
 [`AICode/src/frontend/src/app/prompt/Prompt/usePromptBatchLiveViewer.hook.ts`](https://aicode.local/open-file?
 path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Fprompt%2FPrompt%2FusePromptBatchLiveViewer.hook.ts) →
 `AICode/src/frontend/src/app/prompt/liveSession/core/usePromptLiveSession.hook.ts` -
 [`AICode/src/frontend/src/app/prompt/Prompt/\_\_test\_\_/Prompt.batchLiveViewer.hydraton.test.tsx`](https://aicode.local/open-file?
 path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Fprompt%2FPrompt%2F__test__%2FPrompt.batchLiveViewer.hydraton.test.tsx) → `AICode/src/frontend/src/app/prompt/liveSession/core/\_\_test\_\_/usePromptLiveSession.hydraton.test.tsx` -
 [`AICode/src/frontend/src/app/prompt/Prompt/\_\_test\_\_/Prompt.batchLiveViewer.handoff.test.tsx`](https://aicode.local/open-file?
 path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Fprompt%2FPrompt%2F__test__%2FPrompt.batchLiveViewer.handoff.test.tsx) → `AICode/src/frontend/src/app/prompt/liveSession/core/\_\_test\_\_/usePromptLiveSession.handoff.test.tsx` -
 [`AICode/src/frontend/src/app/prompt/PromptInput/usePromptInputViewerMode.hook.ts`](https://aicode.local/open-file?
 path=AICode%2Fsrc%2Ffrontend%2Fsrc%2Fapp%2Fprompt%2FPromptInput%2FusePromptInputViewerMode.hook.ts) →
 `AICode/src/frontend/src/app/prompt/liveSession/adapters/EasyAgentsBatchLiveSession.adapter.ts` -
 [`AICode/src/frontend/src/messageBus/\_\_test\_\_/useAiStream.hook.easyAgentsBatchLiveObserver.test.ts`](https://aicode.local/open-file?
 path=AICode%2Fsrc%2Ffrontend%2Fsrc%2FmessageBus%2F__test__%2FuseAiStream.hook.easyAgentsBatchLiveObserver.test.ts) → `AICode/src/frontend/src/messageBus/\_\_test\_\_/useAiStream.hook.liveSessionObserver.test.ts` ###
 Supprimés Aucun fichier indépendamment des déplacements. ### Fichiers générés Aucun `\*.generated.\*` n'est modifié.
 La génération normale produit les nouveaux clients/routes, ajoute la route abort exacte et retire les anciennes routes générées. ## Facteur de risque de régression 1. **Risque calculé : 62 %.** Criticité `4`, couplage `5`, blast radius `5`, incertitude `3`, faible confiance `2`. Base `77` ; `round(77 × 4/5)=62 %`. 2. **Zones complexes : ** identité run/lease transportée dès le Batch, exact abort du bouton Stop, source-start registration, armement/fermeture du Readable Orch lazy, erreurs première lecture/source-start/settlement, époque locator anti-ABA, permits et revalidation post-I/O, tombstones, replay post-handoff, ancien/nouveau run, TTL et index actif, settlement sur erreurs parser/callback, éviction pré-baseline, rejet append post-settlement, attente sans réveil perdu, curseurs suffixes, distinction TOOL_USE/vrai terminal live et persistant, snapshot terminal atomique sans second chat/open, branches Git/pause/done, ownership Activité/son, snapshot live-safe, union vs transport Slimbk, bootstrap invalide, dispose et mocks déplacés. 3. **Justification : ** l'UX ne change pas, mais le mécanisme traverse shared, extension et frontend. Une erreur d'identité, d'ordre, de terminalité, de lease, d'abort, de curseur, de cleanup du stream lazy ou de revalidation asynchrone peut perdre un chunk, fermer trop tôt, bloquer le handoff, conserver un `inFlightKey`, supprimer le mauvais run, relire le mauvais tour, interrompre une nouvelle génération ou tuer un Batch. Le turn transporté dès le CTA, l'abort exact sans await intermédiaire, l'enregistrement explicite, le priming/cleanup de l'itérateur, les index exacts, l'époque locator, les permits, les subscriptions liées, le snapshot terminal retourné atomiquement, le helper terminal partagé, l'éviction pré-baseline, les tombstones, la révision et les tests de courses réduisent le risque sans l'annuler. ## Résumé - Extraire un module `liveSession` générique structuré dans Orch, Prompt et shared. - Garder EasyAgents comme adaptateur/policy uniquement. - Identifier le run par locator+turn dès le `Watch live`, le panneau par lease et le parent par groupe JSON canonique. - Publier ligne+turn atomiquement dans l'état Batch live-only. - Après retour de `askStream`, armer l'itérateur Orch lazy, enregistrer le run avant de traiter le premier chunk et garantir la fermeture/observation de la première lecture sur échec source-start. - Garantir settlement exact une fois pour tout stream retourné sans permettre à un source-start échoué de solder un record préexistant. - Conserver leases actives après handoff pour drainage, sans viewer count ni TTL opening. - Rejouer uniquement les suffixes non consommés, sans modifier le buffer. - Corriger le réveil manqué via révision monotone. - Ajouter une époque monotone par locator et des permits d'ouverture pour empêcher les résultats post-I/O obsolètes, y compris les ABA. - Garantir settlement sur tout stream retourné, interdire append ultérieur et handoff avant baseline+settlement. - Évincer immédiatement un record settled sans baseline, impossible à ouvrir ou handoff, afin d'éviter toute rétention infinie. - Ne considérer comme terminal live que `meta.error` ou un `meta.stop\_reason` hors `TOOL\_USE`, avec helper partagé backend/frontend. - Ne considérer comme terminal persistant que le dernier assistant exact portant une

erreur hydratable ou un stop reason terminal valide ; la présence du turn ou TOOL_USE seul ne suffit pas. - Drainer puis échouer si handoff sans vrai terminal, afin que `useAiStream.onError` fournisse le meta terminal frontend sans falsifier le snapshot persistant. - Faire retourner par Open le snapshot terminal complet exact et l'hydrater directement ; supprimer le second `/api/ai/orch/chat/open` du chemin liveSession et sa fenêtre TOCTOU. - Sécuriser le bouton Stop existant avec une route abort locator+turn+lease ; aucune nouvelle action UI. - Préserver lease sur snapshot manquant, contrats Slimbk, canal générique, UX/TTL/Queue A/B et cleanup Prompt robuste. - Ne modifier ni sessions normales, ni Activité, ni SidebarStore, ni `PromptThreadCleanup`, ni fichiers générés. ## Nom du commit `refactor(live-session): extract EasyAgents live watcher into generic orch modules`

</SPECIF_EXTRAIRE_LE_LIVE_WATCHER_EASYAGENTS_TRANCHE_20260711_152702>

</USER>

<ASSISTANT>

I'm thinking...

</ASSISTANT>

<ASSISTANT>

🔧 ✅ scanning git: 🦋 ["État Git actuel"](#), 🦋 ["Métadonnées du commit"](#), 🦋 ["Fichiers du commit"](#), 🦋 ["Statistiques du commit"](#)

🔧 ✅ reading project: 📁 t ["Module backend liveSession"](#), 📁 t ["Module frontend liveSession"](#), 📁 t ["Contrats liveSession"](#), 📄 [liveSession.programming-doc-readme.md](#), 📄 [easyAgents.programming-doc-readme.md](#) [L930-L1110],

📄 [LiveSessionService.ts](#),

📄 [LiveSessionRecord.type.ts](#),

📄 [LiveSessionOpen.service.ts](#),

📄 [LiveSessionReplay.service.ts](#),

📄 [LiveSessionRun.type.ts](#),

📄 [LiveSessionOpenResult.type.ts](#),

📄 [LiveSessionPanel.type.ts](#),

📄 [LiveSessionStream.util.ts](#),

📄 [LiveSessionRegistry.service.ts](#),

📄 [EasyAgentsBatchLiveSessionAdapter.service.ts](#),

📄 [liveSessionStreamHandlers.ts](#),

📄 [abort.endpoint.ts](#),

📄 [PromptLiveSession.types.ts](#),

📄 [usePromptLiveSession.hook.ts](#),

📄 [EasyAgentsBatchLiveSession.adapter.ts](#), 📄 [AiStreamStartArgs.type.ts](#), 🦋 e ["Identité Retry Prompt"](#), 🦋 e ["Identité Continue Prompt"](#), 🦋 e ["Identité Edit Prompt"](#), 🦋 h ["Tours réutilisés Prompt"](#), 🦋 h ["Run actif et query types"](#), 🦋 ["Identités de tours Prompt"](#)

🔧 ✅ scanning git: 🦋 ["Diff version générée"](#), 🦋 ["Diff contrats générés"](#), 🦋 ["Commit HEAD actuel"](#), 🦋 ["Historique après tranche"](#)

🔧 ✅ reading project: 🦋 e ["Branche liveSession Prompt"](#), 🦋 e ["Policy fermeture live"](#), 🦋 e ["Ownership source live"](#), 🦋 e ["Enregistrement source live"](#), 🦋 e ["LiveSession dans cleanup"](#), 🦋 e ["LiveSession dans sidebar"](#), 🦋 e ["Descripteur release panel"](#), 🦋 e ["Parsing descripteur live"](#), 🦋 h ["Cycle panneau liveSession"](#), 🦋 ["Cycle exact panneau live"](#), 📄 [EasyAgentsBatchRunChildExecution.service.ts](#), 🦋 e ["Guard source registration"](#), 🦋 ["Adaptation source Easy"](#),

📄 [iaStreamHandlers.ts](#),

📄 [stream.endpoint.ts](#),

📄 [SidebarQueryRunningBridge.ts](#),

📄 [PromptThreadCleanup.manager.ts](#),

📄 [SidebarStore.queryWindow.listeners.manager.ts](#),

📄 [SidebarStore.queryWindow.mutations.manager.ts](#), 📄 [SidebarStore.rehydrate.manager.ts](#), 🦋 e ["Usage mode viewer Prompt"](#), 🦋 e ["Mode runtime Prompt actuel"](#), 🦋 ["Intégration mode Prompt"](#), 📄 [Prompt.tsx](#) [L100-L260],

📄 [Prompt.tsx](#) [L560-L710], 📄 [Prompt.tsx](#) [L960-L1100], 🦋 e ["Ownership cleanup Prompt"](#),

📄 [Prompt.tsx](#) [L500-L585], 🦋 e ["Verrou final Prompt"](#), 📄 [Prompt.tsx](#) [L460-L510], 🦋 e ["Interactions Prompt désactivées"](#),

📄 [Prompt.tsx](#) [L1280-L1337], 🦋 e ["Références query windows"](#), 🦋 e ["Type QueryWindowItem"](#),

🦋 e ["Champ windows direct"](#), 🦋 e ["Suppression Query window"](#), 🦋 e ["Ajout Query window"](#),

📄 [SidebarActivityQuery.tsx](#), 📄 [SidebarStore.handlers.manager.ts](#), 🦋 e ["Focus Query window"](#),

📄 [SidebarStore.ts](#), 🦋 e ["Rendu view model PromptInput"](#), 🦋 e ["showComposer"](#)

[PromptInput](#)",  [PromptInput.tsx](#) [L1810-L2035],  [Prompt.tsx](#) [L900-L975],  [usePromptRunController.hook.ts](#) [L500-L550]

  scanning git:  "Contrôle patch Git"

  reading project:  e "[Anciens symboles observer](#)",  e "[Anciens endpoints live](#)",  e "[Ancien canal live](#)",  e "[Ancien mode Batch live](#)",  e "[Ancien hook viewer](#)",  h "[Tests registre critiques](#)",  h "[Tests ouverture critiques](#)",  h "[Tests source lazy](#)",  h "[Tests frontend handoff](#)",  h "[Tests dispose panneau](#)",  e "[Test résurrection run](#)",  h "[Tombstone run tests](#)",  e "[Condition fallback persisté](#)",  [OrchAskStreamTurnBootstrap.ts](#) [L100-L210],  [OrchAskStreamTurnBootstrap.ts](#) [L210-L390],  e "[Buffer restant EasyAgents](#)",  e "[Leases restantes EasyAgents](#)",  e "[Replay restant EasyAgents](#)",  e "[Import EasyAgents dans core](#)",  e "[Batch dans core frontend](#)",  e "[Batch dans composant générique](#)",  [uiGenericHandlers.ts](#) [L1-L100],  [PromptQueryDoneSound.manager.ts](#),  [package.json](#) [L1-L20],  [package.json](#) [L1-L20],  [package.json](#) [L1-L20],  [usePromptRunController.hook.ts](#) [L1-L110]

Verdict

Oui : la tranche 1 correspond très largement à sa spécification et constitue une base saine pour commencer la tranche 2. 

Le commit `f65adc8e5172cc1e4f9781bf54ffa6dd089d3d33` est bien :

- le HEAD actuel de main ;
- synchronisé avec `origin/main` ;
- dans un espace de travail propre ;
- composé de **99 changements de fichiers**, avec environ **11 107 ajouts** et **4 572 suppressions** ;
- conforme au découpage backend Orch / frontend Prompt / shared demandé.

La réalisation est sérieuse : les courses les plus dangereuses ont été traitées, les tests sont beaucoup plus riches que l'ancien système, les anciens symboles EasyAgents ont disparu et le cœur générique ne dépend plus d'EasyAgents. C'est une tranche 1 particulièrement solide — tu as fait faire le gros déménagement sans laisser les meubles dans l'escalier 😊.

Mais la tranche 2 ne doit pas être conçue comme un simple branchement des sessions normales sur les API actuelles. La relecture du code normal révèle une correction architecturale obligatoire au début de la tranche 2 : **locator + turnId identifie un tour logique, mais pas une exécution de stream unique.**

Les actions `Retry`, `Edit last prompt` et `Continue response` réutilisent volontairement le même `turnId`. Le registre tranche 1 considère au contraire qu'un run exact ne peut vivre qu'une seule fois. Il faudra donc ajouter une identité d'**instance de source**, par exemple `sourceId`.

Ce n'est pas une régression de la tranche 1 : EasyAgents crée un `turnId` neuf par source et respecte parfaitement le contrat actuel. C'est une extension indispensable pour couvrir **toutes** les sessions normales.

Périmètre de l'audit réalisé

J'ai vérifié :

1. l'inventaire Git exact du commit ;
2. l'architecture réellement créée ;
3. les programming docs après extraction ;
4. la disparition des anciennes implémentations EasyAgents ;
5. le registre, les leases, les époques locator et les tombstones ;
6. l'ouverture live-safe et les snapshots terminaux ;
7. le replay backlog + tail ;
8. le lifecycle du `Readable lazy` ;
9. le pont EasyAgents ;
10. le transport Slimbk et message bus ;
11. le hook frontend et le handoff terminal ;
12. le dispose des panneaux ;
13. les tests des courses critiques ;
14. la compatibilité de cette fondation avec les sessions normales.

J'ai lu en particulier :

- [programming doc liveSession](#)
- [programming doc EasyAgents mise à jour](#)
- [façade LiveSessionService](#)
- [registre exact liveSession](#)
- [service d'ouverture liveSession](#)
- [service de replay liveSession](#)

- [hook frontend liveSession](#)
- [adaptateur EasyAgents frontend](#)
- [exécuteur child EasyAgents](#)
- [cycle de vie actuel des panneaux](#)
- [nettoyage Prompt actuel](#)

Conformité de la tranche 1

Domaine	Constat dans l'implémentation	Verdict
Architecture backend	Module composé en <code>core</code> , <code>registry</code> , <code>open</code> et <code>replay</code> sous liveSession Orch.	✓
Architecture frontend	Séparation <code>core</code> , <code>adapters</code> , <code>components</code> sous liveSession Prompt.	✓
Contrats shared	Run refs, résultats Open, paramètres panneau et helper terminal sous liveSessionTypes.	✓
Indépendance du cœur	Aucun import ni texte EasyAgents dans le cœur backend, le cœur frontend ou le composant générique.	✓
Adaptation EasyAgents	EasyAgents ne conserve que le groupe parent, readiness, hard-stop policy et wiring Batch.	✓
Ancienne implémentation	Les anciens observer/open/services/endpoints/canaux Batch ont été supprimés ou déplacés.	✓
Identité exacte	Locator complet + <code>turnId</code> transporté dans Open, replay, abort et release.	✓ pour la tranche 1
Leases	<code>opening</code> , <code>attached</code> , <code>retired</code> , TTL et release exacte sont implémentés.	✓
Époque locator	Époque monotone et revalidation post-I/O, y compris ABA.	✓
Source lazy	Le premier <code>next()</code> est armé avant le source-start ; erreurs et cleanup sont observés.	✓
Settlement	Exactement une tentative par stream retourné ; guard local EasyAgents.	✓
Settlement sans baseline	Éviction immédiate.	✓
Terminalité	<code>TOOL_USE</code> explicitement non terminal ; helper partagé backend/frontend.	✓
Snapshot terminal	Snapshot exact retourné par Open, sans second <code>/chat/open</code> .	✓
Replay	Backlog + tail par curseur, suffixes readonly, contrôle lease par élément.	✓
Réveil perdu	Listener installé avant relecture de la révision.	✓
Abort	Autorisation exacte run + lease, sans fallback thread-only.	✓
UX EasyAgents	Labels, banner, Stop line, Retry handoff et read-only conservés.	✓
Queue A / Queue B	Non déplacées et algorithmes inchangés.	✓
Sidebar / Activité	Non modifiées dans cette tranche.	✓
Nettoyage normal	PromptThreadCleanup.manager.ts non modifié.	✓
Documentation	Doc générique créée et doc EasyAgents réduite à sa policy produit.	✓
Tests	Registre, Open, Replay, façade, adapter, source lazy, endpoints, transport, dispose, frontend et handoff couverts.	✓

Vérification de la séparation architecturale

Les anciennes responsabilités ne subsistent plus dans EasyAgents :

- aucun `bufferedChunks` dans le domaine EasyAgents ;
- aucune deadline de lease dans EasyAgents ;
- aucun moteur de replay dans EasyAgents ;
- aucun ancien canal `easyAgentsBatchLiveStream` ;
- aucun ancien mode `batchLiveObserver` ;

- aucun ancien hook `usePromptBatchLiveViewer` ;
- aucun ancien endpoint `openLiveChild`.

Le pont métier est bien limité à [EasyAgentsBatchLiveSessionAdapter.service.ts](#).

Le cœur générique n'importe pas EasyAgents, tandis que les textes comme `Watching live Batch line...` restent correctement confinés à [EasyAgentsBatchLiveSession.adapter.ts](#).

C'est exactement la séparation attendue.

Vérification des tests critiques

La couverture ajoutée correspond bien aux zones annoncées dans la spécification.

Registre

[LiveSessionRegistry.service.test.ts](#) couvre notamment :

- readiness baseline + observable ;
- leases multiples ;
- release exacte ;
- TTL ;
- ancien run contre nouveau run ;
- éviction sans baseline ;
- époque locator ;
- ABA ;
- terminal `TOOL_USE` ;
- handoff prématuré ;
- ancienne rétention qui ne détruit pas le nouveau run ;
- autorisation d'abort.

Open

[LiveSessionOpen.service.test.ts](#) couvre notamment :

- snapshot live filtré ;
- snapshot terminal exact ;
- erreur persistée ;
- timeout invalide ;
- assistant sans terminal ;
- `TOOL_USE` non terminal ;
- lease expirée ou released pendant la lecture ;
- nouveau run pendant la lecture ;
- ABA actif → nouveau → aucun ;
- absence de second read terminal.

Replay

[LiveSessionReplay.service.test.ts](#) couvre :

- backlog ;
- tail ;
- consumers multiples ;
- suffixes ;
- release pendant émission ;
- settlement qui ne ferme pas ;
- handoff ;
- absence de terminal ;
- `TOOL_USE` seul ;
- éviction.

Source lazy

[EasyAgentsBatchRunChildExecution.service.test.ts](#) couvre les chemins les plus délicats :

- première lecture primée ;
- source-start avant traitement métier ;
- source-start qui lève ;
- première lecture qui lève ;
- tail qui lève ;
- fermeture du générateur ;
- observation de la promesse pendante ;
- settlement exactement une fois ;

- erreur principale conservée devant les erreurs de cleanup ;
- absence de callback lorsque `askStream` lève avant retour.

Frontend

Les tests de [core liveSession frontend](#) couvrent :

- hydratation live ;
 - handoff exact ;
 - `TOOL_USE` ;
 - erreur injectée frontend ;
 - snapshot persistant non terminal ;
 - Retry single-flight ;
 - absence de `/chat/open` ;
 - refus d'un autre run.
- La règle « tout ce qui est programmé doit être testé » a donc été suivie avec sérieux.

Écarts mineurs entre la spécification et le commit

1. Les fichiers générés ont matériellement changé

La spécification indique littéralement qu'aucun `*.generated.*` ne serait modifié. Le commit contient pourtant :

- les tableaux d'endpoints Slimbk générés ;
- les clients locaux générés ;
- les contrats proxy générés ;
- [appVersion.generated.ts](#).

Pour les routes et clients, c'est cohérent avec une **régénération normale** après l'ajout des endpoints `liveSession`. Ce ne sont pas des contournements ni des augmentations de type.

Le changement de version de `2.2.0` vers `2.3.0` dans [appVersion.generated.ts](#) est également cohérent avec la version `2.3.0` déjà définie dans [package de distribution](#).

Je ne peux pas déterminer à partir du commit seul si ces fichiers ont été modifiés manuellement ou exclusivement par le générateur, mais leur contenu est cohérent avec une sortie de génération.

Impact : aucun blocage pour la tranche 2.

C'est surtout une imprécision littérale dans le récapitulatif de la spécification.

2. Le lifecycle source lazy reste actuellement dans EasyAgents

La logique extrêmement délicate de priming/fermeture/settlement est encore portée par [EasyAgentsBatchRunChildExecution.service.ts](#).

Cela correspond à la spécification de tranche 1, qui demandait explicitement cette modification dans le child executor. Ce n'est donc pas une non-conformité.

En revanche, pour la tranche 2, il ne faudra **pas recopier cette logique** dans l'endpoint normal. Elle devra être factorisée dans un nouveau sous-domaine `source/` du module backend `liveSession`, puis réutilisée par :

- EasyAgents ;
- le stream Prompt normal.

Ce sera l'un des premiers travaux internes de la tranche 2.

Le point bloquant découvert pour les sessions normales

locator + turnId n'est pas une identité d'exécution suffisante

Pour EasyAgents, chaque source possède un nouveau `turnId`. Le contrat actuel est donc correct.

Pour les sessions normales, plusieurs actions ouvrent une **nouvelle exécution** en conservant le même tour logique.

Continue response conserve le turnId

[AICode/src/frontend/src/app/prompt/Prompt/usePromptRunCommands.hook.ts](#)

Copy

```
// Keep the same turnId as the assistant being continued
const continueTurnId =
  lastAssistantTurnId ?? `turn:${uidGenerator.newUniqueId}`;
currentTurnIdRef.current = continueTurnId;

...

// Start continuation (same turnId)
```

Retry conserve également le turnId

[AICode/src/frontend/src/app/prompt/Prompt/usePromptRunCommands.hook.ts](#)

Copy

```
// Reuse SAME turnId as the last user of current turn
let lastTurnIdOfCurrentTurn: string | undefined = undefined;

...

const retryTurnId =
  lastTurnIdOfCurrentTurn ?? `turn:${uidGenerator.newUniqueId}`;
```

Il en va aussi ainsi pour :

- `editLastPrompt` ;
- un Retry de SPECIF ;
- plusieurs `continueResponse` successifs.

Le registre actuel interdit cette réutilisation

[AICode/src/extension/src/aiTools/orch/liveSession/registry/LiveSessionRegistry.service.ts](#)

Copy

```
if (this.retiredRunKeys.has(runKey)) {
  throw new Error('Conflicting live-session run registration.');
```

changes epochs.

```
  if (existing) {
    // ...
    return;
  }
}
```

Deux situations seraient donc possibles sans correction :

1. Le record précédent est encore retenu :

- `registerRun` considère la nouvelle source comme un appel idempotent ;
- il ne réinitialise pas le buffer ;
- le record reste `sourceSettled` et `handoffReady` ;
- le premier `appendChunk` de la nouvelle exécution échoue.

2. Le record précédent a été évincé :

- son run key est dans `retiredRunKeys` ;
- la nouvelle exécution est rejetée comme conflit.

Correction obligatoire

L'identité exacte doit devenir :

- locator ;
- `turnId` logique ;
- **`sourceId` unique par invocation de `askStream`.**

Le `turnId` reste indispensable pour :

- les messages persistés ;
- l'hydratation ;
- les chunks ;
- la mémoire Orch.

Le `sourceId` distingue :

- la première génération ;
- son Retry ;
- son Edit ;
- chaque continuation ;
- tout autre segment de stream partageant le même tour.

EasyAgents

EasyAgents peut utiliser explicitement :

- `sourceId = turnId`

car son invariant produit garantit une source unique pour chaque nouveau `turnId` de child.
Ce n'est pas un fallback : c'est une règle produit simple et testable. Cela évite d'ajouter un troisième champ `live-only` redondant dans l'état Batch.

Sessions normales

Le handler normal doit générer un nouveau `sourceId` pour chaque invocation de stream, avant de publier l'activité.
L'item Activité devra transporter :

- le `turnId` ;
- le `sourceId`.

Le Stop détaché et la réouverture devront valider les deux.

La tranche 2 peut-elle commencer ?

Oui.

Mais son début doit être considéré comme une **extension du cœur générique**, pas encore comme l'UX Activité.
Je recommande quatre sous-phases dans la tranche 2.

Sous-phase	Contenu	Précondition de la suivante
2A — Identité et sources normales	<code>sourceId</code> , lifecycle source générique, readiness et stratégies de replay.	Sources normales enregistrables sans panneau.
2B — Détachement du panneau	Fermeture sans abort, coordination mémoire, leases et finalisation différée.	Une source continue proprement panneau fermé.
2C — Réouverture Prompt	Adaptateur normal, compositeur standard, replay exact, continuation.	Affichage identique au stream original.
2D — Activité et Stop détaché	SidebarStore, troisième ligne, ouverture et arrêt exacts.	UX complète et courses couvertes.

Tranche 2A — Étendre le cœur liveSession

1. Introduire l'instance exacte de source

Le contrat partagé doit distinguer :

Identité	Signification
Locator	Session persistée.
<code>turnId</code>	Tour logique visible et persistant.
<code>sourceId</code>	Invocation exacte du stream.
Lease	Instance exacte d'un panneau rattaché à cette source.

Le run key devient donc structurellement :

```
[[sessionId, storageScopeRelPath], turnId, sourceId]
```

Cela maintient les garanties existantes :

- aucune concaténation ambiguë ;
- aucun fallback vers l'actif ;
- anciennes leases isolées ;
- ancien Stop incapable de tuer une nouvelle exécution ;
- anciennes opérations incapables d'append ou de solder une nouvelle source.

2. Généraliser les politiques de readiness

Le type actuel est volontairement limité :

[AICode/src/extension/src/aiTools/orch/liveSession/registry/LiveSessionRecord.type.ts](#)

Copy

```
/** Readiness policy used by tranche 1 EasyAgents sources. */  
export type LiveSessionReadiness = 'baselineAndObservableChunk';
```

Il faudra au moins trois contrats :

Type de source	Readiness	Pourquoi
EasyAgents	Baseline + chunk observable	Préserve exactement l'apparition actuelle de <code>Watch live</code> .
Prompt normal, Retry, Edit, SPECIF	Baseline <code>fallback_persisted</code>	Une réouverture normale doit être possible avant le premier token visible.
<code>continueResponse</code>	Baseline capturée avant	Aucun <code>fallback_persisted</code> n'est

Type de source

Readiness

Pourquoi

consommation

émis pour une continuation.

Le critère d'ouverture et la stratégie de reconstruction ne doivent pas être mélangés dans un seul enum monolithique.

Une structure plus propre séparerait :

- `readinessPolicy` ;
- `snapshotPolicy` ;
- `replayStrategy`.

Tranche 2A — Le cas spécial `continueResponse`

C'est la partie la plus délicate de la tranche 2.

Pourquoi le snapshot actuel ne suffit pas

[LiveSessionOpen.service.ts](#) retire actuellement tous les assistants du tour demandé, puis le frontend recrée des placeholders vides.

C'est correct pour :

- un nouveau prompt ;
- `Retry` ;
- `Edit` ;
- `SPECIF`.

Ce n'est pas correct pour `continueResponse`.

Une continuation :

- conserve l'assistant existant ;
- conserve son texte antérieur comme préfixe ;
- insère éventuellement un nouveau `thinking` avant cet assistant ;
- ajoute les nouveaux tokens au même message ;
- réutilise le même `turnId`.

Risque de duplication

Si Open lit le fichier persistant au moment de la réouverture, celui-ci peut déjà contenir une partie de la continuation. Puis le replay repart du premier chunk de cette continuation.

Le frontend afficherait alors :

- le préfixe original ;
- une partie déjà persistée ;
- puis cette même partie rejouée une seconde fois.

Solution

Pour `continueResponse`, le backend doit capturer **un snapshot baseline immuable avant le démarrage de la nouvelle consommation**.

Séquence recommandée :

1. détecter `aiQuery.type === 'continueResponse'` ;
2. lire le snapshot persistant avant `askStream`, ou avant le premier `next()` ;
3. appeler `askStream` ;
4. enregistrer la source avec :
 - `replay strategy continueAssistant` ;
 - `snapshot baseline capturé` ;
5. primer l'itérateur ;
6. consommer et bufferiser les nouveaux chunks.

Le résultat Open `live` doit indiquer la stratégie de replay.

Le frontend doit alors :

- hydrater le `snapshot baseline` ;
- retrouver l'assistant exact du `turnId` ;
- conserver son texte comme `baseAssistantTextRef` ;
- cibler son `UID` hydraté ;
- insérer le `thinking` de continuation ;
- rejouer seulement les nouveaux chunks.

La préparation actuelle de `handleContinueResponse` devrait être factorisée afin que :

- le clic `Continue` normal ;
 - le replay d'une continuation ;
- utilisent la même primitive de préparation frontend.

Tranche 2A — Factoriser le lifecycle source lazy

Le code de priming présent dans [EasyAgentsBatchRunChildExecution.service.ts](#) ne doit pas être copié dans [stream.endpoint.ts](#) normal.

Le futur sous-domaine [source/](#) de [liveSession backend](#) doit encapsuler :

- récupération de l'itérateur ;
- première lecture pendante ;
- source-start synchrone ;
- consommation de la première lecture puis du tail ;
- abort best-effort ;
- destruction du `Readable` ;
- `iterator.return()` ;
- observation des promesses ;
- settlement exact une fois ;
- priorité de l'erreur principale ;
- agrégation déterministe des erreurs secondaires.

`EasyAgents` doit ensuite consommer cette primitive générique lui aussi. Cela évite d'avoir deux implémentations différentes de la partie la plus dangereuse du système.

Tranche 2A — Instrumenter la source normale au bon niveau

Le handler normal actuel est [iaStreamHandlers.ts](#).

Le vrai `askStream(...)` est cependant appelé dans [endpoint de stream Orch](#).

L'enregistrement `liveSession` doit se faire près de ce `askStream`, parce que c'est le seul emplacement qui sait exactement que :

- `askStream` a bien retourné ;
- le `Readable` existe ;
- aucun chunk n'a encore été consommé ;
- l'échec avant retour ne doit produire aucun record.

Ordre cible

1. Le handler extension crée un `sourceId`.
2. Il publie l'activité avec locator + turn + source.
3. L'endpoint prépare éventuellement la baseline de continuation.
4. L'endpoint appelle `askStream`.
5. Le lifecycle générique prime le `Readable`.
6. Il enregistre le run ownership `source`.
7. Chaque chunk est :
 - parsé ;
 - ajouté au buffer ;
 - utilisé pour marquer la baseline ;
 - transmis inchangé au transport existant.
8. Le stream complet se termine.
9. `sourceSettled` est signalé.
10. Pour une source normale, `handoffReady` peut suivre immédiatement la finalisation Orch.
11. Le handler termine l'activité et applique la policy panneau fermé/ouvert.

Ordre du chunk

L'append au buffer doit impérativement précéder l'émission vers le webview :

- si le panneau disparaît entre les deux opérations, le chunk est déjà récupérable ;
- le replay ne dépend jamais du succès de `postMessage`.

Tranche 2B — Fermer le panneau sans tuer le processus

Le comportement actuel reste explicitement destructif dans [PromptThreadCleanup.manager.ts](#) :

1. Abort ;
2. `AwaitStop` ;
3. `ClearMemory` ;
4. cleanup des attachments.

La tranche 2 doit séparer ces responsabilités.

Fermeture avec source normale active

La fermeture doit :

1. retirer l'association du panneau avec son thread ;
2. marquer le panneau `closed` ;
3. libérer sa lease live si le panneau était une réouverture ;
4. **ne pas abort** ;
5. **ne pas awaitStop** ;
6. **ne pas clear la mémoire Orch** ;
7. conserver l'activité et son timer ;
8. différer les nettoyages de persistance susceptibles de courir contre le stream.

Fermeture sans source active

Le chemin historique peut rester :

- arrêt si nécessaire ;
- await ;
- clear mémoire ;
- cleanup attachments ;
- cleanup chat trivial.

Coordination fin de source / réouverture

Une file ou un coordinateur par locator est nécessaire.

La course critique est :

1. la source finit ;
2. le panneau est encore fermé ;
3. le finalizer décide de clear la mémoire ;
4. l'utilisateur clique simultanément dans Activité ;
5. un nouveau panneau se crée.

Deux résultats valides sont possibles :

- **réouverture gagne** :
 - pas de clear ;
 - handoff live vers Prompt normal ;
- **cleanup gagne** :
 - mémoire clear ;
 - ouverture classique ;
 - réhydratation Orch par le chemin normal.

Un mélange des deux est interdit :

- panneau live déclaré normal ;
- mais mémoire Orch déjà effacée ;
- puis prochain Send exécuté sur une mémoire vide.

La coordination doit donc sérialiser :

- fermeture du panneau ;
- fin de source ;
- clear mémoire ;
- ouverture classique ou live.

Tranche 2C — Revenir à l'interface normale au plus tôt

La tranche 1 possède volontairement un frontend read-only générique, car seul EasyAgents l'utilise actuellement :

[AICode/src/frontend/src/app/prompt/liveSession/core/PromptLiveSession.types.ts](#)

Copy

```
/** True when Prompt is read-only under the generic live-session lifecycle. */
export function isPromptViewerRuntimeMode(mode: PromptRuntimeMode): boolean {
  return mode !== 'normal';
}
```

Ce contrat doit être assoupli.

Ne pas confondre trois axes

La tranche 2 doit séparer :

Axe

Exemples

Phase du rattachement opening, streaming, handoff, failed

Présentation produit viewer EasyAgents ou Prompt normal

Ownership de cleanup arrêt à la fermeture ou continuation

Actuellement, `promptRuntimeMode !== normal` implique simultanément :

- banner viewer ;
- compositeur caché ;
- interactions désactivées ;
- persistance du draft désactivée ;
- aucun ownership de cleanup.

Ce couplage est correct pour EasyAgents, mais pas pour une session normale rouverte.

Comportement cible d'une session normale rouverte

Phase	Compositeur	Saisie	Send	Stop
Snapshot non installé	Masqué ou temporairement bloqué	Non	Non	Non
Snapshot installé, replay armé	Compositeur normal	Oui	Bloqué	Stop normal
Rattrapage backlog	Compositeur normal	Oui	Bloqué	Stop normal
Tail live	Compositeur normal	Oui	Bloqué	Stop normal
Stream terminé et files drainées	Compositeur normal	Oui	Réactivé	Redevient Send

Le verrou supplémentaire doit durer uniquement pendant :

- l'installation du snapshot ;
- la préparation des cibles de replay ;
- l'armement du listener.

Ensuite, `isStreaming` fournit déjà le verrou normal de Send.

Il ne faut pas conserver le banner read-only EasyAgents pour une session normale.

Draft et attachments

Une session normale rouverte doit reprendre ses invariants normaux :

- le brouillon peut être édité ;
- le draft peut être persisté ;
- les attachments du prochain message peuvent être préparés ;
- Ctrl/Cmd+Enter reste bloqué tant que la génération tourne ;
- le bouton existant est Stop.

Cela implique de ne plus conditionner la persistance du draft uniquement à `promptRuntimeMode === 'normal'`. Elle devra être autorisée par la policy du produit normal dès que le bootstrap est installé.

Aucun second bouton Stop dans le Prompt

La tranche 1 confirme ce point :

- le viewer EasyAgents garde son `Stop line` existant ;
- le Prompt normal possède déjà son bouton Stop dans le composant de saisie ;
- la route `liveSession` exacte sécurise le Stop du viewer.

Pour la session normale rouverte :

- le compositeur normal sera affiché ;
- `running=true` fera apparaître son bouton Stop habituel ;
- aucune seconde action n'est nécessaire dans le panneau.

Le seul nouveau bouton reste celui d'Activité lorsque le panneau est fermé.

Tranche 2D — Évolution du SidebarStore

Le modèle actuel parle encore de fenêtres :

- `SidebarActivityQueryWindowItem` ;
- `queryState.windows`.

Ce vocabulaire devient faux lorsqu'un item peut exister alors qu'aucun panneau n'existe.

Modèle recommandé

Champ	Rôle
Locator	Session exacte.
Titre	Présentation.

Champ	Rôle
<code>panelState: 'open' 'closed'</code>	Seuls états autorisés.
<code>isRunning</code>	Source normale encore active.
<code>activeTurnId</code>	Tour logique.
<code>activeSourceId</code>	Exécution exacte.
<code>stopRequested</code>	Désactivation persistante après clic.
<code>liveQueryElapsedMs</code>	Temps live.
<code>lastQueryElapsedMs</code>	Dernière durée terminale.
Dates existantes	Tri stable.
Je recommande :	
• <code>SidebarActivityQueryWindowItem</code> → <code>SidebarActivityQueryItem</code> ; • <code>windows</code> → <code>items</code>.	
Le blast radius de ce renommage est limité : l’inventaire actuel montre une dizaine de références principales, principalement dans :	
• SidebarStore.ts • SidebarStore.queryWindow.listeners.manager.ts • SidebarStore.queryWindow.mutations.manager.ts • SidebarStore.rehydrate.manager.ts • SidebarActivityQuery.tsx • quelques endpoints et tests de chat.	
Condition exacte de la troisième ligne	
La troisième ligne existe si et seulement si :	
• <code>panelState === 'closed'</code> ; • <code>isRunning === true</code>.	
Le bouton est enabled si :	
• <code>stopRequested === false</code>.	
Il disparaît dès que :	
• le panneau est rouvert ; • ou la source s’arrête.	
Aucun concept de visible/invisible/active/background n’intervient.	

Ouverture depuis Activité

Le frontend Activité ne devrait pas essayer de reconstruire seul une query live à partir d’un état potentiellement légèrement périmé.

Le clic doit envoyer le locator à l’extension.

L’extension décide alors atomiquement :

État backend	Action
Panneau déjà ouvert	Reveal.
Source normale active	Fresh lease + query <code>liveSession</code> exacte.
Source terminée	Ouverture classique.
Source remplacée	Ouverture du run actif exact ou ouverture classique selon l’état.
Cela nécessite une API du cœur pour obtenir l’exacte source active d’un locator, sans exposer le record mutable.	

Ouverture avant `fallback_persisted`

Une nouvelle course existe :

1. Send démarre ;
2. Activité passe Running ;
3. panneau fermé immédiatement ;
4. panneau rouvert avant le premier `fallback_persisted`.

Le panneau ne doit pas afficher une erreur définitive `not_ready` puis se fermer.

Pour une source normale, il faut permettre :

- la réservation d’une lease `opening` avant `replay readiness` ;
- l’attente de la baseline ;
- le passage live dès que la baseline est prête ;
- l’échec uniquement si la source se termine sans baseline.

EasyAgents garde son contrat actuel : son CTA n’apparaît qu’une fois `replay-ready`.

Stop dans Activité

Le Stop Activité ne possède plus de panneau ni de lease active. Il ne doit donc pas réutiliser aveuglément la route viewer actuelle.

Je recommande une commande typed sidebar → extension dédiée.

Elle transporte :

- locator ;
- turnId ;
- sourceId.

L'extension valide synchroniquement :

1. la source exacte est toujours active ;
2. ownership source ;
3. source non settled ;
4. panneau toujours closed ;
5. identité exacte inchangée ;
6. stop non déjà accepté.

Puis elle appelle `abortStream(sessionId, reason)` dans la même pile.

Pourquoi le sourceId est essentiel ici

Sans lui :

1. source A du tour T finit ;
2. Retry source B réutilise le tour T ;
3. un vieux clic Stop provenant de l'item A arrive ;
4. il pourrait tuer B.

Avec sourceId, le clic A est rejeté.

Double comptage Activité et son

La tranche 1 prépare déjà correctement cette séparation :

- ownership observer ouvre les ressources Activité/son ;
- ownership source ne fait qu'un relay.

[liveSessionStreamHandlers.ts](#) possède déjà une branche source sans Activity ni son.

C'est exactement ce qu'il faut pour la session normale :

- le stream original [iaStreamHandlers.ts](#) reste seul propriétaire :
- du timer ;
- du statut Running ;
- de la dernière durée ;
- du son terminal ;
- le panneau reconnecté ne fait que rejouer les chunks.

Il ne doit donc jamais :

- redémarrer le timer ;
- remettre l'elapsed à zéro ;
- jouer deux sons ;
- exécuter deux transitions Running → Idle.

Handoff normal : faut-il recharger le snapshot terminal ?

La tranche 1 recharge le snapshot terminal parce qu'EasyAgents possède une convergence externe au stream Orch :

- Git ;
- état child ;
- état parent ;
- publication Batch.

Une session normale n'a pas ces étapes.

Chemin préféré

Après :

1. réception du vrai terminal meta ;
2. fermeture du stream replay ;
3. drainage de Queue A ;
4. drainage de Queue B ;

le frontend doit pouvoir passer directement à l'état normal.

Avantages :

- aucune requête Open terminal supplémentaire ;
- aucun verrou post-stream ;
- brouillon local intact ;
- attachments du prochain message intacts ;
- invariant normal retrouvé au plus tôt.

Validation indispensable

Il faut un test d'isomorphisme comparant exactement :

- une session streamée sans fermeture ;
- la même session fermée puis rouverte ;
- l'état React terminal ;
- les messages ;
- tool calls ;
- thinking ;
- usage ;
- grouped file edits ;
- notices ;
- gates ;
- métadonnées.

Si l'état est strictement identique, aucun reload terminal n'est nécessaire.

Fallback acceptable

Si une divergence structurelle est découverte :

- appeler l'Open terminal exact ;
- conserver le compositeur visible et éditable ;
- bloquer uniquement Send pendant cette courte réconciliation ;
- préserver draft et attachments locaux ;
- réactiver immédiatement après l'hydratation.

Le viewer EasyAgents reste inchangé.

Courses obligatoires à tester dans la tranche 2

Course	Résultat attendu
Fermeture avant enregistrement liveSession	La source n'est pas abortée ; le cleanup terminal est différé.
Fermeture avant baseline	Réouverture attend la baseline au lieu d'échouer définitivement.
Fermeture après baseline, avant token	Snapshot + replay corrects.
Réouverture pendant append	Aucun chunk perdu ni doublé.
Réouverture pendant terminal meta	Live ou classique, mais jamais snapshot intermédiaire.
Source termine pendant création du panneau	Ouverture live terminale ou classique cohérente.
Cleanup mémoire contre réouverture	Une seule branche gagne ; mémoire jamais vide sous un Prompt déclaré normal.
Ancien Stop Activité contre nouvelle source	Refus grâce au <code>sourceId</code> .
Ancienne lease contre nouvelle source du même tour	Refus.
Retry du même <code>turnId</code>	Nouveau buffer et nouvelle source exacts.
Plusieurs <code>continueResponse</code>	Baselines distinctes et aucun texte dupliqué.
Edit du même tour	Ancien contenu supprimé, nouveau replay exact.
Gate interactive pendant fermeture	Gate rejouée et décision toujours fonctionnelle.
Tool calls pendant fermeture	Résultats et adjacency reconstruits.
Source sans baseline	Éviction et cleanup sans panneau fantôme.
Panneau rouvert puis refermé	Nouvelle lease, release exacte, source continue.
Double clic Stop Activité	Un seul abort accepté.
Panneau rouvert avant le Stop Activité	Stop refusé.
Sidebar rechargée	Run détaché toujours présenté depuis l'état extension.
Stream original et replay simultané	Activité et son possédés uniquement par la source.

Point UX restant à figer dans la spécification de tranche 2

Il existe une seule ambiguïté résiduelle dans la formulation initiale :

Panneau fermé + source terminée

Deux politiques sont techniquement possibles :

Politique

Effet

Retirer l'item

Activité contient uniquement les panneaux ouverts et les processus détachés encore actifs.

Conserver l'item Idle à deux lignes

Activité conserve aussi les sessions détachées terminées.

Ma recommandation est la première :

Conserver l'item tant que le panneau est ouvert ou que la source est active ; le retirer lorsque le panneau est fermé et la source terminée.

Elle évite que la vue Activité devienne une seconde liste d'historique et reste fidèle à son rôle opérationnel.

Dans les deux cas, la règle sur la troisième ligne est identique :

- elle disparaît à la fin ;
- elle disparaît à la réouverture.

Risque réévalué de la tranche 2

Le risque est **plus élevé que celui de la tranche 1**, parce que la tranche 2 touche le chemin Prompt normal utilisé par toutes les sessions.

Facteur

Niveau

Criticité du domaine

Très élevée

Couplage stream/persistence/panel/sidebar

Très élevé

Blast radius

Élevé

Incertitude restante

Moyenne

Fondation et tests disponibles

Bons

J'évalue le risque de régression accidentelle autour de **80-87 % avant les tests de la tranche 2**.

Ce chiffre ne signifie pas que la fonctionnalité a peu de chances de réussir. Il signifie qu'une implémentation sans les garde-fous identifiés produirait presque certainement une course ou une divergence.

La tranche 1 réduit fortement ce risque grâce à :

- l'identité exacte ;
- les leases ;
- le buffer ;
- les époques ;
- le replay testé ;
- la branche ownership *source* déjà prévue ;
- le canal générique ;
- les snapshots exacts ;
- le traitement correct de `TOOL_USE`.

Conclusion finale

Tranche 1

Validée. 

Je ne vois pas de défaut dans la tranche 1 qui justifierait de la refaire ou de retarder artificiellement la suite.

Les seuls écarts sont :

- la présence matérielle de sorties `*.generated.*`, cohérentes avec la génération ;
- la mise à jour générée de version, sans impact sur `liveSession`.

Tranche 2

Faisable à partir de ce commit. 

Mais elle doit commencer par quatre extensions obligatoires :

1. ajouter un `sourceId` distinct du `turnId` ;
2. factoriser le lifecycle source lazy dans le module générique ;
3. ajouter les stratégies de baseline/replay, surtout pour `continueResponse` ;
4. découpler phase `liveSession`, présentation Prompt et ownership de cleanup.

Une fois ces quatre fondations posées :

- la fermeture peut cesser d'aborder ;
- le stream normal peut continuer dans l'extension ;

- Activité peut conserver le processus détaché ;
- la réouverture peut utiliser le buffer existant ;
- le compositeur normal peut revenir immédiatement après le bootstrap ;
- le Stop normal existant peut être réutilisé dans le panneau ;
- le nouveau Stop Activité peut être sécurisé par locator + turn + source ;
- aucune duplication d'Activité ou de son n'est nécessaire.

TL-DR

- Le commit de tranche 1 est propre, complet et conforme à son objectif EasyAgents.
- Les modules, docs, tests, leases, replay et handoff sont correctement extraits.
- La tranche 2 peut débiter.
- Elle ne peut pas conserver seulement `locator + turnId`, car Retry/Edit/Continue réutilisent le même tour.
- Il faut ajouter `sourceId`.
- `continueResponse` exige un snapshot baseline capturé avant la nouvelle continuation.
- Le lifecycle lazy doit être factorisé plutôt que copié.
- Le Prompt normal reconnecté ne doit pas devenir un viewer read-only :
- compositeur normal après bootstrap ;
- saisie autorisée ;
- Send bloqué par le stream ;
- Stop normal existant.
- Le seul nouveau bouton est dans Activité, quand le panneau est fermé et la source active.
- La fin normale doit idéalement revenir directement à l'état normal sans reload terminal, sous preuve d'isomorphisme.

</ASSISTANT>

<USER>

pourquoi ce découpage en 4 ? c'est nécessaire ? ça complique énormément le workflow. en 1 fois ça ferait trop de modifs ? combien de fichiers environ ? en 2 fois ?

</USER>

<ASSISTANT>

I'm thinking...

</ASSISTANT>

<ASSISTANT>

  reading project:  [liveSession.programming-doc-readme.md](#),  e ["RunRef liveSession"](#),  e ["Modes Prompt liveSession"](#),  e ["Nettoyage panneau Prompt"](#),  e ["État Query Activité"](#),  e ["Stream normal frontend"](#),  t ["Fichiers sidebar activité"](#),  t ["Endpoints stream Orch"](#)

Réponse courte

Ta remarque est juste : **quatre étapes distinctes ne sont pas nécessaires**. J'avais décomposé la tranche 2 en quatre **sous-problèmes techniques**, mais je les ai présentés d'une manière qui ressemblait trop à quatre cycles complets spécification → implémentation → validation. Ce serait inutilement lourd.

Organisation	Faisabilité	Volume estimé	Mon avis
Une seule tranche / un seul gros commit	Oui	55–70 fichiers écrits, environ 65–85 fichiers dans le diff avec les générés	Faisable, mais revue et diagnostic difficiles
Deux étapes	Oui	30–40 fichiers par étape, avec quelques fichiers communs	Meilleur compromis
Quatre étapes	Oui	15–25 fichiers par étape	Trop de workflow pour ce besoin

Je retire donc la recommandation de quatre tranches.

Les quatre sujets restent utiles comme **ordre de travail interne**, pas comme quatre livraisons.

J'ai aussi relu [la programming doc liveSession](#) et l'inventaire réel des références actuelles avant de recalculer ces volumes.

Pourquoi j'avais découpé en quatre

Les quatre blocs correspondaient à des dépendances réelles :

1. identifier correctement chaque exécution de source ;
2. permettre au processus de survivre à la fermeture du panneau ;
3. rattacher un nouveau Prompt à cette source ;
4. représenter et contrôler le processus depuis Activité.

L'ordre est logique :

- on ne peut pas détacher proprement une source qu'on ne sait pas identifier ;
 - on ne peut pas rouvrir une source qui n'a pas continué ;
 - on ne peut pas ajouter le bouton Stop d'Activité sans disposer d'une identité exacte de la source à arrêter.
- Mais rien n'oblige à faire quatre spécifications ou quatre commits. Ces dépendances peuvent être traitées séquentiellement à **l'intérieur d'une seule implémentation**.

Le vrai découpage est plutôt :

Nature	Sujet
Fondation interne	Identité de source, lifecycle, baseline et replay
Fonctionnalité utilisateur	Fermeture, réouverture, Prompt normal, Activité et Stop
Cela conduit naturellement à deux étapes , pas quatre.	

En une seule fois

Estimation

En incluant les tests mais en excluant les sorties générées :

- environ **55 à 70 fichiers créés ou modifiés** ;
 - probablement **10 à 18 nouveaux fichiers** ;
 - environ **40 à 55 fichiers existants modifiés**.
- Après régénération des contrats Slimbk et clients :
- environ **65 à 85 fichiers dans le commit final**.

Le commit de tranche 1 en a touché **99**, donc le volume brut n'est pas hors de portée. La tranche 2 pourrait même toucher moins de fichiers que la tranche 1.

Répartition approximative

Domaine	Fichiers écrits ou modifiés
Identité exacte et contrats <code>liveSession</code>	15–22
Lifecycle de source et intégration du stream normal	8–12
Fermeture du panneau et cleanup différé	5–8
SidebarStore, Activité et Stop détaché	10–14
Prompt frontend et rattachement normal	12–18
Documentation et tests transversaux supplémentaires	3–6
Total unique après chevauchements	55–70
Sorties générées	10–15 environ

Les totaux des lignes ne doivent pas être additionnés directement : plusieurs fichiers appartiennent à deux catégories.

Est-ce trop ?

Non, pas techniquement.

Une implémentation unique est possible si la spécification décrit précisément :

- l'identité de source ;
- le traitement de `continueResponse`, Retry et Edit ;
- le cycle de vie ouvert/fermé du panneau ;
- le cleanup mémoire ;
- le rattachement frontend ;
- le modèle SidebarStore ;
- le Stop exact ;
- toutes les courses à tester.

Le problème d'une seule fois n'est pas le nombre de fichiers. C'est que le même diff modifierait simultanément :

- la source normale de toutes les requêtes ;
- la politique de fermeture des Prompt ;
- l'état de la sidebar ;
- le bootstrap frontend ;
- le contrôle Stop ;
- les invariants d'hydratation.

Si un scénario échoue, il devient plus difficile de déterminer si le défaut vient :

- du buffer source ;
- du cleanup ;
- de l'ouverture ;
- du frontend ;

- de l'activité ;
- ou de l'identité du run.

Avantages d'une seule fois

- workflow minimal ;
- pas de tranche intermédiaire ;
- aucune infrastructure temporairement inutilisée ;
- comportement complet livré en une seule fois ;
- une seule revue fonctionnelle globale.

Inconvénients

- gros diff très couplé ;
- revue plus fatigante ;
- diagnostic des courses plus difficile ;
- un défaut tardif peut obliger à reprendre plusieurs couches ;
- plus difficile de prouver séparément que la capture source est correcte avant d'autoriser la fermeture du panneau.

Verdict pour une seule fois

C'est acceptable si tu privilégies fortement la simplicité du workflow.

Je ne considérerais pas cette approche comme irresponsable, surtout après la qualité de la tranche 1.

Il faudra simplement traiter l'implémentation comme un grand chantier de 55–70 fichiers avec inventaire, lectures ciblées, tests par sous-système et validation globale.

En deux fois

C'est ma recommandation finale : **une seule spécification globale de tranche 2, mais deux passes d'implémentation cohérentes.**

Cela évite quatre cycles de workflow tout en conservant un vrai point de contrôle architectural.

Étape 1 — Préparer les sources normales, sans changer l'UX

Objectif

Rendre `liveSession` réellement capable de suivre les sources Prompt normales, mais conserver temporairement le comportement actuel de fermeture :

- fermer un panneau continue encore à arrêter la requête ;
- Activité ne change pas ;
- aucun nouveau bouton ;
- aucune reconnexion utilisateur activée.

Contenu

1. Ajouter l'identité d'instance de source distincte du `turnId`.
2. Étendre les contrats partagés.
3. Adapter le registre, Open, Replay et les tests existants.
4. Extraire le lifecycle lazy hors d'EasyAgents dans le module générique.
5. Rebrancher EasyAgents sur ce lifecycle sans changement fonctionnel.
6. Enregistrer et bufferiser les sources Prompt normales.
7. Ajouter les stratégies de baseline :
 - nouveau tour ;
 - Retry/Edit ;
 - continuation d'assistant.
8. Capturer correctement la baseline de `continueResponse`.
9. Ajouter les tests d'isomorphisme backend et de sources successives.
10. Laisser la fonctionnalité de détachement désactivée.

Volume

- environ **30 à 40 fichiers écrits ou modifiés** ;
- environ **5 à 10 sorties générées** ;
- diff total probable : **35 à 50 fichiers**.

Pourquoi ce nombre est déjà important : `LiveSessionRunRef` est actuellement utilisé dans environ vingt fichiers vérifiés. L'ajout d'une identité de source se propage donc naturellement dans :

- les contrats ;
- le registre ;
- Open ;
- Replay ;
- les endpoints ;

- les handlers ;
- EasyAgents ;
- le frontend ;
- et tous leurs tests.

État du produit après cette étape

Le comportement visible reste inchangé.

C'est important : l'utilisateur ne se retrouve pas avec un panneau qui continue en arrière-plan sans moyen de le rouvrir ou de l'arrêter. La nouvelle capacité existe en interne, mais la fermeture destructive reste active jusqu'à l'étape 2.

Étape 2 — Activer la fermeture persistante et l'UX complète

Objectif

Activer atomiquement toute la fonctionnalité utilisateur :

- fermeture sans kill ;
- conservation dans Activité ;
- troisième ligne Stop ;
- réouverture ;
- replay ;
- retour rapide au Prompt normal ;
- cleanup terminal correct.

Contenu

1. Séparer le cleanup du panneau de l'abort Orch.
2. Introduire la coordination fin de source / réouverture / clear mémoire.
3. Faire évoluer le SidebarStore de « fenêtres » vers « activités de session ».
4. Ajouter `panelState: open | closed`.
5. Conserver l'identité exacte active :
 - `locator` ;
 - `turnId` ;
 - identité de source.
6. Ajouter le Stop détaché exact.
7. Ajouter la troisième ligne dans Activité.
8. Faire décider l'ouverture live/classique côté extension.
9. Ajouter l'adaptateur frontend pour les sessions normales.
10. Réutiliser le compositeur Prompt normal :
 - saisie autorisée ;
 - Send bloqué pendant le stream ;
 - Stop normal existant.
11. Gérer l'ouverture avant la baseline.
12. Valider l'isomorphisme terminal.
13. Ajouter les tests de toutes les courses panneau/source/sidebar.

Volume

- environ **30 à 40 fichiers écrits ou modifiés** ;
- environ **5 à 10 sorties générées** ;
- diff total probable : **35 à 50 fichiers**.

Certains fichiers de l'étape 1 seront retouchés, notamment :

- [LiveSessionService.ts](#)
- [LiveSessionRegistry.service.ts](#)
- [usePromptLiveSession.hook.ts](#)
- [Prompt.tsx](#)

Les deux étapes ne représentent donc pas 70–80 fichiers uniques. Le total unique reste autour de **55 à 70 fichiers écrits**, comme dans l'option en une seule fois.

Pourquoi le découpage en deux est propre

La séparation ne suit pas arbitrairement « backend puis frontend ». Ce serait dangereux, car on pourrait livrer un backend actif avec une UX incomplète.

Elle suit la frontière :

Étape	État visible
1 — Capacité interne	Aucun changement fonctionnel

Étape

État visible

2 — Activation complète

Toute l'UX arrive ensemble

Cela garantit qu'il n'existe jamais de version où :

- fermer le panneau ne tue plus le processus ;
- mais le processus ne peut ni être rouvert ni arrêté.

C'est la principale raison pour laquelle ce découpage est meilleur qu'un découpage naïf backend/frontend.

Comparaison finale

Critère	Une fois	Deux fois
Cycles de workflow	1	2
Fichiers par revue	55–70	30–40
Risque d'état intermédiaire incomplet	Aucun	Aucun avec le découpage proposé
Facilité de revue	Moyenne à faible	Bonne
Localisation d'un bug	Difficile	Nettement meilleure
Risque de duplication temporaire	Aucun	Aucun
Validation du cœur avant activation	Non séparée	Oui
Temps total de conception	Légèrement plus faible	Légèrement plus élevé
Sécurité	Correcte avec énormément d'attention	Meilleure

Recommandation

Deux étapes sont le meilleur compromis.

Mais je ne recommande pas :

- quatre spécifications ;
- quatre implémentations ;
- quatre validations fonctionnelles complètes.

Je recommande :

1. **une spécification globale de la tranche 2**, qui décrit le résultat final et les deux étapes ;
2. **une première implémentation interne sans changement visible**, environ 30–40 fichiers ;
3. **une seconde implémentation d'activation complète**, environ 30–40 fichiers.

Si tu veux absolument réduire le workflow à un seul cycle, **une implémentation unique de 55–70 fichiers écrits reste raisonnable**. Le commit de tranche 1 prouve que le projet et la méthode supportent déjà un changement encore plus large.

Le choix entre une et deux fois est donc un arbitrage de revue et de maîtrise du risque, pas une impossibilité technique.

</ASSISTANT>

<USER>

je préfère en une seule fois, tu te sens capable ? le risque en 2 fois c'est de rater le couplage, ça n'est pas sans risque , tu comprends ?

</USER>

<ASSISTANT>

Oui, Joel, je m'en sens capable et je suis d'accord. 

</ASSISTANT>

<USER>

tu es prêt à spécifier ? tes choix me conviennent. il reste des points obscurs pour cette tranche finale ?

</USER>

<ASSISTANT>

I'm thinking...

</ASSISTANT>

<ASSISTANT>

Oui, je suis prêt à spécifier 

Je suis prêt à produire **une spécification unique et complète pour toute la tranche finale**, conçue pour être implémentée **en une seule fois**.

Je comprends et partage ton raisonnement : couper artificiellement le chantier créerait aussi un risque de régression, notamment en ratant le couplage entre :

- l'identité et l'enregistrement de la source ;

- la fermeture du panneau ;
- la projection dans Activité ;
- la réouverture du Prompt ;
- le replay du stream ;
- le retour à l'état normal ;
- l'arrêt depuis le Prompt ou depuis Activité ;
- la finalisation et le nettoyage de la mémoire Orch.

Une implémentation unique permet de définir et tester **la machine d'état complète**, sans introduire de contrat intermédiaire temporaire.

Décisions déjà suffisamment établies

Sujet	Décision retenue
Livraison	Une seule tranche finale, une seule implémentation cohérente.
Identité	Locator + <code>turnId</code> logique + <code>sourceId</code> d'exécution.
Types de requêtes	Prompt, SPECIF action, Compact Context, Retry, Edit last prompt et Continue response.
Continue response	Baseline immuable capturée avant la continuation, pour éviter toute duplication.
Lifecycle source	Factorisé dans le module générique, partagé par EasyAgents et les sources normales.
Panneau	Deux états seulement : ouvert ou fermé .
Fermeture active	La source continue ; aucun Abort/AwaitStop/ClearMemory prématuré.
Réouverture	L'extension choisit atomiquement entre ouverture live et ouverture classique.
Ouverture avant baseline	Le panneau attend la baseline au lieu d'échouer définitivement.
Prompt reconnecté	Interface normale rétablie dès que le snapshot et le replay sont correctement armés.
Compositeur	Visible et éditabile pendant la réponse ; seul l'envoi reste bloqué par le verrou normal du stream.
Stop dans le Prompt	Réutilisation du bouton Stop existant ; aucun second bouton.
Stop dans Activité	Nouveau bouton uniquement quand le panneau est fermé et la source active.
Bouton Activité	Icône Stop, taille small, variante danger, désactivation immédiate après clic.
Stop exact	Validation locator + turn + source ; aucun ancien clic ne peut arrêter une nouvelle source.
SidebarStore	Devient une projection d'activité de session, et non plus uniquement de fenêtre ouverte.
Activité et son	Restent possédés uniquement par le stream source ; le replay ne les redémarre pas.
Handoff normal	Retour direct à l'état normal si les tests d'isomorphisme le valident ; réconciliation terminale exacte seulement si indispensable.
Interact gates	Doivent survivre à la fermeture et être rejouées lors de la réouverture.
Tests	Tout comportement créé ou modifié, toutes les courses, tous les types de requête et l'isomorphisme terminal.
Documentation	Mise à jour de la programming doc <code>liveSession</code> et des sections d'intégration concernées.
Volume	Environ 55 à 70 fichiers écrits , plus les sorties générées normales.

Les difficultés techniques restantes sont **comprises**, mais ne demandent pas de nouvelle décision utilisateur :

- coordination réouverture contre cleanup mémoire ;
- races avant `fallback_persisted` ;
- sources successives partageant le même `turnId` ;
- baseline spéciale de `continueResponse` ;
- préservation des brouillons et attachments ;
- arrêt depuis Activité sans lease de panneau ;
- gate interactive pendant le détachement ;

- drainage Queue A / Queue B ;
- différence entre source terminale, replay drainé et état React invariant.

Un seul point UX reste réellement ambigu

Panneau fermé, puis processus terminé

Mes analyses précédentes ont donné deux conclusions différentes sur le devenir de l'item Activité. Il faut donc figer ce point avant la spécification.

Option recommandée selon ta demande initiale

L'item **reste dans Activité avec ses deux lignes normales** :

- titre ;
- durée de la dernière requête ;
- aucune troisième ligne ;
- aucun bouton Stop ;
- clic ouvrant désormais la session classique terminée.

Cette lecture suit directement ta phrase initiale : lorsque le processus termine, **le bouton et la troisième ligne disparaissent**. Tu n'avais pas demandé la suppression de l'item.

Autre possibilité

L'item est entièrement retiré lorsque :

- le panneau est fermé ;
- et la source n'est plus active.

La session reste alors accessible uniquement depuis la liste normale des chats.

Ma recommandation

Conserver l'item à deux lignes.

C'est le comportement le plus fidèle à ta demande initiale et il évite une disparition soudaine de l'item au moment où le travail finit. Une fois ce choix confirmé, il ne reste plus aucun point obscur bloquant et je peux produire immédiatement la spécification complète. 🚀



Garder l'item à deux lignes puis spécifier



Retirer l'item terminé puis spécifier

</ASSISTANT>

<USER>

si le processus est terminé et que la fenetre est fermée, aucune raison de garder l'item dans sidebar activités. Il reste si la fenetre est ouverte ou la fenetre est fermée et il est en exécution. Tu as compris ? C'est le seul point ? prêt à spécifier ?

</USER>

<ASSISTANT>

Oui, Joel. J'ai compris : c'était le seul point restant, et je suis prêt à spécifier la tranche finale en une seule fois. ✅

</ASSISTANT>

<USER>

Ecris la spécification

</USER>

<ASSISTANT>

I'm thinking...

</ASSISTANT>

<ASSISTANT>



✅ reading project: 🟡 e "[Identité liveSession](#)", 🟡 e "[État Activité Query](#)", 🟡 e "[État windows](#)

[Query](#)", 🟡 e "[Cleanup fermeture Prompt](#)", 🟡 e "[Stream Prompt normal](#)", 🟡 h "[Ouverture depuis](#)

[Activité](#)", 🟡 h "[Lifecycle source normal](#)", 🟡 t "[Module liveSession complet](#)", 🟡 t "[Frontend liveSession](#)

[complet](#)", 🟡 t "[Sidebar extension complète](#)", 📄 [LiveSessionRun.type.ts](#) [L1-L120], 📄 [LiveSessionPanel.type.ts](#) [L1-

[LiveSessionOpenResult.type.ts](#) [L1-L180], 📄 [LiveSessionRecord.type.ts](#) [L1-

[LiveSessionService.ts](#) [L1-L260], 📄 [LiveSessionOpen.service.ts](#) [L1-

[LiveSessionReplay.service.ts](#) [L1-L180], 📄 [PromptLiveSession.types.ts](#) [L1-

[AiStreamStartArgs.type.ts](#) [L1-L100], 📄 [SidebarUiMessages.ts](#) [L1-L160], 📄 [stream.endpoint.ts](#) [L1-

[iaStreamHandlers.ts](#) [L1-L260], 📄 [SidebarQueryRunningBridge.ts](#) [L1-L280], 📄 [OrchAsk.ts](#) [L1-

[OrchAskStream.ts](#) [L1-L240], 📄 [useAiStream.hook.ts](#) [L500-

[L714](#)], 📄 [usePromptRunCommands.hook.ts](#) [L300-L660], 📄 [preparePromptStreamingTurn.util.ts](#) [L1-

[L260](#)], 🟡 e "[Appels endpoint stream](#)", 📄 [Prompt.tsx](#) [L110-L260], 📄 [EasyAgentsBatchLiveSession.adapter.ts](#) [L1-

[L300](#)], 📄 [usePromptLiveSession.hook.ts](#) [L1-L420], 📄 [usePromptRunController.hook.ts](#) [L460-

L560],  [Prompt.tsx](#) [L480-L720],  [PromptInput.tsx](#) [L2020-L2168],  e ["Viewer mode PromptInput"](#),  [PromptInput.tsx](#) [L330-L430],  [SidebarStore.ts](#) [L100-L240],  [SidebarStore.queryWindow.listeners.manager.ts](#) [L1-L330],  [SidebarStore.queryWindow.mutations.manager.ts](#) [L1-L180],  [SidebarStore.rehydrate.manager.ts](#) [L1-L240],  [SidebarActivityQuery.tsx](#) [L1-L260],  [SidebarActivityQuery.test.tsx](#) [L1-L260],  [SidebarStore.handlers.manager.ts](#) [L1-L180],  [SidebarStore.chatTitles.manager.ts](#) [L1-L140],  [openPanel.util.ts](#) [L1-L230],  [openPanel.util.ts](#) [L230-L500],  [openPanel.util.ts](#) [L500-L740],  [PromptThreadCleanup.manager.ts](#) [L1-L240],  [uiAiPromptHandlers.ts](#) [L100-L220],  [AiPromptUiMessages.ts](#) [L1-L140],  [OpenedPanelsRegistry.util.ts](#) [L1-L180],  [closePromptPanel.util.ts](#) [L1-L160],  [LiveSessionRegistry.service.ts](#) [L1-L260],  [LiveSessionRegistry.service.ts](#) [L261-L520],  [LiveSessionRegistry.service.ts](#) [L521-L700],  [PromptQueryDoneSound.manager.ts](#) [L1-L260],  [PromptQueryDoneSound.manager.ts](#) [L260-L320],  e ["Usage interactionsEnabled"](#),  e ["Garde interactions history"](#),  e ["Utilisations interactions history"](#),  [PromptHistory.tsx](#) [L230-L330],  e ["Prop interactions bubble"](#),  e ["Variable interactionsEnabled"](#),  [PromptHistory.tsx](#) [L1030-L1176],  e ["Draft mode normal"](#),  [Prompt.tsx](#) [L930-L1010],  e ["Usage isPromptViewer"](#),  e ["Nettoyage activeStream"](#),  [useAiStream.hook.ts](#) [L440-L505],  e ["Réfs setQueryRunning"](#),  e ["Réfs addQueryWindow"](#),  e ["Réfs removeQueryWindow"](#),  e ["Réfs elapsed live"](#),  e ["Réfs elapsed last"](#),  e ["Réfs titre Query"](#),  e ["Imports manager window"](#),  [EasyAgentsBatchLiveSessionAdapter.service.ts](#) [L1-L240],  e ["Construction runRef Batch"](#),  ["RunRef EasyAgents exact"](#),  [AiQuery.type.ts](#) [L1-L220],  [MessageRegistry.ts](#) [L1-L100],  [SerialTaskQueue.ts](#) [L1-L220],  e ["Réfs rehydrate Query"](#),  [SidebarStore.core.manager.ts](#) [L1-L190],  l ["Tests handlers IA"](#),  l ["Tests panneau Prompt"](#),  e ["Initialisations windows Query"](#),  e ["Fixtures Query windows"](#),  h ["Fixtures Sidebar Query"](#),  g ["Tests abort query"](#),  e ["Tests abort useAiStream"](#),  [abort.endpoint.ts](#) [L1-L140],  g ["Tests commandes Prompt"](#),  [uiGenericHandlers.ts](#) [L1-L80],  e ["Appels openCentralPanel"](#),  g ["Tests handler UI générique"](#),  [NewChat.command.ts](#) [L1-L100],  [registerDeepLinkUriHandler.ts](#) [L100-L165],  [stream.endpoint.test.ts](#) [L1-L260],  e ["Construction ChildExecution"](#),  [uiGenericHandlers.openExternalUrl.test.ts](#) [L1-L80]

  specifying

"SPECIF_FINALISER_LIVESESSION_POUR_LES_SESSIONS_PROMPT_NORMALES_20260711_232041"

SPECIF_FINALISER_LIVESESSION_POUR_LES_SESSIONS_PROMPT_NORMALES_20260711_232041

Copy

Refine

Code

Verify

Markdown formattedMarkdown source

Spécification — Finaliser liveSession pour les sessions Prompt normales

Contexte

La tranche 1 a été entièrement implémentée dans le commit f65adc8e5172cc1e4f9781bf54ffa6dd089d3d33. Elle a extrait le live watcher EasyAgents Batch dans un module générique liveSession réparti entre les contrats shared, le backend Orch et le frontend Prompt. Cette fondation fournit déjà :

- un registre process-local de runs exacts ;
- un buffer ordonné complet de AiStreamedChunk ;
- des leases de viewer exactes ;
- les barrières distinctes sourceSettled et handoffReady ;
- l'ouverture live-safe sans reset de la mémoire Orch ;
- le replay backlog + tail ;
- la protection anti-ABA par époque de locator ;
- le terminal exact et la distinction TOOL_USE / vraie fin de tour ;
- les transports Slimbk et message bus génériques ;
- un hook frontend générique réutilisant Queue A, Queue B, les locks et les sanitizers Prompt ;
- un adaptateur EasyAgents qui conserve le comportement produit historique.

Cette tranche finale doit activer ce mécanisme pour les sessions Prompt AICode normales, en une seule implémentation cohérente. Le couplage entre capture de la source, fermeture du panneau, projection Activité, réouverture, replay, contrôle Stop et nettoyage terminal doit être traité dans le même changement afin qu'aucun contrat intermédiaire incomplet ne soit livré.

Le problème utilisateur à résoudre est le suivant : un renderer Prompt peut devenir gris, notamment après une saturation mémoire Electron. Aujourd'hui, fermer ce panneau déclenche explicitement `Abort` → `AwaitStop` → `ClearMemory`, détruit la génération en cours et oblige à relancer le dernier message depuis zéro. Après cette tranche :

- fermer un panneau Prompt normal pendant une génération ne doit plus arrêter cette génération ;
- la génération continue dans l'extension ;
- la session reste visible dans Activité pendant son exécution ;
- cliquer la session dans Activité ou dans la liste des chats doit rouvrir le Prompt sur la source exacte encore active ;
- le nouveau panneau reconstruit l'historique et rattrape le stream sans perte ni duplication ;
- si le panneau reste fermé jusqu'à la fin du processus, l'item disparaît d'Activité ;
- si le panneau est ouvert à la fin, l'item reste en état Idle comme aujourd'hui ;
- un bouton Stop apparaît sur une troisième ligne d'Activité uniquement quand le panneau est fermé et le processus encore actif.

La tranche doit préserver les invariants suivants :

- aucune modification des algorithmes `Queue A`, `Queue B`, `PromptStreamLockStore` ou des sanitizers ;
- aucune duplication du timer Activité ou du son de fin lors d'un replay ;
- aucun second bouton Stop dans le Prompt : le panneau reconnecté réutilise le bouton Stop normal du compositeur ;
- aucune notion de panneau « invisible » : le panneau est soit `open`, soit `closed` ;
- le compositeur normal doit être restauré dès que le snapshot de base est installé et que le replay est armé ; pendant la réponse, seul le verrou `Send` normal reste actif, la saisie du prochain brouillon et la préparation des attachments restent autorisées ;
- aucun `*.generated.*` ne doit être édité manuellement ; les fichiers générés seront régénérés par la compilation normale à partir des endpoints et contrats sources ;
- aucun fichier sous le répertoire documentaire personnel interdit ne doit être chargé ou modifié.

Une extension d'identité est obligatoire. La tranche 1 identifie un run par `locator` + `turnId`. Cela suffit pour EasyAgents, qui crée un nouveau `turnId` par source. Les sessions normales réutilisent volontairement un même `turnId` pour `retryQuery`, `editLastPrompt`, les retries SPECIF et `continueResponse`. Le run exact doit donc devenir :

- `locator` complet ;
- `turnId` logique et persistant ;
- `sourceId` unique pour une invocation précise du stream.

Le `sourceId` est généré une fois par `useAiStream.start()` pour chaque `promptQuery`, transporté au handler extension et à l'endpoint Orch, conservé dans Activité, puis utilisé par Open, Replay, Stop et les leases. Pour EasyAgents, l'adaptateur utilise explicitement `sourceId = turnId`, car son contrat produit garantit une source unique par tour.

Objectifs décrits par l'utilisateur

Besoin initial exprimé par l'utilisateur

- Livrer toute la tranche finale en une seule implémentation, et non en plusieurs tranches fonctionnelles susceptibles de rater le couplage entre backend, panneau, frontend et Activité.
- Permettre à une génération Prompt normale de continuer lorsque son panneau est fermé.
- Conserver l'item dans Activité si le panneau est ouvert, ou si le panneau est fermé mais que le processus est encore en exécution.
- Supprimer l'item d'Activité si le panneau est fermé et que le processus est terminé.
- Ajouter une troisième ligne avec un bouton Stop uniquement lorsque le panneau est fermé et que le processus est actif.
- Utiliser pour ce bouton l'icône Stop existante, la taille `small`, la variante `danger`, et le désactiver immédiatement après clic.
- Faire disparaître la troisième ligne et le bouton dès que le panneau est rouvert ou que le processus se termine.
- Réutiliser dans le panneau reconnecté le bouton Stop normal déjà existant ; n'ajouter aucun second bouton Stop dans le Prompt.
- Restaurer l'état normal du Prompt au plus tôt : aucun read-only prolongé propre au mécanisme de reconnexion ; seule la courte phase nécessaire à l'installation du snapshot et à l'armement du replay peut masquer ou verrouiller le compositeur.
- Une fois le replay armé, afficher le compositeur normal, autoriser la saisie et les attachments du prochain message, mais conserver le verrou `Send` normal tant que le stream actuel est actif.
- Étendre `SidebarStore` pour représenter des activités de session et pas uniquement des fenêtres ouvertes.
- Conserver le buffer raw-live complet, sans compression ni limitation ajoutée.
- Tester tout code créé, déplacé ou modifié, y compris toutes les courses de fermeture, réouverture, arrêt et succession de sources.
- Mettre à jour la programming doc générique `liveSession` et la documentation d'isomorphisme streaming/réhydratation.

- Respecter l'architecture modulaire existante : backend OOP composé, frontend en hooks/adaptateurs/composants unitaires, contrats stricts shared, commentaires techniques en anglais, aucun `any`, aucun cast inutile, aucun `eslint-disable`, aucun TODO et aucune importation dynamique.

Clarifications implicites ajoutées par AICode

- Le changement part exactement du commit `f65adc8e5172cc1e4f9781bf54ffa6dd089d3d33` et conserve intégralement le comportement EasyAgents visible.
- Le `sourceId` est obligatoire dans `LiveSessionRunRef` et dans toutes les routes/query strings `liveSession` ; aucune opération ne retombe sur le run actif d'un locator lorsqu'une identité exacte échoue.
- Le registre ajoute des réservations de source pour les streams normaux entre l'entrée du handler extension et le retour réussi de `askStream(...)`. Une réservation n'est pas un record replayable et ne viole pas l'invariant « seul `registerRun` crée un record ».
- Une réservation protège l'ouverture avant baseline, l'arrêt précoce et l'identité Activité. Elle est transformée atomiquement en record après le retour de `askStream(...)`, ou annulée/retirée si la source ne démarre pas.
- Le registre conserve pour les runs terminaux évincés une métadonnée de retraite exacte avec l'époque locator terminale. Un fallback terminal persistant n'est accepté que si aucune autre source du locator n'a fait évoluer cette époque ; un ancien `sourceId` ne peut jamais lire le résultat d'un Retry/Continue ultérieur du même `turnId`.
- Le cœur prend en charge deux readiness policies : `baselineAndObservableChunk` pour EasyAgents, inchangée, et `baselineOnly` pour les sessions normales afin qu'elles puissent être rouvertes dès la baseline durable, avant le premier token visible.
- Le cœur transporte un `replayMode` strict : `replaceTurnAssistants` pour `prompt/retry/edit/specif/compact-context` et `continueAssistant` pour `continueResponse`.
- `continueResponse` capture un snapshot baseline immuable avant `askStream(...)`. Ce snapshot conserve l'assistant existant ; le replay ajoute seulement la continuation. Les autres types de requête utilisent la baseline `fallback_persisted` et continuent de retirer/reconstruire les assistants du tour exact.
- Le lifecycle du `Readable` lazy est extrait du child executor EasyAgents dans un service générique réutilisé par EasyAgents et par l'endpoint de stream normal. Il conserve l'ordre : retour `askStream` → première lecture pendante → source-start synchrone → traitement du premier chunk → tail → fermeture/abort/settlement exact.
- Le stream normal est enregistré avec ownership `source`. Le stream de replay n'ouvre jamais une seconde activité et ne joue jamais un second son.
- Le handler source normal reste l'unique propriétaire du timer Activité et du son de fin.
- L'ouverture de tout panneau Prompt passe par un coordinateur extension commun, y compris les ouvertures depuis la liste des chats, Activité, les deep links et la commande New Chat. Le coordinateur choisit atomiquement entre ouverture classique et query `liveSession` exacte.
- Les opérations « ouverture de panneau », « dispose du panneau » et « nettoyage après fin d'une source détachée » sont sérialisées par locator afin d'empêcher la course clear-memory / réouverture.
- Si le nettoyage terminal gagne la course, l'ouverture suivante est classique et réhydrate la mémoire ; si l'ouverture gagne, le nettoyage voit le panneau ouvert et ne clear pas la mémoire. Aucun Prompt déclaré normal ne peut rester attaché à une mémoire déjà effacée.
- Le panneau reconnecté utilise une policy frontend `normalPrompt`, distincte de la présentation viewer EasyAgents. L'état `liveSessionStreaming` n'implique plus automatiquement une présentation read-only.
- Le hook frontend traite `not_ready` différemment selon l'adaptateur : EasyAgents conserve son échec actuel ; le Prompt normal réessaie tant que la source/réservation exacte existe.
- Pour le Prompt normal, le handoff terminal est direct : après vrai terminal et drainage des queues, le hook passe à `normal` sans second Open terminal et sans réhydratation destructive. Un test d'isomorphisme strict prouve que l'état final est identique au stream continu.
- Pour EasyAgents, le handoff terminal strict avec snapshot exact retourné par Open reste inchangé.
- Le Stop normal d'un `promptQuery` devient exact grâce au `sourceId` conservé dans l'état interne de `useAiStream` ; il utilise une nouvelle route source exacte. L'ancien endpoint thread-only reste réservé aux nettoyages destructifs de cycle de vie.
- Le Stop d'Activité ne possède pas de lease. Il valide synchroniquement : run exact actif, ownership `source`, panneau toujours fermé, source non settled, aucun Stop déjà accepté ; puis il réclame l'abort exact et appelle Orch sans `await` intermédiaire.
- SidebarStore utilise une union stricte : un item Idle implique un panneau ouvert ; un item Running transporte `activeTurnId`, `activeSourceId`, `stopRequested` et `panelState` : `open|closed`. Un item fermé et idle est impossible et doit être supprimé.

- Le champ `queryState.windows` et le type `SidebarActivityQueryWindowItem` sont renommés respectivement `queryState.items` et `SidebarActivityQueryItem`, car le contenu n'est plus un miroir de fenêtres.
- Aucun résultat généré n'est listé comme fichier à éditer. La génération normale devra produire les clients et contrats correspondants après modification des endpoints sources.

Vue d'ensemble de la méthode choisie

Identité et machine d'état

`LiveSessionRunRef` devient `{ locator, turnId, sourceId }`. Le run key structurel inclut les trois valeurs.

Le `turnId` continue d'identifier les messages et chunks ; le `sourceId` distingue plusieurs exécutions successives du même tour.

Le registre possède :

- des réservations exactes de sources normales ;
- des records enregistrés/replayables ;
- un index actif par locator ;
- des leases actives/retired ;
- une époque monotone par locator ;
- des métadonnées de retraite terminale ;
- un état `abortRequested` atomique ;
- la date de démarrage de la source pour la réhydratation Activité.

Une source normale suit :

1. création du `sourceId` frontend ;
2. message bus vers l'extension ;
3. réservation exacte ownership `source` ;
4. publication Activité Running avec état du panneau réel ;
5. capture éventuelle de la baseline `continueResponse` ;
6. retour réussi de `askStream(...)` ;
7. priming du `Readable` lazy ;
8. transformation réservation → record ;
9. append de chaque chunk avant son exposition au consommateur ;
10. baseline durable ;
11. replay possible ;
12. source settled ;
13. handoff immédiat pour une source normale ;
14. transition Activité Idle si panneau ouvert, ou suppression si panneau fermé ;
15. cleanup mémoire/attachments/trivial chat seulement si aucun panneau n'est ouvert.

Replay et baselines

`LiveSessionReplayMode` expose deux branches :

- `replaceTurnAssistants` : le snapshot live retire les assistants du tour exact ; le frontend recrée think + assistant et rejoue tous les chunks de cette source ;
- `continueAssistant` : le snapshot baseline immuable conserve l'assistant existant ; le frontend réutilise son UID hydraté, place un thinking de continuation avant lui et applique les nouveaux tokens sur son texte de base.

Le replay continue d'utiliser Queue A et Queue B sans modification algorithmique.

Cycle de vie du panneau

Toutes les ouvertures Prompt passent par le coordinateur. Le low-level `openCentralPanel` reste responsable de la création/reveal VS Code ; le coordinateur décide de la query à utiliser et sérialise avec les cleanups.

- panneau ouvert + source active : item Running à deux lignes ;
- panneau fermé + source active : item Running à trois lignes avec Stop ;
- panneau ouvert + source terminée : item Idle à deux lignes ;
- panneau fermé + source terminée : aucun item.

La fermeture d'un panneau actif détache le mapping cleanup, libère la lease si le panneau était un observer normal, conserve mémoire/son/activity, et diffère les nettoyages. La fermeture d'un panneau inactif conserve le cleanup historique complet.

Frontend Prompt normal reconnecté

Pendant `liveSessionOpening`, le panneau peut afficher un bref état de reconnexion. Dès que le snapshot est hydraté et le replay armé :

- l'adaptateur normal retourne `isViewerMode=false` ;

- le compositeur et les attachments sont visibles ;
- la saisie et la sauvegarde du draft sont actives ;
- `isStreaming` conserve le verrou Send et transforme le bouton normal en Stop ;
- aucune action Stop supplémentaire n'est rendue par le banner `liveSession` ;
- les interactions d'historique normales sont réactivées ;
- l'ownership de cleanup reste différé jusqu'au vrai handoff terminal.

À la fin du replay, la policy normale passe directement le runtime à `normal`, conserve le draft/attachments locaux et enregistre à nouveau le thread pour les futures fermetures. `EasyAgents` conserve son hydratation terminale stricte actuelle.

Sidebar Activité

Le bouton de la troisième ligne utilise `VscButton` avec :

- `icon="SquareStop"` ;
- `size="small"` ;
- `variant="danger"` ;
- label Stop ;
- `disableAfterClick` ;
- un `testId` stable basé sur le `sourceId` ;
- une garde store `stopRequested` pour synchroniser plusieurs vues Sidebar.

Le clic « titre » n'assemble plus une query Prompt côté frontend. Il envoie le locator à l'extension, qui résout la source active exacte et ouvre live ou classique.

Liste exacte des fichiers avec détails d'implémentation

Contrats shared

1) `AICode/src/sharedlib/src/types/liveSessionTypes/LiveSessionReplay.type.ts` — créé

- Définir `ZLiveSessionReplayMode` et `LiveSessionReplayMode` avec `replaceTurnAssistants | continueAssistant`.
- Documenter le contrat baseline/reconstruction de chaque branche.

2) `LiveSessionRun.type.ts` — modifié

- Ajouter `sourceId` strict, trim/non vide, obligatoire dans `ZLiveSessionRunRef`.
- Inclure `sourceId` dans `buildLiveSessionRunKey`.
- Conserver la clé locator inchangée.
- Documenter que `turnId` est logique/persistant et `sourceId` exécutif.

3) `LiveSessionPanel.type.ts` — modifié

- Ajouter `LIVE_SESSION_SOURCE_ID_PARAM`.
- Ajouter un paramètre d'adaptateur explicite `liveSessionAdapter` et son enum `easyAgentsBatch | normalPrompt`.
- Conserver les deux close policies ; activer réellement `continueOnPanelClose` pour le Prompt normal.
- Ajouter les schémas stricts source/adaptateur.

4) `LiveSessionOpenResult.type.ts` — modifié

- Ajouter `sourceId` aux identités live et terminal.
- Ajouter `replayMode` uniquement à la branche live.
- Étendre le transport `Slimbk` et ses `superRefine` pour exiger/interdire exactement les nouveaux champs selon la branche.

5) `LiveSessionContracts.test.ts` — modifié

- Couvrir la nouvelle clé locator+turn+source, les collisions, les champs transport, les adaptateurs, le nouveau paramètre query et les replay modes.
- Vérifier que deux sources du même tour ont des run keys différentes.

6) `SidebarStore.ts` — modifié

- Remplacer `SidebarActivityQueryWindowItem` par l'union stricte `SidebarActivityQueryItem`.
- Branches : Idle uniquement avec `panelState='open'` ; Running avec `panelState='open' | 'closed'`, `activeTurnId`, `activeSourceId`, `stopRequested`, `live elapsed`.
- Renommer `queryState.windows` en `queryState.items`.
- Préserver `title`, `last elapsed`, `updatedAt` et `openedAt`.

7) `SidebarUiMessages.ts` — modifié

- Ajouter `sidebar->ext:openQueryActivity` avec locator complet.
- Ajouter `sidebar->ext:stopDetachedQuery` avec `LiveSessionRunRef`, réponse `{ accepted: boolean }`.

8) `MessageRegistry.ts` — modifié

- Ajouter `sourceId` au canal `proxylocalapi:aiStream`.
- Ajouter `sourceId` au canal `proxylocalapi:liveSessionStream`.

Backend Orch liveSession

9) [liveSession.programming-doc-readme.md](#) — modifié

- Documenter sourceId, réservations, readiness `baselineOnly`, replay modes, baseline de continuation, source ownership normale, lifecycle partagé, panel open/closed, direct handoff normal, Stop source et Stop détaché.
- Ajouter la machine d'état complète et les races source/panel/cleanup.

10) [LiveSessionRecord.type.ts](#) — modifié

- Ajouter `baselineOnly` à `LiveSessionReadiness`.
- Ajouter `replayMode`, `baselineSession`, `sourceStartedAtMs`, `abortRequested` et `viewerGroup` optionnel.
- Ajouter le type interne de réservation source et la métadonnée de retraite terminale avec époque locator.
- Adapter permits/snapshots à l'identité avec `sourceId`.

11) [LiveSessionRegistry.service.ts](#) — modifié

- Ajouter réservation/cancel exacts pour ownership `source`.
- Transformer atomiquement une réservation en record lors de `registerRun`.
- Faire évoluer `refreshReplayReady` selon la readiness policy.
- Permettre de stocker une baseline snapshot immuable lors de `markBaselineReady`.
- Ajouter `getActiveSourceRun`, `listActiveSourceRuns`, `hasActiveOrReservedSource`.
- Ajouter claim d'abort exact/idempotent pour viewer et detached source, avec `abortRequested`.
- Conserver un tombstone terminal avec époque afin de protéger les fallbacks après éviction contre les sources ultérieures du même tour.
- Étendre l'époque locator aux transitions reservation/registration/cancel/handoff.
- Préserver toutes les garanties tranche 1.

12) [LiveSessionRegistry.service.test.ts](#) — modifié

- Ajouter sources successives partageant `turnId` mais pas `sourceId`.
- Couvrir réservation → record, cancellation, ouverture not-ready, baselineOnly, abort pré/post-registration, activité active list, tombstone terminal et époque après Retry/Continue.
- Rejouer tous les tests tranche 1 avec `sourceId` explicite.

13) [LiveSessionService.ts](#) — modifié

- Exposer `reserve/cancel source`, `active source resolution/listing`, baseline snapshot, `settlement+handoff` normal atomique et claims d'abort.
- Conserver `Open/Replay/release/sweep` existants.

14) [LiveSessionService.test.ts](#) — modifié

- Couvrir le parcours normal complet depuis réservation jusqu'au handoff, avec réouverture avant et après baseline.
- Couvrir même turn / sources différentes et abort exact.

15) [LiveSessionOpen.service.ts](#) — modifié

- Retourner `not_ready` pour une réservation exacte.
- Pour `replaceTurnAssistants`, conserver le filtrage existant.
- Pour `continueAssistant`, retourner le baseline snapshot immuable sans supprimer l'assistant cible.
- Retourner `sourceId` et `replayMode`.
- Valider les fallbacks terminaux par métadonnée de retraite + époque exacte.

16) [LiveSessionOpen.service.test.ts](#) — modifié

- Couvrir réservation, baselineOnly, continue snapshot, deux sources du même tour, fallback terminal avant/après source ultérieure, `sourceId mismatch` et toutes les courses existantes.

17) [LiveSessionReplay.service.test.ts](#) — modifié

- Ajouter `sourceId` à toutes les fixtures.
- Vérifier qu'une subscription d'une ancienne source du même turn ne reçoit jamais les chunks de la nouvelle.

18) [AICode/src/extension/src/aiTools/orch/liveSession/source/LiveSessionSourceLifecycle.service.ts](#) — créé

- Extraire le lifecycle générique du `Readable` lazy sous forme d'async generator/service OOP.
- Garantir `createReadable` sans `settlement` si `throw` pré-retour, premier `next()` armé, source-start sync, drainage, abort/return/destroy best-effort, observation de la promesse pendante, `settlement` exact et priorité des erreurs.

19) [AICode/src/extension/src/aiTools/orch/liveSession/source/__test__/LiveSessionSourceLifecycle.service.test.ts](#) — créé

- Migrer et généraliser les tests de priming/cleanup : source-start throw, première lecture throw, tail throw, callback consumer throw, abort, return/destroy, `settlement` throw et agrégation déterministe.

20) [AICode/src/extension/src/aiTools/orch/liveSession/source/NormalPromptLiveSessionSource.service.ts](#) — créé

- Mapper les `AiQuery` vers readiness et replayMode.

- Capturer la baseline immuable de `continueResponse` avant `askStream`.
- Enregistrer la source ownership `source` après retour de `askStream`.
- Parser chaque chunk, l'apprendre avant exposition, marquer fallback baseline, puis settlement+handoff normal.
- Respecter les abort requests pré-enregistrement.

21) [AICode/src/extension/src/aiTools/orch/liveSession/source/__test__/NormalPromptLiveSessionSource.service.test.ts](#) — **créé**

- Couvrir prompt/retry/edit/specif/compact/continue, baseline, ordre append-before-yield, même turn avec `sourceId` différent, abort précoce et handoff.

Adaptation EasyAgents

22) [EasyAgentsBatchLiveSessionAdapter.service.ts](#) — **modifié**

- Enregistrer `replayMode='replaceTurnAssistants'`.
- Utiliser l'invariant explicite `sourceId=turnId`.
- Injecter le lifecycle source générique dans le wiring EasyAgents.

23) [EasyAgentsBatchLiveSessionAdapter.service.test.ts](#) — **modifié**

- Vérifier `sourceId=turnId` et l'absence de modification du comportement EasyAgents.

24) [EasyAgentsBatchRunExecution.service.ts](#) — **modifié**

- Construire tous les run refs EasyAgents avec `sourceId=liveTurnId`.
- Conserver ligne/turn live-only inchangés.

25) [EasyAgentsBatchRunExecution.live-activity-and-gating.service.test.ts](#) — **modifié**

- Étendre les assertions d'identité exacte au `sourceId`.

26) [EasyAgentsBatchRunChildExecution.service.ts](#) — **modifié**

- Remplacer son lifecycle local par `LiveSessionSourceLifecycleService`.
- Conserver parsing Batch, callbacks métier et résultat discriminé.

27) [EasyAgentsBatchRunChildExecution.service.test.ts](#) — **modifié**

- Conserver des tests d'intégration du child executor sur le lifecycle partagé ; déplacer les tests purement génériques vers le nouveau service.

28) [EasyAgentsBatchRunManager.service.ts](#) — **modifié**

- Injecter le lifecycle partagé dans le child executor de production.
- Adapter la lecture du nouveau `queryState.items`.

29) [EasyAgentsBatchRunExecution.testHarness.ts](#) — **modifié**

- Injecter/mock le lifecycle partagé et fournir des run refs avec `sourceId`.

30) [EasyAgentsBatchRunLifecycle.service.test.ts](#) — **modifié**

- Adapter la construction du child executor au lifecycle partagé.

31) [EasyAgentsBatchLastViewerStop.service.test.ts](#) — **modifié**

- Ajouter `sourceId` aux fixtures et conserver les courses A/B et ancien/nouveau run.

32) [easyAgents.programming-doc-readme.md](#) — **modifié**

- Documenter `sourceId=turnId` et le lifecycle désormais partagé ; conserver la policy zéro viewer = hard stop.

Endpoints et message bus

33) [liveSession.open.endpoint.ts](#) — **modifié**

- Exiger `sourceId` et transporter `replayMode/sourceId`.

34) [liveSession.open.endpoint.test.ts](#) — **modifié**

- Couvrir les nouveaux champs et branches.

35) [liveSession.stream.endpoint.ts](#) — **modifié**

- Exiger `sourceId` et relayer le run exact complet.

36) [liveSession.stream.endpoint.test.ts](#) — **modifié**

- Couvrir `sourceId` exact et mismatch.

37) [liveSession.abort.endpoint.ts](#) — **modifié**

- Exiger `sourceId`.
- Autoriser le Stop d'un viewer exact pour ownership observer ou source, toujours avec lease active et claim atomique.

38) [liveSession.abort.endpoint.test.ts](#) — **modifié**

- Couvrir viewer EasyAgents, viewer Prompt normal, `sourceId` stale, double Stop et source remplacée.

39) [AICode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/abortSource.endpoint.ts](#) — **créé**

- Route exacte pour le bouton Stop normal du panneau source d'origine.
- Recevoir `locator+turnId+sourceId`, réclamer l'abort exact, puis appeler Orch sans await intermédiaire.

40) [AICode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/__test__/abortSource.endpoint.test.ts](#) — **créé**

- Couvrir source réservée/enregistrée, stale source, double clic, source absente et ordre claim → abort.

41) [query ask stream.endpoint.ts](#) — modifié

- Ajouter sourceId au schéma.
- Déléguer le lifecycle et l'instrumentation à `NormalPromptLiveSessionSourceService`.
- Conserver la persistance de préférence SPECIF et la conversion du premier meta error en header Slimbk.

42) [query ask stream.endpoint.test.ts](#) — modifié

- Tester la délégation, sourceId, ordre SPECIF, premier chunk/header et erreurs.

43) [liveSessionStreamHandlers.ts](#) — modifié

- Relayer sourceId.
- Conserver ownership source sans Activité/son et ownership observer avec ressources actuelles.

44) [liveSessionStreamHandlers.test.ts](#) — modifié

- Couvrir sourceId et absence de double activité/son pour source ownership.

45) [iaStreamHandlers.ts](#) — modifié

- Recevoir le sourceId généré frontend.
- Réserver le run avant l'appel endpoint.
- Démarrer l'activité avec run ref exact et panelState réel.
- Transmettre sourceId à l'endpoint.
- Appeler le coordinateur panneau après settlement.
- Annuler la réservation de façon idempotente sur échec pré-enregistrement.

46) [iaStreamHandlers.queryDoneSound.test.ts](#) — modifié

- Vérifier que la fermeture du panneau actif ne supprime pas le son avant terminal et qu'un replay ne joue rien.

47) [iaStreamHandlers.runIdPropagation.test.ts](#) — modifié

- Étendre les assertions au run ref locator+turn+source et au cleanup final.

Activité runtime extension

48) [SidebarQueryRunningBridge.ts](#) — modifié

- Renommer la classe interne en `SidebarQueryActivityRuntimeService` sans déplacer le fichier pour limiter le churn d'imports.
 - Stocker runRef exact, timer, startMs et runId.
 - Utiliser sourceId pour les guards onChunk/onStop.
 - Piloter les nouvelles transitions SidebarStore : source start, elapsed, source settle.
- #### 49) [SidebarQueryRunningBridge.race.test.ts](#) — modifié
- Couvrir même turn avec sources successives, panneau fermé/ouvert et stale stop.
- #### 50) [SidebarQueryRunningBridge.usage.test.ts](#) — modifié
- Couvrir usage exact par source et transition Idle/suppression selon panelState.

Coordination panneau Prompt

51) [AICode/src/extension/src/extension/panels/prompt/liveSession/PromptLiveSessionPanelQuery.util.ts](#) — créé

- Extraire parsing locator canonique, parsing du descripteur live complet et construction des query strings normalPrompt/EasyAgents.
- Générer une lease fraîche pour toute réouverture normale active.

52) [AICode/src/extension/src/extension/panels/prompt/liveSession/__test__/PromptLiveSessionPanelQuery.util.test.ts](#) — créé

- Couvrir locator strict, sourceId, adaptateur, ordre query, close policies et paramètres invalides.

53) [AICode/src/extension/src/extension/panels/prompt/liveSession/PromptLiveSessionPanelCoordinator.service.ts](#) — créé

- OOP composé avec queues FIFO par locator.
- `openPromptPanel`: sérialiser, révéler l'existant, choisir live/classique, puis appeler le low-level `openCentralPanel`.
- `onPanelDisposed`: sérialiser, détacher ou nettoyer selon source active.
- `onSourceSettled`: sérialiser, conserver si panneau ouvert ou nettoyer si fermé.
- Réévaluer l'état du panneau juste avant tout `ClearMemory`.

54) [AICode/src/extension/src/extension/panels/prompt/liveSession/__test__/PromptLiveSessionPanelCoordinator.service.test.ts](#) — créé

- Couvrir les deux ordres open/cleanup, close/source settle, source avant baseline, panneau reconnecté, clear gagné/perdu et absence d'état mixte.

55) [AICode/src/extension/src/extension/panels/prompt/liveSession/PromptLiveSessionDetachedStop.service.ts](#) — créé

- Valider panneau fermé, item exact Running, run ownership source et claim atomique.
- Appeler Orch sans await intermédiaire, puis marquer `stopRequested` dans le store.

56) [AICode/src/extension/src/extension/panels/prompt/liveSession/__test__/PromptLiveSessionDetachedStop.service.test.ts](#) — **créé**

- Couvrir accepted/refused, panneau rouvert, source remplacée, double clic et source pré-enregistrée.

57) [PromptThreadCleanup.manager.ts](#) — **modifié**

- Factoriser : détachement du mapping sans abort, cleanup destructif historique, cleanup après source déjà settled.
- Partager le cleanup orphan attachments.
- Ne jamais Abort/ClearMemory une source normale active lors du dispose.

58) [PromptThreadCleanup.manager.attachmentsCleanup.test.ts](#) — **modifié**

- Couvrir détachement actif, cleanup idle historique, cleanup terminal sans abort et sérialisation attachments.

59) [openPanel.util.ts](#) — **modifié**

- Utiliser le nouvel util de query.
- Marquer panelState open à create/reveal.
- Sur dispose : retirer registry, release exacte selon policy, marquer closed, puis déléguer au coordinateur.
- Retirer cleanup/trivial/sound/remove item inline devenus responsabilités du coordinateur.

60) [openPanel.util.test.ts](#) — **modifié**

- Couvrir sourceId, adaptateur, continueOnPanelClose release, délégation coordinateur et ordre registry fermé avant cleanup.

61) [uiGenericHandlers.ts](#) — **modifié**

- Pour htmlPage= 'prompt', appeler le coordinateur async ; conserver openCentralPanel pour les autres pages.

62) [uiGenericHandlers.openExternalUrl.test.ts](#) — **modifié**

- Mocker le coordinateur Prompt et ajouter une assertion d'ouverture Prompt coordonnée sans changer les tests URL.

63) [NewChat.command.ts](#) — **modifié**

- Ouvrir la nouvelle session via le coordinateur Prompt.

64) [NewChat.command.test.ts](#) — **modifié**

- Adapter le mock et vérifier l'appel coordonné.

65) [registerDeepLinkUriHandler.ts](#) — **modifié**

- Passer les deep links chat par le coordinateur afin qu'un locator actif soit rouvert live.

66) [registerDeepLinkUriHandler.scope.test.ts](#) — **modifié**

- Vérifier le nouveau chemin d'ouverture et préserver les tests de scope.

SidebarStore et Activité

67) [SidebarStore.queryWindow.listeners.manager.ts](#) — **modifié**

- Remplacer add/running génériques par transitions explicites panel opened, source started, source settled.
- Remplacer les objets atomiquement selon l'union SidebarActivityQueryItem.
- Conserver listeners title/running pour les tabs Prompt.

68) [SidebarStore.queryWindow.mutations.manager.ts](#) — **modifié**

- Renommer les accès windows en items.
- Ajouter panel closed, force remove, live elapsed exact, stopRequested exact.
- Conserver rename ephemeral.

69) [SidebarStore.core.manager.ts](#) — **modifié**

- Initialiser queryState.items.
- Rehydrater panneaux ouverts et sources ownership source actives/réservées.

70) [SidebarStore.rehydrate.manager.ts](#) — **modifié**

- Renommer la fonction en rehydrateQueryActivities.
- Fusionner panneaux ouverts et liveSession.listActiveSourceRuns() sans doublons.
- Calculer panelState depuis le registry de panneaux et elapsed depuis sourceStartedAtMs.

71) [SidebarStore.rehydrate.manager.test.ts](#) — **modifié**

- Couvrir panneau idle ouvert, source ouverte running, source fermée running, et aucune entrée closed+idle.

72) [SidebarStore.manager.queryRunningUpsert.test.ts](#) — **modifié**

- Tester la machine d'état complète et les identités source exactes.

73) [SidebarStore.chatTitles.manager.ts](#) — **modifié**

- Lire queryState.items et conserver le fallback title même pour un item fermé/running.

74) [SidebarStore.handlers.manager.ts](#) — **modifié**

- Enregistrer les handlers openQueryActivity et stopDetachedQuery.
- Déléguer au coordinateur et au service Stop.

75) [AICode/src/extension/src/extension/panels/sidebar/__test__/SidebarStore.handlers.manager.test.ts](#) — **créé**

- Vérifier payloads locator/runRef, retours accepted et délégation stricte.

76) [delete.endpoint.ts](#) — **modifié**

- Lire queryState.items ; continuer à bloquer toute suppression d'une source active ouverte ou fermée.
- Force-remove l'activité seulement après fermeture sûre.

- 77) [list.endpoint.windowsLock.eperm.test.ts](#) — **modifié**
- Adapter les fixtures `items`.
- 78) [diskAliases.lastWins.endpoints.test.ts](#) — **modifié**
- Adapter les fixtures `items` et les scénarios delete.
- 79) [ChatPersistenceWriter.windowsFallback.test.ts](#) — **modifié**
- Adapter la fixture `SidebarStore` au nouveau champ `items`.
- 80) [OrchMemory.naive.title.user.prompt.specif.multiline.e2e.repro.test.ts](#) — **modifié**
- Adapter la fixture `SidebarStore` au nouveau champ `items`.
- 81) [SidebarActivityQuery.tsx](#) — **modifié**
- Utiliser `queryState.items` et `SidebarActivityQueryItem`.
- Remplacer l'ouverture directe par `sidebar->ext:openQueryActivity`.
- Ajouter callback Stop via `sidebar->ext:stopDetachedQuery`.
- Rendre la troisième ligne uniquement pour `Running+closed`.
- Utiliser `VscButton SquareStop/small/danger/disableAfterClick` et `testId sourceId`.
- Définir les fonctions React modifiées avec `useCallback` et les projections avec `useMemo`.
- 82) [SidebarActivityQuery.test.tsx](#) — **modifié**
- Tester ouverture root/nested, troisième ligne conditionnelle, disparition open/terminal, payload exact, disable immédiat et stopRequested.
- Frontend liveSession et Prompt**
- 83) [AICode/src/frontend/src/app/prompt/liveSession/adapters/NormalPromptLiveSession.adapter.ts](#) — **créé**
- Parser adapter normal, `sourceId`, `lease` et `continueOnPanelClose`.
- Fournir behavior `retryNotReady`, `resumePromptState`, surface Prompt normale pendant streaming.
- Construire un view model non-viewer dès le replay armé ; aucune action Stop banner.
- 84) [AICode/src/frontend/src/app/prompt/liveSession/adapters/__test__/NormalPromptLiveSession.adapter.test.ts](#) — **créé**
- Couvrir parsing, `sourceId`, `policy`, comportements et view models opening/streaming/normal.
- 85) [AICode/src/frontend/src/app/prompt/liveSession/adapters/PromptLiveSessionBootstrap.adapter.ts](#) — **créé**
- Dispatcher explicite par `liveSessionAdapter` vers `EasyAgents` ou `normalPrompt`.
- Rejeter adapter absent/inconnu en mode `liveSession`.
- 86) [AICode/src/frontend/src/app/prompt/liveSession/adapters/__test__/PromptLiveSessionBootstrap.adapter.test.ts](#) — **créé**
- Couvrir dispatch, configurations partielles et zéro fallback implicite.
- 87) [PromptLiveSession.types.ts](#) — **modifié**
- Ajouter le behavior runtime : `not-ready policy`, `handoff policy`, surface interactive.
- Supprimer l'inférence `mode!=normal` ⇒ viewer.
- Étendre le bootstrap au run ref avec `sourceId`.
- 88) [usePromptLiveSession.hook.ts](#) — **modifié**
- Transporter `sourceId`.
- Mémoriser `replayMode` du résultat live.
- Retry contrôlé de `not_ready` pour `normalPrompt` ; comportement `EasyAgents` inchangé.
- Démarrer le replay avec `replayMode`.
- Pour `normalPrompt`, passer directement à normal après `terminal+drainage` sans `Open terminal`.
- Pour `EasyAgents`, conserver `Open terminal/hydratation stricte/Retry`.
- Nettoyer tous timers de retry à l'unmount.
- 89) [usePromptLiveSession.hydratation.test.tsx](#) — **modifié**
- Ajouter `sourceId/replayMode` et conserver tous les scénarios `EasyAgents`.
- 90) [usePromptLiveSession.handoff.test.tsx](#) — **modifié**
- Vérifier que `EasyAgents` conserve le `handoff terminal` exact et que `normalPrompt` n'appelle aucun `Open terminal`.
- 91) [AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.normalPrompt.test.tsx](#) — **créé**
- Couvrir `not-ready retry`, surface normale, `draft/attachments` préservés, `Stop normal`, `direct handoff`, `bootstrap terminal` et `close/reopen`.
- 92) [AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.normalPromptIsomorphism.test.tsx](#) — **créé**
- Comparer état final stream continu vs `baseline+replay` pour `prompt`, `retry`, `edit`, `specif`, `compact`, `continue`, `tool calls`, `thinking`, `gates`, `usage` et `file edits`.

93) [EasyAgentsBatchLiveSession.adapter.ts](#) — modifié

- Émettre sourceId=turnId et liveSessionAdapter=easyAgentsBatch dans la query.
- Parser la nouvelle identité sans modifier labels/UX.

94) [EasyAgentsBatchLiveSession.adapter.test.ts](#) — modifié

- Adapter ordre query, sourceId et dispatcher.

95) [AiStreamStartArgs.type.ts](#) — modifié

- Séparer les arguments publics de start et l'état actif interne enrichi.
- Conserver promptQuery caller sans sourceId explicite ; useAiStream.start le génère.
- Remplacer les champs live observer séparés par un runRef exact.

96) [useAiStream.hook.ts](#) — modifié

- Générer un sourceId par promptQuery accepté.
- Le transporter au stream handler et le conserver dans activeStream.
- Stop promptQuery via abortSource; Stop live observer via abort viewer exact.
- Conserver queues, sanitizers, raw-live et terminal handling inchangés.

97) [useAiStream.hook.liveSessionObserver.test.ts](#) — modifié

- Ajouter runRef sourceId et vérifier les deux routes Stop exactes.

98) [AICode/src/frontend/src/messageBus/__test__/useAiStream.hook.promptSourceIdentity.test.ts](#) — créé

- Vérifier sourceId unique par start, stabilité pendant le stream, transport et abort exact.

99) [usePromptRunController.hook.ts](#) — modifié

- startLiveObserver reçoit runRef+lease+replayMode.
- Utiliser le bootstrap replace ou continuation approprié.

100) [usePromptRunController.liveSessionObserver.test.tsx](#) — modifié

- Couvrir sourceId et les deux replay modes.

101) [AICode/src/frontend/src/app/prompt/Prompt/preparePromptContinuationStreaming.util.ts](#) — créé

- Extraire la préparation de continuation : assistant cible exact, base text, thinking avant assistant, refs, buffers, guards et fileEditsUiFinalized.
- Réutilisable par le clic Continue et le replay live.

102) [AICode/src/frontend/src/app/prompt/Prompt/__test__/preparePromptContinuationStreaming.util.test.ts](#) — créé

- Couvrir target exact, préfixe, placement thinking, assistant manquant, sources successives et aucune duplication.

103) [usePromptRunCommands.hook.ts](#) — modifié

- Remplacer le bootstrap inline de continueResponse par le helper partagé.
- Conserver Retry/Edit/Send inchangés.

104) [AICode/src/frontend/src/app/prompt/Prompt/__test__/usePromptRunCommands.continueResponse.test.tsx](#) — créé

- Vérifier que Continue utilise le helper et démarre une nouvelle source avec le même turn logique.

105) [Prompt.tsx](#) — modifié

- Capturer sourceId/adaptateur dans la query bootstrap.
- Utiliser le dispatcher d'adaptateurs.
- Déduire interactionsEnabled du view model/policy, pas du seul runtime mode.
- Autoriser draft persistence et interactions normales pendant replay normalPrompt.
- Garder ownsThreadForCleanup faux jusqu'au direct handoff normal, puis enregistrer le thread.
- Préserver classic hydration off pour tout live bootstrap.

106) [Prompt.liveSessionBootstrap.test.tsx](#) — modifié

- Couvrir dispatch EasyAgents/normal, sourceId, composer/interactions, query figée, hydration off et ownership cleanup.

107) [PromptInput.batchLiveViewerMode.test.tsx](#) — modifié

- Conserver viewer EasyAgents et ajouter un view model normalPrompt streaming prouvant composer visible + Stop normal sans banner Stop supplémentaire.

108) [EasyAgentsBatch.liveWatch.test.tsx](#) — modifié

- Adapter la query attendue avec sourceId et adaptateur.

109) [EasyAgentBatchSurface.test.tsx](#) — modifié

- Adapter la query live exacte sans changement UX.

110) [isomorphism-invariant-rule-between-streaming-and-rehydration-of-chat-sessions.programming-doc-readme.md](#) — modifié

- Ajouter la fermeture/réouverture normale comme troisième chemin devant produire le même état React que streaming continu et hydratation classique.
- Documenter le cas spécifique de continuation.

Récapitulatif exact des chemins de fichiers

Créés

- `AIcode/src/sharedlib/src/types/liveSessionTypes/LiveSessionReplay.type.ts`
- `AIcode/src/extension/src/aiTools/orch/liveSession/source/LiveSessionSourceLifecycle.service.ts`
- `AIcode/src/extension/src/aiTools/orch/liveSession/source/__test__/LiveSessionSourceLifecycle.service.test.ts`
- `AIcode/src/extension/src/aiTools/orch/liveSession/source/NormalPromptLiveSessionSource.service.ts`
- `AIcode/src/extension/src/aiTools/orch/liveSession/source/__test__/NormalPromptLiveSessionSource.service.test.ts`
- `AIcode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/abortSource.endpoint.ts`
- `AIcode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/__test__/abortSource.endpoint.test.ts`
- `AIcode/src/extension/src/extension/panels/prompt/liveSession/PromptLiveSessionPanelQuery.util.ts`
- `AIcode/src/extension/src/extension/panels/prompt/liveSession/__test__/PromptLiveSessionPanelQuery.util.test.ts`
- `AIcode/src/extension/src/extension/panels/prompt/liveSession/PromptLiveSessionPanelCoordinator.service.ts`
- `AIcode/src/extension/src/extension/panels/prompt/liveSession/__test__/PromptLiveSessionPanelCoordinator.service.test.ts`
- `AIcode/src/extension/src/extension/panels/prompt/liveSession/PromptLiveSessionDetachedStop.service.ts`
- `AIcode/src/extension/src/extension/panels/prompt/liveSession/__test__/PromptLiveSessionDetachedStop.service.test.ts`
- `AIcode/src/extension/src/extension/panels/sidebar/__test__/SidebarStore.handlers.manager.test.ts`
- `AIcode/src/frontend/src/app/prompt/liveSession/adapters/NormalPromptLiveSession.adapter.ts`
- `AIcode/src/frontend/src/app/prompt/liveSession/adapters/__test__/NormalPromptLiveSession.adapter.test.ts`
- `AIcode/src/frontend/src/app/prompt/liveSession/adapters/PromptLiveSessionBootstrap.adapter.ts`
- `AIcode/src/frontend/src/app/prompt/liveSession/adapters/__test__/PromptLiveSessionBootstrap.adapter.test.ts`
- `AIcode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.normalPrompt.test.tsx`
- `AIcode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.normalPromptIsomorphism.test.tsx`
- `AIcode/src/frontend/src/messageBus/__test__/useAiStream.hook.promptSourceIdentity.test.ts`
- `AIcode/src/frontend/src/app/prompt/Prompt/preparePromptContinuationStreaming.util.ts`
- `AIcode/src/frontend/src/app/prompt/Prompt/__test__/preparePromptContinuationStreaming.util.test.ts`
- `AIcode/src/frontend/src/app/prompt/Prompt/__test__/usePromptRunCommands.continueResponse.test.tsx`

Modifiés

- [`AIcode/src/sharedlib/src/types/liveSessionTypes/LiveSessionRun.type.ts`](#)
- [`AIcode/src/sharedlib/src/types/liveSessionTypes/LiveSessionPanel.type.ts`](#)
- [`AIcode/src/sharedlib/src/types/liveSessionTypes/LiveSessionOpenResult.type.ts`](#)
- [`AIcode/src/sharedlib/src/types/liveSessionTypes/test/LiveSessionContracts.test.ts`](#)
- [`AIcode/src/sharedlib/src/types/busTypes/ui/SidebarStore.ts`](#)
- [`AIcode/src/sharedlib/src/types/busTypes/ui/SidebarUiMessages.ts`](#)
- [`AIcode/src/sharedlib/src/types/busTypes/global/MessageRegistry.ts`](#)

- [AICode/src/extension/src/aiTools/orch/liveSession/liveSession.programming-doc-readme.md](#)
- [AICode/src/extension/src/aiTools/orch/liveSession/registry/LiveSessionRecord.type.ts](#)
- [AICode/src/extension/src/aiTools/orch/liveSession/registry/LiveSessionRegistry.service.ts](#)
- [AICode/src/extension/src/aiTools/orch/liveSession/registry/**test**/LiveSessionRegistry.service.test.ts](#)
- [AICode/src/extension/src/aiTools/orch/liveSession/core/LiveSessionService.ts](#)
- [AICode/src/extension/src/aiTools/orch/liveSession/core/**test**/LiveSessionService.test.ts](#)
- [AICode/src/extension/src/aiTools/orch/liveSession/open/LiveSessionOpen.service.ts](#)
- [AICode/src/extension/src/aiTools/orch/liveSession/open/**test**/LiveSessionOpen.service.test.ts](#)
- [AICode/src/extension/src/aiTools/orch/liveSession/replay/**test**/LiveSessionReplay.service.test.ts](#)
- [AICode/src/extension/src/aiTools/easyAgents/batch/run/liveSession/EasyAgentsBatchLiveSessionAdapter.service.ts](#)
- [AICode/src/extension/src/aiTools/easyAgents/batch/run/liveSession/**test**/EasyAgentsBatchLiveSessionAdapter.service.test.ts](#)
- [AICode/src/extension/src/aiTools/easyAgents/batch/run/EasyAgentsBatchRunExecution.service.ts](#)
- [AICode/src/extension/src/aiTools/easyAgents/batch/run/**test**/EasyAgentsBatchRunExecution.live-activity-and-gating.service.test.ts](#)
- [AICode/src/extension/src/aiTools/easyAgents/batch/run/EasyAgentsBatchRunChildExecution.service.ts](#)
- [AICode/src/extension/src/aiTools/easyAgents/batch/run/**test**/EasyAgentsBatchRunChildExecution.service.test.ts](#)
- [AICode/src/extension/src/aiTools/easyAgents/batch/run/EasyAgentsBatchRunManager.service.ts](#)
- [AICode/src/extension/src/aiTools/easyAgents/batch/run/**test**/EasyAgentsBatchRunExecution.testHarness.ts](#)
- [AICode/src/extension/src/aiTools/easyAgents/batch/run/**test**/EasyAgentsBatchRunLifecycle.service.test.ts](#)
- [AICode/src/extension/src/aiTools/easyAgents/batch/run/**test**/EasyAgentsBatchLastViewerStop.service.test.ts](#)
- [AICode/src/extension/src/aiTools/easyAgents/easyAgents.programming-doc-readme.md](#)
- [AICode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/open.endpoint.ts](#)
- [AICode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/**test**/open.endpoint.test.ts](#)
- [AICode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/stream.endpoint.ts](#)
- [AICode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/**test**/stream.endpoint.test.ts](#)
- [AICode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/abort.endpoint.ts](#)
- [AICode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/**test**/abort.endpoint.test.ts](#)
- [AICode/src/extension/src/slimbk-local/endpoints/ai/orch/query/ask/stream.endpoint.ts](#)
- [AICode/src/extension/src/slimbk-local/endpoints/ai/orch/query/ask/**test**/stream.endpoint.test.ts](#)
- [AICode/src/extension/src/messageBus/handlers/ia/liveSessionStreamHandlers.ts](#)
- [AICode/src/extension/src/messageBus/handlers/ia/**test**/liveSessionStreamHandlers.test.ts](#)
- [AICode/src/extension/src/messageBus/handlers/ia/iaStreamHandlers.ts](#)
- [AICode/src/extension/src/messageBus/handlers/ia/**test**/iaStreamHandlers.queryDoneSound.test.ts](#)
- [AICode/src/extension/src/messageBus/handlers/ia/**test**/iaStreamHandlers.runIdPropagation.test.ts](#)
- [AICode/src/extension/src/messageBus/handlers/ia/SidebarQueryRunningBridge.ts](#)
- [AICode/src/extension/src/messageBus/handlers/ia/**test**/SidebarQueryRunningBridge.race.test.ts](#)
- [AICode/src/extension/src/messageBus/handlers/ia/**test**/SidebarQueryRunningBridge.usage.test.ts](#)
- [AICode/src/extension/src/extension/panels/prompt/PromptThreadCleanup.manager.ts](#)
- [AICode/src/extension/src/extension/panels/prompt/**test**/PromptThreadCleanup.manager.attachmentsCleanup.test.ts](#)
- [AICode/src/extension/src/extension/panels/core/openPanel.util.ts](#)
- [AICode/src/extension/src/extension/panels/core/**test**/openPanel.util.test.ts](#)
- [AICode/src/extension/src/messageBus/handlers/ui/uiGenericHandlers.ts](#)
- [AICode/src/extension/src/messageBus/handlers/ui/**test**/uiGenericHandlers.openExternalUrl.test.ts](#)
- [AICode/src/extension/src/extension/commands/NewChat.command.ts](#)
- [AICode/src/extension/src/extension/commands/**test**/NewChat.command.test.ts](#)
- [AICode/src/extension/src/extension/uriHandlers/registerDeepLinkUriHandler.ts](#)
- [AICode/src/extension/src/extension/uriHandlers/**test**/registerDeepLinkUriHandler.scope.test.ts](#)
- [AICode/src/extension/src/extension/panels/sidebar/SidebarStore.queryWindow.listeners.manager.ts](#)
- [AICode/src/extension/src/extension/panels/sidebar/SidebarStore.queryWindow.mutations.manager.ts](#)
- [AICode/src/extension/src/extension/panels/sidebar/SidebarStore.core.manager.ts](#)
- [AICode/src/extension/src/extension/panels/sidebar/SidebarStore.rehydrate.manager.ts](#)
- [AICode/src/extension/src/extension/panels/sidebar/**test**/SidebarStore.rehydrate.manager.test.ts](#)
- [AICode/src/extension/src/extension/panels/sidebar/**test**/SidebarStore.manager.queryRunningUpsert.test.ts](#)

- [AICode/src/extension/src/extension/panels/sidebar/SidebarStore.chatTitles.manager.ts](#)
- [AICode/src/extension/src/extension/panels/sidebar/SidebarStore.handlers.manager.ts](#)
- [AICode/src/extension/src/slimbk-local/endpoints/ai/orch/chat/delete.endpoint.ts](#)
- [AICode/src/extension/src/slimbk-local/endpoints/ai/orch/chat/test/list.endpoint.windowsLock.eperm.test.ts](#)
- [AICode/src/extension/src/slimbk-local/endpoints/ai/orch/chat/test/diskAliases.lastWins.endpoints.test.ts](#)
- [AICode/src/extension/src/aiTools/orch/context/persistence/test/ChatPersistenceWriter.windowsFallback.test.ts](#)
- [AICode/src/extension/src/aiTools/orch/context/memory/core/mem/test/OrchMemory.naive_title.user_prompt_specif_multiline.e2e.repro.test.ts](#)
- [AICode/src/frontend/src/app/sidebar/sidebar-activity/SidebarActivityQuery.tsx](#)
- [AICode/src/frontend/src/app/sidebar/sidebar-activity/test/SidebarActivityQuery.test.tsx](#)
- [AICode/src/frontend/src/app/prompt/liveSession/core/PromptLiveSession.types.ts](#)
- [AICode/src/frontend/src/app/prompt/liveSession/core/usePromptLiveSession.hook.ts](#)
- [AICode/src/frontend/src/app/prompt/liveSession/core/test/usePromptLiveSession.hydration.test.tsx](#)
- [AICode/src/frontend/src/app/prompt/liveSession/core/test/usePromptLiveSession.handoff.test.tsx](#)
- [AICode/src/frontend/src/app/prompt/liveSession/adapters/EasyAgentsBatchLiveSession.adapter.ts](#)
- [AICode/src/frontend/src/app/prompt/liveSession/adapters/test/EasyAgentsBatchLiveSession.adapter.test.ts](#)
- [AICode/src/frontend/src/messageBus/AiStreamStartArgs.type.ts](#)
- [AICode/src/frontend/src/messageBus/useAiStream.hook.ts](#)
- [AICode/src/frontend/src/messageBus/test/useAiStream.hook.liveSessionObserver.test.ts](#)
- [AICode/src/frontend/src/app/prompt/Prompt/usePromptRunController.hook.ts](#)
- [AICode/src/frontend/src/app/prompt/Prompt/test/usePromptRunController.liveSessionObserver.test.tsx](#)
- [AICode/src/frontend/src/app/prompt/Prompt/usePromptRunCommands.hook.ts](#)
- [AICode/src/frontend/src/app/prompt/Prompt/Prompt.tsx](#)
- [AICode/src/frontend/src/app/prompt/Prompt/test/Prompt.liveSessionBootstrap.test.tsx](#)
- [AICode/src/frontend/src/app/prompt/PromptInput/test/PromptInput.batchLiveViewerMode.test.tsx](#)
- [AICode/src/frontend/src/app/easy-agents/easy-agents-batch/core/test/EasyAgentsBatch.liveWatch.test.tsx](#)
- [AICode/src/frontend/src/app/easy-agents/easy-agents-item/core/batch/test/EasyAgentBatchSurface.test.tsx](#)
- [AICode/src/frontend/src/app/prompt/hydration/isomorphism-invariant-rule-between-streaming-and-rehydration-of-chat-sessions.programming-doc-readme.md](#)

Déplacés

Aucun.

Supprimés

Aucun.

Facteur de risque de régression

1. **Risque calculé : 89 %.** Criticité 5, couplage 5, blast radius 5, incertitude 4, faible confiance 2. Base = $10 \times 5 + 3 \times 5 + 2 \times 5 + 2 \times 4 + 3 \times 2 = 89$; risque = $\text{round}(89 \times 5/5) = 89 \%$.
2. **Zones complexes :** sources successives partageant un turn, réservation avant `askStream`, lifecycle `Readable` lazy, baseline `continueResponse`, `sourceId` transporté dans tous les chemins, races panel open/close/cleanup, direct handoff frontend, conservation draft/attachments, source Stop précoce, Stop Activité sans lease, SidebarStore union stricte, rehydrate de sources détachées, son terminal, interact gates, tool calls, file edits, terminal persistant après source ultérieure et coexistence EasyAgents/Prompt normal.
3. **Justification :** la fondation tranche 1 est solide et fortement testée, mais cette tranche modifie le chemin normal de toutes les requêtes Prompt et la politique historique de dispose. Une erreur peut perdre/dupliquer un chunk, clear la mémoire pendant une réouverture, interrompre la mauvaise source, laisser un processus orphelin ou produire un état React différent du stream continu. L'identité `sourceId`, les réservations, les epochs, les queues par locator, les claims synchrones et les tests d'isomorphisme/races sont donc obligatoires.

Résumé

- Étendre `liveSession` aux sources Prompt normales en une seule implémentation complète.
- Ajouter `sourceId` afin de distinguer Retry/Edit/Continue partageant un `turnId`.
- Réserver puis enregistrer les sources normales, bufferiser avant forwarding, et utiliser un lifecycle lazy partagé avec EasyAgents.
- Capturer une baseline immuable pour `continueResponse` et introduire deux replay modes.
- Ne plus tuer une source active à la fermeture du panneau ; différer cleanup mémoire/attachments/son.
- Sérialiser ouverture, dispose et cleanup terminal par locator.
- Rouvrir automatiquement live depuis Activité, Chats ou deep link lorsqu'une source exacte est active.

- Restaurer le compositeur normal dès que le replay est armé ; conserver seulement le verrou Send normal.
- Réutiliser le Stop normal dans le Prompt et ajouter uniquement le Stop détaché dans Activité.
- Garder l'item Activité si panneau ouvert ou source active ; le supprimer si panneau fermé et source terminée.
- Préserver intégralement l'UX EasyAgents et les algorithmes Queue A/Queue B.
- Tester chaque unité, chaque course et l'isomorphisme streaming continu / fermeture-réouverture.

Nom du commit

feat(live-session): keep prompt runs alive across panel close and reopen

SPECIF_FINALISER_LIVESESSION_POUR_LES_SESSIONS_PROMPT_NORMALES_20260711_232041

Copy

Refine

Code

Verify

Spécification créée

La spécification finale couvre l'implémentation **en une seule tranche cohérente**, depuis le commit f65adc8e5172cc1e4f9781bf54ffa6dd089d3d33, avec :

- l'identité exacte locator + turnId + sourceId ;
- les sources normales et leurs réservations ;
- le cas spécial continueResponse ;
- le lifecycle Readable partagé ;
- la fermeture sans interruption ;
- la coordination réouverture/cleanup ;
- le retour rapide au Prompt normal ;
- le SidebarStore étendu ;
- le bouton Stop détaché ;
- la suppression de l'item lorsque panneau fermé + processus terminé ;
- les tests unitaires, d'intégration, de courses et d'isomorphisme ;
- les mises à jour des programming docs ;
- **110 fichiers planifiés**, dont **24 créations**, sans édition manuelle des fichiers générés.

</ASSISTANT>

SHOW ALLPage 1 / 1