

Permalink URI:

vscode://joelabenhaim.aicode-ja/chat?sessionId=bd049e13cb7c4fbc834a0e2e31e7d9ed-VS-V%C3%A9rifier-Sessions

<USER>

[Verify]SPECIF FINALISER LIVESESSION POUR LES SESSIONS PROMPT NORMALES 20260712 204731

</USER>

<ASSISTANT>

I'm thinking...

</ASSISTANT>

<ASSISTANT>

🔧  scanning git: 📁 "État du dépôt courant", 📁 "Amplitude des changements", 📁 "Fichiers réellement chang"

🔧  scanning recent files

🔧  reading project: 📁 g "Docs architecture live", 📁 g "Docs architecture Prompt", 📁 g "Docs architecture liées", 📁 g "Docs architecture bus", 📄 liveSession.programming-doc-readme.md, 📄 easyAgents.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, 📄 prompt-stream-dispatch-lock.programming-doc-readme.md, 📄 fallback-persist-and-title.programming-doc-readme.md

🔧  scanning git: 📁 "Historique récent", 📁 "Inventaire du commit", 📁 "Résumé quantitatif commit", 📁 "Diff depuis fondation", 📁 "Tests et sources changés"

🔧  reading project: 📄 e "Ancienne API abort", 📄 e "Ancien store windows", 📄 e "Ancienne identité stream", 📄 e "Tentatives replay Open", 📄 e "Identité stream frontend", 📄 e "Erreur Slimbk body", 📄 e "Origine exacte webview", 📄 e "Descripteur panneau live", 📄 e "Finalisation forwarding", 📄 h "Claims destructions live", 📄 h "Barrières Prompt stream", ❌ 📄 h "Panneau et activité",

📄 LiveSessionRun.type.ts,

📄 LiveSessionPanel.type.ts,

📄 LiveSessionOpenResult.type.ts,

📄 LiveSessionReplay.type.ts,

📄 SidebarStore.ts,

📄 AiStreamStartArgs.type.ts,

📄 LiveSessionRecord.type.ts,

📄 LiveSessionService.ts,

📄 LiveSessionSourceLifecycle.service.ts,

📄 NormalPromptLiveSessionSource.service.ts,

📄 parseSlimbkStreamBodyError.util.ts, 📄 extTypedMessageBus.util.ts, 📄 LiveSessionRegistry.service.ts [L1-

L460], 📄 LiveSessionRegistry.service.ts [L461-L850], 📄 LiveSessionRegistry.service.ts [L851-

L1200],

📄 LiveSessionOpen.service.ts,

📄 LiveSessionReplay.service.ts,

📄 stream.endpoint.ts,

📄 abortSource.endpoint.ts,

📄 abort.endpoint.ts,

📄 open.endpoint.ts,

📄 stream.endpoint.ts,

📄 aiStopSuffix.util.ts,

📄 iaStreamHandlers.ts,

📄 liveSessionStreamHandlers.ts,

📄 PromptLiveSessionPanelQuery.util.ts,

📄 PromptLiveSessionDetachedStop.service.ts,

📄 PromptLiveSessionPanelCoordinator.service.ts,

📄 SidebarQueryRunningBridge.ts, 📄 PromptThreadCleanup.manager.ts, 📄 e "Settlement après

promotion", 📄 e "Attente source settlement", 📄 e "Annulation promue test", 📄 h "Tests latch source", 📄 e "Finalisation

handler source", 📄 "Tests du latch source", 📄 e "Descripteur précédent", 📄 h "Réouverture après fin", 📄 e "Scénarios

nouvelle source", 📄 e "Handoff prêt coordinateur", 📄 "Courses du

coordinateur",

📄 SidebarStore.queryWindow.listeners.manager.ts,

📄 SidebarStore.queryWindow.mutations.manager.ts,

[SidebarStore.rehydrate.manager.ts](#),
[SidebarStore.handlers.manager.ts](#), [SidebarActivityQuery.tsx](#), [SidebarUiMessages.ts](#),  e "[Option disableAfterClick](#)",
[VscButton.tsx](#), [VscButton.module.css](#), [frontMessageBus.util.ts](#), [SlimbkResult.type.ts](#), [SlimbkEndpoint.ts](#),
[Slimbk.ts](#) [L1-L260],
[slimbkBuildHeader.ts](#),
[Slimbk.streamingErrorHandler.test.ts](#), [slimbkBuildHeader.errorCode.test.ts](#), [Slimbk.ts](#) [L261-L421]
  scanning git:  ["Diff Slimbk précis"](#)
  reading project:  e "[Contrats streams bus](#)",  "[Contrats bus exacts](#)",
[usePromptLiveSession.hook.ts](#),
[PromptLiveSession.types.ts](#),
[NormalPromptLiveSession.adapter.ts](#),
[PromptLiveSessionBootstrap.adapter.ts](#),
[EasyAgentsBatchLiveSession.adapter.ts](#),
[applyHydratedPromptSessionState.util.ts](#), [preparePromptContinuationStreaming.util.ts](#),  e "[Composer interactif échec](#)",  e "[Surface replay failed](#)",  h "[Tests brouillon replay](#)",  "[Contrat adaptateur normal](#)", [useAiStream.hook.ts](#) [L1-L360], [useAiStream.hook.ts](#) [L361-L720], [useAiStream.hook.ts](#) [L721-L1000], [useAiStream.hook.ts](#) [L1001-L1172],
[usePromptRunController.finalization.hook.ts](#),
[usePromptCancelController.hook.ts](#),
[usePromptRunController.hook.ts](#),
[usePromptRunCommands.hook.ts](#), [preparePromptStreamingTurn.util.ts](#),  e "[Stop live Prompt intégration](#)",  "[Intégration Stop et vue](#)", [Prompt.tsx](#) [L1-L220],  e "[Garde autoRun live](#)", [Prompt.tsx](#) [L768-L990], [Prompt.tsx](#) [L1090-L1428], [Prompt.tsx](#) [L220-L540], [usePromptThinkingLifecycle.hook.ts](#),  e "[Vue live dans composer](#)", [PromptInput.tsx](#) [L320-L470],  e "[Rendu composer live](#)", [PromptInput.tsx](#) [L1800-L2050],
[PromptInput.tsx](#) [L2051-L2140], [openPanel.util.ts](#) [L1-L260], [openPanel.util.ts](#) [L261-L520], [openPanel.util.ts](#) [L521-L678], [delete.endpoint.ts](#),  e "[Décodage chaîne V1](#)",  "[Cas protocole V1](#)", [aiStopSuffix.util.test.ts](#) [L70-L130], [extMessageBus.util.ts](#) [L1-L260], [extMessageBus.util.ts](#) [L261-L383],
[PromptQueryDoneSound.manager.ts](#),
[EasyAgentsBatchLiveSessionAdapter.service.ts](#), [EasyAgentsBatchRunRetry.service.ts](#) [L240-L380], [EasyAgentsBatchRunRetry.service.ts](#) [L380-L560], [EasyAgentsBatchRunChildExecution.service.ts](#) [L1-L260],  e "[Rehydrate source settled](#)",  h "[Terminal avant handoff](#)",  e "[Rehydrate skip settled](#)", [SidebarStore.core.manager.ts](#) [L1-L180], [SidebarStore.core.manager.ts](#) [L181-L200],  e "[Client stream local](#)",
[OrchMessagesMemoryAssistantPatch.ts](#), [OrchMessagesMemoryAssistantPatch.errorSuffix.test.ts](#),  e "[Patch terminal erreur](#)",  e "[EasyAgents runRef exact](#)",  e "[SourceId EasyAgents](#)",  r "[openCentralPanel](#)", [registerDeepLinkUriHandler.ts](#) [L120-L155], [usePromptSessionLocator.hook.ts](#),  "[Horodatage source Easy](#)", [PromptLiveSessionBanner.tsx](#),  e "[Ancien type Activity](#)",  e "[Ancien champ windows](#)",  e "[Param EasyAgents dupliqué](#)",  h "[Delete autorité observer](#)", [delete.endpoint.liveSession.test.ts](#) [L217-L350]
  scanning git:  ["Qualité du diff"](#),  ["Désactivations ajoutées"](#),  ["TODO ajoutés"](#),   ["Imports dynamiques ajout"](#),  ["Any explicites ajoutés"](#),   ["Require ajoutés"](#),  ["Imports dynamiques ajout"](#),  ["Require ajoutés"](#),  ["Casts doubles ajoutés"](#),  ["Non-null ajoutés"](#),  ["Casts nouveaux commandes"](#),  ["Diff plan streaming"](#)
  reading project: [usePromptLiveSession.normalPrompt.test.tsx](#) [L1-L382],
[usePromptSessionHydration.hook.ts](#),
[frontSessionHydration.type.ts](#),
[PromptInput.module.css](#), [PromptLiveSessionBanner.module.css](#), [AiQueryOptions.type.ts](#),  e "[Réutilisation lease après éviction](#)",  h "[Lease permanente exacte](#)", [LiveSessionRegistry.service.test.ts](#) [L1-L110], [useStreamListener.hook.ts](#), [usePromptRunCommands.startAcceptance.test.tsx](#) [L1-L192],  r "[rehydrateQueryActivities](#)",  h "[Course rehydrate source](#)"

Verdict exécutif

Joel, la spécification est remarquablement solide — elle a presque autant de barrières qu’un château fort, ce qui est exactement adapté à ces courses asynchrones. 🏰

La vérification porte sur le dépôt **propre** au commit courant 3101cd8a, intitulé `feat(live-session): keep prompt runs alive across panel close and reopen`. Le commit représente **185 fichiers changés**, environ **20 703 insertions** et **11 460 suppressions**.

J’ai également lu les documents d’architecture pertinents :

- [architecture générique liveSession](#)
- [architecture EasyAgents](#)
- [invariant d’isomorphisme streaming/hydratation](#)
- [invariants thinking/hourglass](#)
- [architecture Queue B](#)
- [barrière fallback persist](#)

Résultat global

Dimension	Verdict	Estimation
Couverture structurelle de la spécification	✅ Très élevée	98 %
Conformité technique et comportementale	⚠ Presque complète	93–94 %
Couverture des scénarios par les tests présents	✅ Très forte, avec des angles morts précis	90 %
Niveau de confiance après lecture du code	✅ Élevé	95 %
Prêt à déclarer la tranche totalement finalisée	❌ Non	—

1) La spécification entière est-elle implémentée ?

Non, pas intégralement.

Toutes les grandes briques demandées existent :

- identité exacte `locator + turnId + sourceId` ;
- registre avec réservations, époques, tombstones, retraites et claims destructifs ;
- poursuite des sources Prompt après fermeture ;
- reconstruction Open + replay + tail ;
- source normale avec persistance des erreurs avant append ;
- protocole V1 premier header ;
- filtrage des enveloppes Slimbk post-header ;
- origine exacte de webview ;
- frontend transactionnel avec `streamInstanceId` ;
- direct handoff normal Prompt ;
- hydratation `preserveLocal` ;
- activité source ouverte/fermée ;
- Stop détaché ;
- coordination de panneau ;
- protections Delete/Retry ;
- conservation d’EasyAgents ;
- continuation/thinking ;
- barrière `fallback_persisted` ;
- documentation et tests conséquents.

Je n’ai trouvé **aucune famille architecturale majeure entièrement absente**. L’implémentation dépasse même la liste de fichiers prévue en adaptant proprement le framework Slimbk et son générateur pour permettre un header d’erreur non-200 laissant le body ouvert.

En revanche, j’ai identifié :

- **quatre défauts fonctionnels ou concurrentiels** ;
- **une régression de sûreté de typage** ;
- **un écart architectural mineur** sur la construction préalable des plans frontend.

Estimation de complétude : 94 %.

Conclusion Q1 : ❌ la tranche est presque entièrement codée, mais elle ne doit pas encore être considérée comme totalement finalisée.

2) L'implémentation actuelle fait-elle ce qu'elle doit faire, pour chaque fonctionnalité technique demandée ?

Fonctionnalité demandée	Éléments vérifiés
Identité exacte <code>locator + turnId + sourceId</code>	Schémas stricts et run key structurelle dans LiveSessionRun.ts
Séparation <code>sourceId</code> backend / <code>streamInstanceId</code> frontend / tentative Open	<code>sourceId</code> est transporté ; <code>streamInstanceId</code> reste frontend-only sous <code>openAttemptRef</code>
Descripteurs live stricts	Union stricte, close policy, adaptateur, ligne EasyAgents et égalité dans LiveSessionPanel.type.ts
EasyAgents/normalPrompt	
Détection des paramètres partiels, inconnus et dupliqués	Vérification indépendante extension/frontend avec <code>getAll()</code> côté
Priorité de la query live explicite et rejet avant opener	Implémentée dans PromptLiveSessionPanelQuery.util.ts et PromptLiveSessionPanelQuery.ts
Résultat Open strict et enveloppe frontend sélective	Branches <code>live</code> , <code>terminal</code> , <code>not_ready</code> , <code>not_found</code> , propriétés
Réservation synchrone avant tout <code>await</code>	Première mutation du handler source, promotion sans nouvelle époque
Refus des réservations dupliquées et tombstones exactes	Présent dans LiveSessionRegistry.service.ts
Finalisation post-forwarding à cinq résultats	Les cinq résultats exacts existent avec priorité <code>retiredMissing</code>
Latch de settlement de la source	Le latch existe, mais peut être libéré avant le drainage FIFO et la fin
Buffer complet, ordonné, non compressé et non borné	Aucun plafond ou mécanisme de compression ajouté
Replay backlog + tail avec lease exacte	Curseur, revision, missed-wakeup protection et autorisation par élé
Reconstruction <code>replaceTurnAssistants</code>	Retrait des seuls assistants du tour demandé
Reconstruction <code>continueAssistant</code>	Superposition de la baseline immuable capturée avant <code>askStream</code>
Source Prompt normale	Lifecycle lazy, baseline, persistance des erreurs, append-before-for
Persistance de tout <code>meta.error</code> avant <code>append</code>	Implémentée dans NormalPromptLiveSessionSource.service.ts
Suffixes	
SYSTEM_ERROR/INTERRUPTED	Implémentés et testés via OrchMessagesMemoryAssistantPat
idempotents	
Protocole V1 du premier <code>meta.error</code>	Header structuré strict, body maintenu ouvert et terminal reconstru
Enveloppe Slimbk non-200 après header 200	Détectée avant toute logique <code>AiStreamedChunk</code> dans parseSli
Origine exacte de webview	Contexte extension-only, non sérialisé, comparé au panneau actuel
Start frontend synchrone et single-flight	Retour strict accepté/refusé ; IDs créés après acquisition ; refus san
Plans purs avant Start pour les sept commandes	Refus sans mutation correctement obtenu, mais certains plans repl
Guards <code>sourceId + streamInstanceId</code>	Présents dans callbacks, Queue A, Queue B et continuations post-a
Drainage Queue A + Queue B + commit avant outcome	Implémenté avec conservation des erreurs par tentative
Terminal data puis <code>transportError</code> sans doublon	Le terminal existant est conservé et aucun second suffixe n'est inj
Direct handoff normal Prompt	Terminal + completed + queues + finalisation exacte, sans second
Retry <code>preserveLocal</code> opaque	Aucun accès aux champs <code>draft/attachments/nextTurnId</code> ignorés
Continuation et thinking par tentative	Assistant réutilisé, thinking frais, séparateur unique et guards par s
Barrière <code>{turnId, streamInstanceId}</code> de fallback	Clear sur ack exact, démasquage sans clear sur finalisation sans ac
Composer normal Prompt interactif après armement du replay	Draft et attachments activés pendant <code>liveSessionStreaming</code> , S
Stop unique dans Prompt	Le bandeau <code>normalPrompt</code> n'ajoute aucun Stop ; le compositeur re
Stop détaché dans Activité	Source/closed uniquement, <code>SquareStop</code> , <code>small</code> , <code>danger</code> , désactiva
Son sans doublon lors du replay	Source normale unique, aucun nouvel attempt lors de la reconnexion
Activité ouverte/fermée, Idle ou suppression	Les transitions normales sont exactes, mais la réhydratation contie

Fonctionnalité demandée

Éléments vérifiés

terminale

Coordination panneau/cleanup sous claim

Delete destructif

Retry destructif EasyAgents

Préservation EasyAgents historique

Programming docs

Tests des courses principales

Contraintes de qualité

Typage Retry évolutif

Conclusion de l'analyse fonctionnelle

Le cœur produit demandé fonctionne bien conceptuellement :

- une source Prompt normale peut être retenue après fermeture ;
- son état peut être projeté dans Activité ;
- une réouverture obtient baseline puis replay ;
- le compositeur peut préparer le brouillon suivant ;
- les chunks sont protégés par identité exacte ;
- le handoff final est direct ;
- EasyAgents conserve ses propres politiques.

Les défauts se concentrent sur les **barrières destructives et la reconstruction d'Activité**, pas sur le transport principal des tokens.

Conclusion Q2 : $\triangle$ le comportement principal est opérationnel, mais les invariants de settlement, de Delete et de réhydratation Activity ne sont pas totalement respectés.

3) Y a-t-il des bugs ?

Oui : quatre bugs fonctionnels confirmés par le code, plus une régression de sûreté de typage.



Bug 1 — Le latch de settlement est libéré trop tôt après promotion

Sévérité : élevée — confiance : 100 %.

Dans `LiveSessionRegistry.service.ts`, une annulation appelée avec un handle déjà promu résout le latch :

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

Copy

```
if (this.reservationsByRunKey.get(runKey) !== reservation) {
  // The exact owner may report cancellation after destructive eviction
  removed its indexes.
  this.resolveSourceSettlement(reservation.runRef);
  return false;
}
```

Or `iaStreamHandlers.ts` appelle systématiquement cette annulation après le retour du transport, **avant** le drainage de la FIFO :

[AICode/src/extension/src/messageBus/handlers/ia/iaStreamHandlers.ts](#)

Copy

```
unstageNormalPromptLiveSessionReservation({ runRef, reservation });
liveSession.cancelSourceReservation(reservation);

try {
  await fifo;
```

Après une promotion réussie :

1. la réservation a disparu de `reservationsByRunKey` ;
2. le handle existe encore dans le `WeakMap` ;

FIFO par locator, cleanup différé, claims et wake-ups présents

Correct pour une source normale, mais l'autorité observer du regis

Double guard, claim exact, éviction sous handle, release `finally`

`sourceId=turnId`, observer ownership, last-viewer kill, UX rea

Les six documents demandés sont mis à jour et globalement aligné

Couverture très importante, mais trois angles morts correspondent

Diff propre, aucun nouveau TODO, `eslint-disable` ou requi

tests

Le garde exhaustif des clés `AiQueryOptions` a été supprimé

3. `cancelSourceReservation()` entre donc dans la branche ci-dessus ;
4. `waitForSourceSettlement()` est libéré avant `await fifo` et avant `finalizeSourceAfterForwarding()`.

Effet

Une suppression destructive peut croire que le handler source a entièrement abandonné son autorité alors que :

- les chunks extension ne sont pas tous relayés ;
- la finalisation registre n'a pas eu lieu ;
- le son et le coordinateur n'ont pas encore convergé.

La persistance réalisée dans le service source est normalement déjà terminée à cet instant, ce qui limite le risque de résurrection disque, mais **la barrière spécifiée n'est plus une preuve de fin du handler.**

Angle mort de test

Les tests couvrent :

- annulation avant promotion ;
- finalisation normale ;
- finalisation après éviction ;

mais pas `registerRun` → `cancelSourceReservation(handle)` → `waiter` encore bloqué.



Bug 2 — Delete ne vérifie pas l'autorité observer du registre

Sévérité : critique — confiance : 100 %.

Dans [delete.endpoint.ts](#) :

- un observer est bloqué seulement si la projection Sidebar indique `Running/observer` ;
- après acquisition du claim, la lecture du registre ne conserve que les sources `ownership === 'source'`.

[AICode/src/extension/src/slimbk-local/endpoints/ai/orch/chat/delete.endpoint.ts](#)

Copy

```
const activeSourceRunRefs = liveSession
  .listActiveSnapshots()
  .filter(
    (snapshot) =>
      snapshot.runRef.locator.sessionId ===
        locator.sessionId &&
      snapshot.runRef.locator.storageScopeRelPath ===
        locator.storageScopeRelPath &&
      snapshot.ownership === 'source' &&
```

Scénario défectueux

1. Un observer EasyAgents est toujours actif dans le registre.
2. Sa projection Activity est absente, obsolète ou devenue Idle.
3. Delete ne voit aucun `Running/observer` dans le store.
4. Le claim destructif est acquis.
5. Aucun contrôle autoritatif du registre ne bloque l'observer.
6. Le panneau et la session durable peuvent être supprimés pendant que la source observer continue sa convergence. Cela contredit directement le commentaire du endpoint indiquant que les runs observer doivent être arrêtés explicitement.

Risque

La source EasyAgents peut encore :

- persister sa conversation enfant ;
- mettre à jour son état Batch ;
- poursuivre sa finalisation métier après la suppression.

Le registre doit être l'autorité, tandis que le SidebarStore doit rester une projection UI.



Bug 3 — Un observer `sourceSettled` mais non `handoffReady` devient Idle lors de la réhydratation

Sévérité : élevée — confiance : 100 %.

Dans [SidebarStore.rehydrate.manager.ts](#) :

[AICode/src/extension/src/extension/panels/sidebar/SidebarStore.rehydrate.manager.ts](#)

Copy

```

if (
  snapshot.kind === 'record' &&
  (snapshot.sourceSettled || snapshot.handoffReady)
) {
  continue;
}

```

Pour EasyAgents, `markSourceSettled()` et `markHandoffReady()` sont volontairement deux transitions différentes :

1. le stream source est terminé ;
2. la convergence Git/persistence Batch continue ;
3. le replay viewer attend encore `handoffReady`.

Pendant cet intervalle, le viewer est toujours actif et son Activité doit rester **Running/observer/open**.

Le code ignore pourtant le record dès `sourceSettled=true`, puis la boucle des panneaux ouverts recrée une projection **Idle/open**.

Conséquences

- Activité devient faussement Idle alors que le replay attend encore ;
- le timer et l'état visuel divergent ;
- le garde Delete basé sur Activity peut être contourné, ce qui renforce le bug précédent.

Aucun test de réhydratation ne couvre

actuellement `ownership='observer', sourceSettled=true, handoffReady=false`.



Bug 4 — La reconstruction d'Activité utilise des snapshots devenus obsolètes pendant ses propres `await`

Sévérité : élevée — confiance : 95 %.

Toujours dans `SidebarStore.rehydrate.manager.ts`, l'algorithme :

1. capture les panneaux ouverts ;
2. capture les réservations/records ;
3. effectue plusieurs lectures persistantes asynchrones ;
4. vide intégralement `queryState.items` ;
5. reconstruit séquentiellement depuis les captures antérieures.

Pendant les lectures persistantes, il est possible qu'un panneau :

- s'ouvre ;
- se ferme ;
- démarre une nouvelle source ;
- passe à une source successive ;
- reçoive un Stop.

La reconstruction ne recapture pas les autorités après ces `await`.

Exemple concret

- les snapshots sont capturés sans source active ;
- une source démarre pendant une lecture disque et installe son runtime Activity ;
- la réhydratation reprend, vide le store et reconstruit depuis l'ancien snapshot ;
- la source en cours disparaît d'Activité ou redevient Idle.

Le runtime reste actif, mais les ticks et les chunks ne recréent pas nécessairement une ligne Running manquante.

Correction architecturale attendue

La projection devrait être construite hors store puis installée **atomiquement** depuis une recapture récente des autorités, avec revalidation ou génération si l'état change pendant les lectures.



Régression de sûreté — Le garde exhaustif de `AiQueryOptions` a été supprimé

Sévérité actuelle : moyenne — confiance : 100 %.

Le diff de `usePromptRunCommands.hook.ts` supprime le garde de compilation qui vérifiait que toutes les clés de `AiQueryOptions` avaient reçu une décision explicite pour Retry.

Le code actuel reste correct avec les cinq propriétés existantes de `AiQueryOptions.type.ts`. En revanche, l'ajout futur d'une option ne provoquera plus l'erreur de compilation volontaire : Retry pourra silencieusement l'ignorer.

La consigne de vérification demande explicitement de signaler une amélioration ou modification qui **relâche une contrainte de typage stricte**. C'est donc un écart à corriger, même sans panne immédiate.



Écart architectural non bloquant — Tous les plans UI ne sont pas préparés avant Start

`preparePromptStreamingTurn.util.ts` expose bien `createPromptStreamingTurnPlan()`, mais `Send`, `SPECIF` et `Edit` appellent encore la fonction mutante après l'acceptation au lieu de :

1. construire le plan complet ;
2. appeler `start()` ;
3. committer ce plan.

Le contrat essentiel « aucune mutation sur refus » est respecté et testé. L'écart porte sur la garantie plus forte : **aucune construction potentiellement faillible après acceptation du transport.**

Conclusion Q3 : ❌ il existe trois défauts importants de sûreté/destruction, un défaut de réhydratation concurrente supplémentaire et une régression de typage à corriger.

4) Quel est mon avis sur le code source et la spécification ?

Avis sur la spécification

Critère

Avis

Précision	Excellente
Gestion des courses	Exceptionnellement détaillée
Séparation des identités	Très bien conçue
Destruction/cleanup	Exigeante mais justifiée
Testabilité	Très bonne
Facilité d'implémentation atomique	Faible , en raison du blast radius
Note	9,5/10

Le risque annoncé de **92 %** était réaliste. La spécification touche simultanément :

- transport ;
- registre ;
- persistance ;
- webviews ;
- Activity ;
- React ;
- queues ;
- son ;
- Delete ;
- Retry ;
- EasyAgents.

Les défauts trouvés se situent précisément dans les zones que la spécification identifiait comme dangereuses.

Avis sur le code source



Points particulièrement réussis

- Les identités exactes sont omniprésentes et cohérentes.
- Les run keys et locator keys utilisent des tuples JSON sans collisions de séparateur.
- Les réservations et claims sont opaques.
- L'origine exacte de webview est bien non sérialisée.
- Le service source normal respecte correctement la barrière durable-before-append.
- Le premier header V1 et les erreurs Slimbk de body sont séparés proprement.
- Le direct handoff ne refait ni Open ni hydratation du compositeur.
- `preserveLocal` évite réellement la lecture des champs protégés.
- Les callbacks frontend sont largement protégés contre les tentatives obsolètes.
- La gestion terminal + `transportError` est bien pensée.
- Les commentaires techniques sont nombreux et utiles.
- Les tests utilisent globalement de bons sélecteurs et des mocks strictement typés.
- L'extension du framework Slimbk, non prévue dans la liste initiale, est **justifiée, étroite et testée**.

⚠️ Points faibles

- Le commit atomique de 185 fichiers reste extrêmement difficile à relire et maintenir.
- La réhydratation Activity possède plusieurs autorités et effectue une reconstruction destructive après des lectures asynchrones.
- La sémantique exacte du latch de settlement n'est pas encodée assez fortement dans l'API.
- Delete dépend encore d'une projection UI pour une décision métier destructive.

- useAiStream.hook.ts est devenu un hotspot très important : les invariants sont bons, mais la densité de responsabilités augmente fortement.
- La suppression du garde exhaustif `AiQueryOptions` réduit la protection contre les futurs changements.
- Les tests sont nombreux, mais les trois trous principaux sont précisément des **courses entre autorités**, donc des scénarios qui doivent être matérialisés explicitement.

Notes estimatives

Élément	Note
Architecture générale	9,1/10
Correction fonctionnelle actuelle	8,2/10
Qualité du typage	8,7/10
Qualité des tests	8,8/10
Maintenabilité	8,4/10
Qualité attendue après corrections	9,3/10

Conclusion Q4 :  la conception est de très haut niveau et l'implémentation est globalement professionnelle, mais quelques courses destructives empêchent encore de qualifier le résultat de production-final.

5) Quel est le résumé du rapport ?

Sujet	Résultat	Priorité
Fonction centrale « continuer après fermeture »	 Implémentée	—
Réouverture baseline + replay + tail	 Implémentée	—
Identités exactes source/stream/Open	 Implémentées	—
Direct handoff sans second Open	 Implémenté	—
Composer interactif avec Send verrouillé	 Implémenté	—
Préservation locale du brouillon lors des retries	 Implémentée	—
Stop détaché et propagation <code>stopRequested</code>	 Implémentés	—
Persistance des <code>meta.error</code> avant append	 Implémentée	—
Premier header V1 et erreur Slimbk post-header	 Implémentés	—
Exactitude de l'origine webview	 Implémentée	—
Conservation d'EasyAgents	 Implémentée	—
Latch source après promotion	 Libéré trop tôt	P1
Delete avec observer registre actif	 Autorité insuffisante	P0
Observer settled/non-handoff dans Activity	 Projeté Idle	P1
Réhydratation Activity concurrente	 Snapshots obsolètes possibles	P1
Exhaustivité future de <code>AiQueryOptions</code>	 Protection supprimée	P2
Plans replace intégralement préparés avant Start	 Partiel	P2
Tests	 Très nombreux, trois courses manquantes	P1
Recommandation de livraison	 Corriger P0/P1 avant finalisation	—
Conclusion Q5 : la tranche est fonctionnellement très avancée, mais elle conserve un risque destructif réel autour de Delete et deux défauts de convergence Activity/settlement.		

6) Quelle est la conclusion en une phrase ?

La tranche finalise presque entièrement `liveSession` pour les Prompt normaux, mais elle ne doit pas être déclarée terminée avant de corriger l'autorité observer de Delete, le latch de settlement et la réhydratation concurrente d'Activité.

7) Quelles corrections sont nécessaires ?

Lot A — Corriger uniquement le latch de settlement source

- Faire en sorte qu'un handle déjà promu soit définitivement consommé.
- Une annulation après promotion doit être un vrai no-op, sans résoudre le latch.
- Conserver l'annulation pré-promotion et la finalisation post-éviction.
- Ajouter un test prouvant que `waitForSourceSettlement()` reste bloqué entre promotion et finalisation.

- Cibles principales :
- [LiveSessionRegistry.service.ts](#)
- [LiveSessionRegistry.service.test.ts](#)
- [iaStreamHandlers.queryDoneSound.test.ts](#)

Lot B — Rendre la réhydratation Activity exacte et atomique

- Conserver `Running/observer/open` pour `sourceSettled=true` && `handoffReady=false`.
- Construire la prochaine projection hors store.
- Recapturer/revalider panneaux, sources, réservations et runtimes après les lectures persistantes.
- Remplacer `items` une seule fois, au lieu de vider puis reconstruire avec plusieurs broadcasts.
- Ajouter les tests :
- observer settled non-handoff ;
- source démarrant pendant une lecture persistante ;
- panneau ouvert/fermé pendant la reconstruction ;
- ancien handoff retenu + nouvelle réservation.

- Cibles principales :
- [SidebarStore.rehydrate.manager.ts](#)
- [SidebarStore.rehydrate.manager.test.ts](#)

Lot C — Sécuriser Delete avec l'autorité du registre

- Après acquisition du claim et avant fermeture du panneau, lire les snapshots exacts.
- Refuser Delete si un observer est réservé ou non handoff-ready, même si Activity est absente ou Idle.
- Continuer à abort/drainer seulement les sources normales `ownership='source'`.
- Ajouter un test avec store vide ou stale et observer actif dans le registre.

- Cibles principales :
- [delete.endpoint.ts](#)
- [delete.endpoint.liveSession.test.ts](#)

Lot D — Rétablir la stricte atomicité frontend

- Restaurer le garde exhaustif de toutes les clés de `AiQueryOptions`.
- Construire les plans `replace-style` complets avant `start()`.
- Ajouter un commit explicite du plan après acceptation.
- Conserver exactement le comportement actuel sur refus et sur attachements.

- Cibles principales :
- [usePromptRunCommands.hook.ts](#)
- [preparePromptStreamingTurn.util.ts](#)
- [usePromptRunCommands.startAcceptance.test.tsx](#)

Lot E — Toutes les corrections

Ce lot regroupe A, B, C et D, avec les tests de non-régression associés.

Conclusion Q7 : quatre lots ciblés sont disponibles, ainsi qu'un lot global recommandé avant de déclarer la tranche terminée.



Corriger le latch source



Corriger la réhydratation Activity



Sécuriser Delete face aux observers



Rétablir la stricte atomicité frontend



Corriger tous les écarts

</ASSISTANT>

<USER>

[Corriger tous les écarts]

</USER>

<ASSISTANT>

I'm thinking...

</ASSISTANT>

<ASSISTANT>



reading project: [liveSession.programming-doc-readme.md](#), [prompt-stream-dispatch-lock.programming-doc-readme.md](#), [fallback-persist-and-title.programming-doc-readme.md](#), [LiveSessionRecord.type.ts](#) [L1-

L280],  [LiveSessionRegistry.service.ts](#) [L1-L420],  [LiveSessionRegistry.service.test.ts](#) [L1-L230],

 [SidebarStore.rehydrate.manager.ts](#),
 [SidebarStore.rehydrate.manager.test.ts](#),
 [delete.endpoint.ts](#),
 [delete.endpoint.liveSession.test.ts](#),
 [preparePromptStreamingTurn.util.ts](#),
 [usePromptRunCommands.hook.ts](#),
 [usePromptRunCommands.startAcceptance.test.tsx](#),
 [preparePromptStreamingTurn.util.test.ts](#),  [AiQueryOptions.type.ts](#),  [SidebarStore.core.manager.ts](#) [L1-L215],

 [SidebarStore.queryWindow.listeners.manager.ts](#),
 [SidebarStore.queryWindow.mutations.manager.ts](#),  [LiveSessionRegistry.service.ts](#) [L420-L760],  [LiveSessionRegistry.service.ts](#) [L760-L1040],  [LiveSessionRegistry.service.ts](#) [L1040-L1200],  [usePromptRunCommands.retryQuery.attachmentOnly.test.tsx](#) [L1-L260],  [usePromptRunCommands.retryQuery.compactContextNow.test.tsx](#) [L1-L260],  [usePromptRunCommands.continueResponse.test.tsx](#) [L1-L260],  e ["Ancien garde options"](#)

  scanning git:  "Version garde avant ref"

  reading project:  r ["cancelSourceReservation"](#),  e ["Annulations réservations"](#),  e ["Annulations handler exact"](#),  [iaStreamHandlers.ts](#) [L1-L135],  [iaStreamHandlers.ts](#) [L220-L294],  r ["onQueryWindowTitleChanged"](#),  r ["onQueryWindowRunningChanged"](#),  e ["Réhydratation appels"](#),  [openPanel.util.ts](#) [L350-L470],  [openPanel.util.ts](#) [L70-L150],  r ["handleSubmitEdit"](#),  e ["Soumission Edit usage"](#),  e ["Flux appel Edit"](#),  [Prompt.tsx](#) [L1130-L1215],  r ["createPromptStreamingTurnPlan"](#),  e ["Utilisateurs plan Prompt"](#),  e ["Isomorphisme du plan"](#)

  patching file:  [LiveSessionRegistry.service.ts](#)

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

Copy

Review

Accept

PatchFull file

```
-         this.recordsByRunKey.set(runKey, record);
-
+         this.recordsByRunKey.set(runKey, record);
+         // Promotion permanently consumes the reservation capability. A later generic
cleanup call
+         // must not resolve the settlement latch; only post-forwarding finalization now
owns it.
+         this.reservationByHandle.delete(args.reservation);
+
-         if (this.reservationsByRunKey.get(runKey) !== reservation) {
-             // The exact owner may report cancellation after destructive eviction
removed its indexes.
-
+         if (this.reservationsByRunKey.get(runKey) !== reservation) {
+             // Promotion consumes the handle above, so this branch is reserved for an
unpromoted
+             // source whose registry indexes were removed by destructive eviction.
+             this.reservationByHandle.delete(handle);
+
-         }
-         this.reservationsByRunKey.delete(runKey);
-
+         }
+         this.reservationsByRunKey.delete(runKey);
+         this.reservationByHandle.delete(handle);
+

```

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

Copy
Review
Accept

  patching file:  [LiveSessionRegistry.service.test.ts](#)

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

Copy
Review
Accept

PatchFull file

```
-      await expect(cancelledWait).resolves.toBeUndefined();
-
+      await expect(cancelledWait).resolves.toBeUndefined();
+
+      const promotedRegistry = new LiveSessionRegistryService();
+      const promotedReservation = reserve(promotedRegistry, RUN_A);
+      promote(promotedRegistry, RUN_A, promotedReservation);
+      const promotedWait = promotedRegistry.waitForSourceSettlement(RUN_A);
+      let promotedSettled = false;
+      const observePromotedSettlement = async (): Promise<void> => {
+        // The observation task records the exact moment the source handler
relinquishes authority.
+        await promotedWait;
+        promotedSettled = true;
+      };
+      const promotedSettlementObservation = observePromotedSettlement();
+
+      // Generic handler cleanup runs before FIFO drainage, but promotion must make
cancellation inert.
+      expect(
+        promotedRegistry.cancelSourceReservation(promotedReservation),
+      ).toBe(false);
+      await Promise.resolve();
+      expect(promotedSettled).toBe(false);
+
+      promotedRegistry.appendChunk({
+        runRef: RUN_A,
+        chunk: terminal(RUN_A),
+      });
+      expect(promotedRegistry.finalizeSourceAfterForwarding(RUN_A)).toBe(
+        'handoffReady',
+      );
+      await promotedSettlementObservation;
+      expect(promotedSettled).toBe(true);
+
-
-      const evictedRegistry = new LiveSessionRegistryService();
-      reserve(evictedRegistry, RUN_A);
-
+      const evictedRegistry = new LiveSessionRegistryService();
+      const evictedReservation = reserve(evictedRegistry, RUN_A);
+
-
-      evictedRegistry.evictLocator(LOCATOR, claim);
-      expect(evictedRegistry.finalizeSourceAfterForwarding(RUN_A)).toBe(
-        'stale',
-      );
-
+      evictedRegistry.evictLocator(LOCATOR, claim);
+      // The unpromoted owner is still responsible for reporting post-eviction
cancellation.
+      expect(
```

```
+         evictedRegistry.cancelSourceReservation(evictedReservation),
+     ).toBe(false);
+
AICode/src/extension/src/aiTools/orch/liveSession/registry/\_test\_/LiveSessionRegistry.service.test.ts
Copy
Review
Accept
```



patching file: [SidebarStore.queryWindow.listeners.manager.ts](#)

```
AICode/src/extension/src/extension/panels/sidebar/SidebarStore.queryWindow.listeners.manager.ts
Copy
Review
Accept
PatchFull file
```

```
-     }
- }
-
+     }
+ }
+
+ /**
+  * Replace the complete Prompt Activity projection as one store mutation during
+  * rehydration.
+  *
+  * The caller must build this immutable snapshot from a synchronous authority capture.
+  * once prevents consumers from observing the previous implementation's transient
+  * empty/partial list.
+  */
+ export async function replaceQueryActivitiesForRehydration(
+     items: readonly SidebarActivityQueryItem[],
+ ): Promise<void> {
+     // Clone the completed projection before it crosses into the mutable Sidebar store
+     // boundary.
+     const nextItems = structuredClone([...items]);
+     const publication = updateSidebarStore((store) => {
+         store.sidebarContentState.activityState.queryState.items = nextItems;
+     });
+
+     // The mutation above is synchronous. Notify panel-title listeners immediately from
+     // the fully
+     // installed snapshot while the asynchronous Sidebar broadcast remains in flight.
+     for (const item of nextItems) {
+         const locator: ChatSessionLocator = {
+             sessionId: item.sessionId,
+             storageScopeRelPath: item.storageScopeRelPath,
+         };
+         emitSafely(titleListeners, { locator, title: item.title });
+         emitSafely(runningListeners, {
+             locator,
+             running: item.status === 'running',
+         });
+     }
+
+     await publication;
+ }
```

```
AICode/src/extension/src/extension/panels/sidebar/SidebarStore.queryWindow.listeners.manager.ts
Copy
Review
Accept
```

  patching file:  [SidebarStore.rehydrate.manager.ts](#)

[AICode/src/extension/src/extension/panels/sidebar/SidebarStore.rehydrate.manager.ts](#)

Error

([what is it?](#))

  patching file:  [SidebarStore.rehydrate.manager.ts](#)

[AICode/src/extension/src/extension/panels/sidebar/SidebarStore.rehydrate.manager.ts](#)

Copy

Review

Accept

PatchFull file

```
- import { getLiveSessionServiceForProd } from
'../../../../../aiTools/orch/liveSession/core/LiveSessionService.js';
-
+ import { getLiveSessionServiceForProd } from
'../../../../../aiTools/orch/liveSession/core/LiveSessionService.js';
+ import type {
+   LiveSessionActiveRunSnapshot,
+   LiveSessionSourceReservationSnapshot,
+ } from ' ../../../../../aiTools/orch/liveSession/registry/LiveSessionRecord.type.js';
+
-   isSameQueryActivityLocator,
-   panelOpened,
-
+   isSameQueryActivityLocator,
+   replaceQueryActivitiesForRehydration,
+
-
- /** Return one finite non-negative millisecond value or undefined. */
-
+ /** Registry projection that may contribute one authoritative Prompt Activity row. */
+ type LiveSessionActivitySnapshot =
+   | LiveSessionSourceReservationSnapshot
+   | LiveSessionActiveRunSnapshot;
+
+ /** Return one finite non-negative millisecond value or undefined. */
+
-
- /**
-  * Reconcile one now-closed Prompt activity after a failed destructive operation
releases its claim.
-
+ /** Capture every currently open Prompt panel without retaining mutable panel objects.
*/
+ function captureOpenPromptActivities(): Map<
+   string,
+   { locator: ChatSessionLocator; title: string }
+ > {
+   const openLocators = new Map<
+     string,
+     { locator: ChatSessionLocator; title: string }
+   >();
+   openedPanelsRegistry.forEach((key, panel) => {
+     // Only session-bound Prompt panels participate in the Activity projection.
+     const parsed = parsePromptPanelKey(key);
+     if (!parsed || parsed.kind !== 'session') return;
```

```

+         const title =
+             stripPromptTabTitlePrefix(String(panel.title ?? '')).trim() ||
+             getVisibleChatTitleOrUntitled(parsed.locator);
+         openLocators.set(locatorKey(parsed.locator), {
+             locator: parsed.locator,
+             title,
+         });
+     });
+     return openLocators;
+ }
+
+ /** Return whether one registry snapshot still owns a visible running projection. */
+ function isProjectableLiveSnapshot(
+     snapshot: LiveSessionActivitySnapshot,
+     openLocators: ReadonlyMap<
+         string,
+         { locator: ChatSessionLocator; title: string }
+     >,
+ ): boolean {
+     // Handoff-ready records no longer represent active Activity work. A source-owned
+     // record cannot
+     // expose a stable sourceSettled/non-handoff intermediate because its finalization
+     // is synchronous.
+     if (snapshot.kind === 'record' && snapshot.handoffReady) return false;
+
+     // Observer work is meaningful only through its exact still-open Prompt viewer.
+     // Importantly,
+     // sourceSettled observers remain Running until the later EasyAgents business
+     // handoff completes.
+     return !(
+         snapshot.ownership === 'observer' &&
+         !openLocators.has(locatorKey(snapshot.runRef.locator))
+     );
+ }
+
+ /** Collect the locators whose presentation metadata is required by one authority
+     capture. */
+ function buildActivityProjectionTargets(
+     openLocators: ReadonlyMap<
+         string,
+         { locator: ChatSessionLocator; title: string }
+     >,
+     snapshots: readonly LiveSessionActivitySnapshot[],
+ ): Map<string, { locator: ChatSessionLocator; runRef?: LiveSessionRunRef }> {
+     const targets = new Map<
+         string,
+         { locator: ChatSessionLocator; runRef?: LiveSessionRunRef }
+     >();
+     for (const open of openLocators.values()) {
+         targets.set(locatorKey(open.locator), { locator: open.locator });
+     }
+     for (const snapshot of snapshots) {
+         // Retained terminal records must not overwrite the identity of a newly active
+         // source on the same locator.
+         if (!isProjectableLiveSnapshot(snapshot, openLocators)) continue;
+         targets.set(locatorKey(snapshot.runRef.locator), {
+             locator: snapshot.runRef.locator,
+             runRef: snapshot.runRef,
+         });
+     }
+     return targets;
+ }
+
+

```

```

+ /** Resolve stable UI metadata for one locator from the documented priority chain. */
+ function resolveActivityMetadata(args: {
+   target: { locator: ChatSessionLocator; runRef?: LiveSessionRunRef };
+   openTitle?: string;
+   previous?: SidebarActivityQueryItem;
+   persisted: PersistedActivityMetadata;
+   rehydratedAtMs: number;
+ }): ResolvedActivityMetadata {
+   const runtime = sidebarQueryRunningBridge.getSnapshot(args.target.locator);
+   const runtimeMatchesTarget = isSameRunRef(
+     runtime?.runRef,
+     args.target.runRef,
+   );
+   const previousRunRef =
+     args.previous?.status === 'running'
+       ? args.previous.runRef
+       : undefined;
+   const previousMatchesTarget = isSameRunRef(
+     previousRunRef,
+     args.target.runRef,
+   );
+   // Runtime and the current store preserve exact live values; panel and disk remain
+   // presentation fallbacks only.
+   const title =
+     runtime?.title?.trim() ||
+     args.previous?.title.trim() ||
+     args.openTitle?.trim() ||
+     args.persisted.title?.trim() ||
+     getVisibleChatTitleOrUntitled(args.target.locator);
+   const openedAt =
+     normalizeActivityTimestamp(runtime?.openedAt) ??
+     normalizeActivityTimestamp(args.previous?.openedAt) ??
+     normalizeActivityTimestamp(args.persisted.openedAt) ??
+     args.rehydratedAtMs;
+   const usage = args.target.runRef
+     ? runtimeMatchesTarget
+       ? runtime?.usage
+       : previousMatchesTarget
+         ? args.previous?.usage
+         : undefined
+     : (runtime?.usage ?? args.previous?.usage);
+   const liveQueryElapsedMs = args.target.runRef
+     ? runtimeMatchesTarget
+       ? runtime?.liveQueryElapsedMs
+       : previousMatchesTarget && args.previous?.status === 'running'
+         ? args.previous.liveQueryElapsedMs
+         : undefined
+     : undefined;
+   const lastQueryElapsedMs = args.target.runRef
+     ? runtimeMatchesTarget
+       ? runtime?.lastQueryElapsedMs
+       : previousMatchesTarget
+         ? args.previous?.lastQueryElapsedMs
+         : undefined
+     : (runtime?.lastQueryElapsedMs ??
+       args.previous?.lastQueryElapsedMs ??
+       (args.previous?.status === 'running'
+         ? args.previous.liveQueryElapsedMs
+         : undefined));
+   return {

```

```
+ title,  
+ openedAt,  
+ ...(liveQueryElapsedMs === undefined ? {} : { liveQueryElapsedMs })),  
+ ...(lastQueryElapsedMs === undefined ? {} : { lastQueryElapsedMs })),  
+ ...(usage === undefined ? {} : { usage })),  
+ };  
+ }  
+  
+ /**  
+ * Reconcile one now-closed Prompt activity after a failed destructive operation  
releases its claim.  
+  


---

- export async function rehydrateQueryActivities(): Promise<void> {  
    const openLocators = new Map<  
  
+ export async function rehydrateQueryActivities(): Promise<void> {  
+     const persistence = new ChatPersistence();  
+     const persistedMetadataByLocatorKey = new Map<  
+  


---

-     const persistedMetadataByLocatorKey = new Map<  
-         string,  
-         { locator: ChatSessionLocator; title: string }  
-  
+     const persistedMetadataByLocatorKey = new Map<  
+         string,  
+         PersistedActivityMetadata  
+  


---

- >());  
- openPanelsRegistry.forEach((key, panel) => {  
-     const parsed = parsePromptPanelKey(key);  
-     if (!parsed || parsed.kind !== 'session') return;  
-     const title =  
-         stripPromptTabTitlePrefix(String(panel.title ?? '')).trim() ||  
-         getVisibleChatTitleOrUntitled(parsed.locator);  
-     openLocators.set(locatorKey(parsed.locator), {  
-         locator: parsed.locator,  
-         title,  
-     });  
- });  
- >());  
+  


---

-  
- // Snapshot every authority before clearing the projection. Runtime getSnapshot  
intentionally reads  
- // the current item to preserve stable title, openedAt, usage, and elapsed values  
across Sidebar reloads.  
- const snapshots = getSessionServiceForProd().listActiveSnapshots();  
- const previousItemsByLocatorKey = new Map(  
-     sidebarStore.sidebarContentState.activityState.queryState.items.map(  
-         (item) => [locatorKey(item), item] as const,  
-     ),  
- );  
- const projectionTargets = new Map<  
-     string,  
-     { locator: ChatSessionLocator; runRef?: LiveSessionRunRef }  
- >());  
- for (const open of openLocators.values()) {  
-     projectionTargets.set(locatorKey(open.locator), {  
-         locator: open.locator,  
-     });  
- }
```

```

-     }
-     for (const snapshot of snapshots) {
-         projectionTargets.set(locatorKey(snapshot.runRef.locator), {
-             locator: snapshot.runRef.locator,
-             runRef: snapshot.runRef,
-         });
-     }
-
+
+     while (true) {
+         // Every loop starts from fresh synchronous authorities. Any source/panel that
+         // changed while
+         // persistence was awaited is therefore reflected before the single atomic
+         // store replacement.
+         const openLocators = captureOpenPromptActivities();
+         const snapshots = getLiveSessionServiceForProd().listActiveSnapshots();
+         const projectionTargets = buildActivityProjectionTargets(
+             openLocators,
+             snapshots,
+         );
+         const missingPersistedTargets = [...projectionTargets.entries()].filter(
+             ([key]) => !persistedMetadataByLocatorKey.has(key),
+         );
+         if (missingPersistedTargets.length > 0) {
+             // Persistence is presentation-only. Cache each completed lookup, then
+             // restart from live
+             // authorities instead of applying the now-stale pre-await panel/registry
+             // snapshot.
+             await Promise.all(
+                 missingPersistedTargets.map(
+                     async ([key, target]): Promise<void> => {
+                         persistedMetadataByLocatorKey.set(
+                             key,
+                             await readPersistedActivityMetadata(
+                                 persistence,
+                                 target.locator,
+                             ),
+                         );
+                     },
+                 ),
+             );
+             continue;
+         }
+     }
+
-
-     const rehydratedAtMs = Date.now();
-     const persistence = new ChatPersistence();
-     const metadataByLocatorKey = new Map<string, ResolvedActivityMetadata>();
-
+
+     // No await occurs between this final capture and the store mutation below.
+     // JavaScript event
+     // ordering therefore prevents a concurrent source/panel transition from being
+     // overwritten.
+     const previousItemsByLocatorKey = new Map(
+         sidebarStore.sidebarContentState.activityState.queryState.items.map(
+             (item) => [locatorKey(item), item] as const,
+         ),
+     );
+     const rehydratedAtMs = Date.now();
+     const metadataByLocatorKey = new Map<
+         string,
+         ResolvedActivityMetadata

```

```

+ >());
+ for (const [key, target] of projectionTargets) {
+   metadataByLocatorKey.set(
+     key,
+     resolveActivityMetadata({
+       target,
+       openTitle: openLocators.get(key)?.title,
+       previous: previousItemsByLocatorKey.get(key),
+       persisted: persistedMetadataByLocatorKey.get(key) ?? {},
+       rehydratedAtMs,
+     })),
+   );
+ }
+
-
- // Resolve the explicit priority chain before mutating the store:
- // runtime → previous store → open panel → persistence → deterministic default.
- await Promise.all(
-   [...projectionTargets.entries()].map(
-     async ([key, target]): Promise<void> => {
-       const runtime = sidebarQueryRunningBridge.getSnapshot(
-         target.locator,
-
+
+ const nextItems: SidebarActivityQueryItem[] = [];
+ const projected = new Set<string>();
+ for (const snapshot of snapshots) {
+   if (!isProjectableLiveSnapshot(snapshot, openLocators)) continue;
+   const key = locatorKey(snapshot.runRef.locator);
+   if (projected.has(key)) continue;
+   const open = openLocators.get(key);
+   const metadata = metadataByLocatorKey.get(key);
+   if (!metadata) {
+     throw new Error(
+       'Prompt Activity metadata is missing for an authoritative live-
session snapshot.',
+
-
-   );
-   const previous = previousItemsByLocatorKey.get(key);
-   const open = openLocators.get(key);
-   const persisted = await readPersistedActivityMetadata(
-     persistence,
-     target.locator,
-
+
+   );
+ }
+ const runtime = sidebarQueryRunningBridge.getSnapshot(
+   snapshot.runRef.locator,
+
+ );
+ const runtimeMatchesSnapshot = isSameRunRef(
+   runtime?.runRef,
+   snapshot.runRef,
+
+ );
+ const common = {
+   status: 'running' as const,
+   sessionId: snapshot.runRef.locator.sessionId,
+   storageScopeRelPath:
+     snapshot.runRef.locator.storageScopeRelPath,
+   title: metadata.title,
+   runRef: snapshot.runRef,
+   liveQueryElapsedMs:
+     metadata.liveQueryElapsedMs ??

```

```

+       Math.max(0, rehydratedAtMs - snapshot.sourceStartedAtMs),
+       ...(metadata.lastQueryElapsedMs === undefined
+         ? {}
+         : { lastQueryElapsedMs: metadata.lastQueryElapsedMs })),
+       ...(metadata.usage === undefined
+         ? {}
+         : { usage: metadata.usage })),
+       updatedAt: rehydratedAtMs,
+       openedAt: metadata.openedAt,
+     };
+     const item: SidebarActivityQueryItem =
+       snapshot.ownership === 'source'
+         ? {
+             ...common,
+             ownership: 'source',
+             panelState: open ? 'open' : 'closed',
+             stopRequested:
+               snapshot.abortRequested ||
+               (runtimeMatchesSnapshot &&
+                 runtime?.stopRequested === true),
+           }
+         : {
+             ...common,
+             ownership: 'observer',
+             panelState: 'open',
+           };
+     nextItems.push(item);
+     projected.add(key);
+   }
+
+   for (const [key, open] of openLocators) {
+     if (projected.has(key)) continue;
+     const metadata = metadataByLocatorKey.get(key);
+     if (!metadata) {
+       throw new Error(
+         'Prompt Activity metadata is missing for an open Prompt panel.',
+       );
+     }
+     const runtimeMatchesTarget = isSameRunRef(
+       runtime?.runRef,
+       target.runRef,
+     );
+     const previousRunRef =
+       previous?.status === 'running'
+         ? previous.runRef
+         : undefined;
+     const previousMatchesTarget = isSameRunRef(
+       previousRunRef,
+       target.runRef,
+     );
+
+     const title =
+       runtime?.title?.trim() ||
+       previous?.title?.trim() ||
+       open?.title?.trim() ||
+       persisted.title?.trim() ||
+       getVisibleChatTitleOrUntitled(target.locator);
+     const openedAt =
+       normalizeActivityTimestamp(runtime?.openedAt) ??
+       normalizeActivityTimestamp(previous?.openedAt) ??
+       normalizeActivityTimestamp(persisted.openedAt) ??
+       rehydratedAtMs;

```

```

-         const usage = target.runRef
-             ? runtimeMatchesTarget
-                 ? runtime?.usage
-                 : previousMatchesTarget
-                     ? previous?.usage
-                     : undefined
-             : (runtime?.usage ?? previous?.usage);
-         const liveQueryElapsedMs = target.runRef
-             ? runtimeMatchesTarget
-                 ? runtime?.liveQueryElapsedMs
-                 : previousMatchesTarget &&
-                     previous?.status === 'running'
-                     ? previous.liveQueryElapsedMs
-                     : undefined
-             : undefined;
-         const lastQueryElapsedMs = target.runRef
-             ? runtimeMatchesTarget
-                 ? runtime?.lastQueryElapsedMs
-                 : previousMatchesTarget
-                     ? previous?.lastQueryElapsedMs
-                     : undefined
-             : (runtime?.lastQueryElapsedMs ??
-                 previous?.lastQueryElapsedMs ??
-                 (previous?.status === 'running'
-                     ? previous.liveQueryElapsedMs
-                     : undefined));
-
-         metadataByLocatorKey.set(key, {
-             title,
-             openedAt,
-             ...(liveQueryElapsedMs === undefined
-                 ? {}
-                 : { liveQueryElapsedMs }),
-             ...(lastQueryElapsedMs === undefined
-                 ? {}
-                 : { lastQueryElapsedMs }),
-             ...(usage === undefined ? {} : { usage }),
-         });
-     },
- );
- );
-
- await updateSidebarStore((store) => {
-     store.sidebarContentState.activityState.queryState.items = [];
- });
-
- const projected = new Set<string>();
- for (const snapshot of snapshots) {
-     if (
-         snapshot.kind === 'record' &&
-         (snapshot.sourceSettled || snapshot.handoffReady)
-     ) {
-         continue;
-     }
-
-     nextItems.push({
-         status: 'idle',
-         panelState: 'open',
-         sessionId: open.locator.sessionId,
-         storageScopeRelPath: open.locator.storageScopeRelPath,
-         title: metadata.title,
-         ...(metadata.lastQueryElapsedMs === undefined

```

```

+         ? {}
+         : { lastQueryElapsedMs: metadata.lastQueryElapsedMs }},
+     ...(metadata.usage === undefined
+         ? {}
+         : { usage: metadata.usage })),
+     updatedAt: metadata.openedAt,
+     openedAt: metadata.openedAt,
+ });
+
-     }
-     const key = locatorKey(snapshot.runRef.locator);
-     const open = openLocators.get(key);
-     // Reservations retain their real ownership so observer-only sources can never
become detached source rows.
-     const ownership = snapshot.ownership;
-     if (ownership === 'observer' && !open) continue;
-     const runtime = sidebarQueryRunningBridge.getSnapshot(
-         snapshot.runRef.locator,
-     );
-     const metadata = metadataByLocatorKey.get(key);
-     const runtimeMatchesSnapshot = isSameRunRef(
-         runtime?.runRef,
-         snapshot.runRef,
-     );
-     await sourceStarted({
-         runRef: snapshot.runRef,
-         ownership,
-         panelState: open ? 'open' : 'closed',
-         sourceStartedAtMs: snapshot.sourceStartedAtMs,
-         title: metadata?.title,
-         stopRequested:
-             snapshot.abortRequested ||
-             (runtimeMatchesSnapshot && runtime?.stopRequested === true),
-         openedAt: metadata?.openedAt,
-         liveQueryElapsedMs: metadata?.liveQueryElapsedMs,
-         lastQueryElapsedMs: metadata?.lastQueryElapsedMs,
-         usage: metadata?.usage,
-     });
-     projected.add(key);
- }
+
+ }
+
-
- for (const [key, open] of openLocators) {
-     if (projected.has(key)) continue;
-     const metadata = metadataByLocatorKey.get(key);
-     await panelOpened({
-         locator: open.locator,
-         title: metadata?.title,
-         openedAt: metadata?.openedAt ?? rehydratedAtMs,
-         lastQueryElapsedMs: metadata?.lastQueryElapsedMs,
-         usage: metadata?.usage,
-     });
-
+     await replaceQueryActivitiesForRehydration(nextItems);
+     return;
+

```

[AICode/src/extension/src/extension/panels/sidebar/SidebarStore.rehydrate.manager.ts](#)

Copy

Review

Accept

  patching file:  [SidebarStore.rehydrate.manager.test.ts](#)
[AI/Code/src/extension/src/extension/panels/sidebar/_test_/SidebarStore.rehydrate.manager.test.ts](#)
Error
([what is it?](#))

  reading project:  [SidebarStore.rehydrate.manager.test.ts](#) [L45-L85],  [SidebarStore.rehydrate.manager.test.ts](#) [L340-L500]

  patching file:  [SidebarStore.rehydrate.manager.test.ts](#)
[AI/Code/src/extension/src/extension/panels/sidebar/_test_/SidebarStore.rehydrate.manager.test.ts](#)

Copy

Review

Accept

PatchFull file

```
-     return { title, active: false } as unknown as vscode.WebviewPanel;
- }
-
+     return { title, active: false } as unknown as vscode.WebviewPanel;
+ }
+
+ /** Controllable promise used to exercise authority changes during persistence reads.
+ */
+ function createVoidDeferred(): {
+     readonly promise: Promise<void>;
+     readonly resolve: () => void;
+ } {
+     let resolvePromise: (() => void) | undefined;
+     const promise = new Promise<void>((resolve) => {
+         resolvePromise = () => resolve();
+     });
+     if (!resolvePromise) {
+         throw new Error('Deferred resolver was not initialized.');
```

```
-     });
-
-     it('never reconstructs observer ownership after its Prompt panel closed', async ()
=> {
-
+     });
+
+     it('keeps a source-settled observer Running until its later business handoff
completes', async () => {
+         const locator = {
+             sessionId: '6'.repeat(32),
+             storageScopeRelPath: 'batch/chat',
+         };
+         const runRef = {
+             locator,
+             turnId: 'turn-observer-settled',
+             sourceId: 'turn-observer-settled',
+         };
+         openedPanelsRegistry.set(
+             buildPromptPanelKey(locator),
+             panel(
```

```

+         `${ChatConfigConst.CONFIG_PROMPT_TAB_TITLE_PREFIX_RUNNING}Watching
batch`,
+     ),
+ );
+ listActiveSnapshotsMock.mockReturnValue([
+     {
+         kind: 'record',
+         runRef,
+         ownership: 'observer',
+         sourceStartedAtMs: 5,
+         abortRequested: false,
+         sourceSettled: true,
+         handoffReady: false,
+     },
+ ]);
+
+ await rehydrateQueryActivities();
+
+ expect(
+     sidebarStore.sidebarContentState.activityState.queryState.items,
+ ).toEqual([
+     expect.objectContaining({
+         runRef,
+         status: 'running',
+         ownership: 'observer',
+         panelState: 'open',
+     }),
+ ]);
+ });
+
+ it('recaptures panel and source authorities after an awaited persistence lookup',
+ async () => {
+     const staleLocator = {
+         sessionId: '7'.repeat(32),
+         storageScopeRelPath: '',
+     };
+     const freshLocator = {
+         sessionId: '8'.repeat(32),
+         storageScopeRelPath: 'chat',
+     };
+     const freshRunRef = {
+         locator: freshLocator,
+         turnId: 'turn-fresh',
+         sourceId: 'source-fresh',
+     };
+     const firstReadStarted = createVoidDeferred();
+     const releaseFirstRead = createVoidDeferred();
+     openedPanelsRegistry.set(
+         buildPromptPanelKey(staleLocator),
+         panel(
+             `${ChatConfigConst.CONFIG_PROMPT_TAB_TITLE_PREFIX_IDLE}Stale panel`,
+         ),
+     );
+     loadSessionFileMock.mockImplementation(async (locator) => {
+         if (locator.sessionId === staleLocator.sessionId) {
+             // Hold the first metadata read while the authoritative panel/source
set changes.
+             firstReadStarted.resolve();
+             await releaseFirstRead.promise;
+         }
+         return null;
+     });
+
+

```

```

+         const rehydration = rehydrateQueryActivities();
+         await firstReadStarted.promise;
+
+         // The old panel disappears and an unrelated source starts while persistence is
+         still pending.
+         openedPanelsRegistry.delete(buildPromptPanelKey(staleLocator));
+         listActiveSnapshotsMock.mockReturnValue([
+             {
+                 kind: 'reservation',
+                 runRef: freshRunRef,
+                 ownership: 'source',
+                 sourceStartedAtMs: 10,
+                 abortRequested: false,
+             },
+         ]);
+         releaseFirstRead.resolve();
+         await rehydration;
+
+         expect(
+             sidebarStore.sidebarContentState.activityState.queryState.items,
+         ).toEqual([
+             expect.objectContaining({
+                 runRef: freshRunRef,
+                 status: 'running',
+                 ownership: 'source',
+                 panelState: 'closed',
+             }),
+         ]);
+         expect(
+             sidebarStore.sidebarContentState.activityState.queryState.items.some(
+                 (item) => item.sessionId === staleLocator.sessionId,
+             ),
+         ).toBe(false);
+     });
+
+     it('never reconstructs observer ownership after its Prompt panel closed', async ()
=> {
+

```

[AI Code/src/extension/src/extension/panels/sidebar/ test /SidebarStore.rehydrate.manager.test.ts](#)

Copy

Review

Accept

  patching file:  [delete.endpoint.ts](#)

[AI Code/src/extension/src/slimbk-local/endpoints/ai/orch/chat/delete.endpoint.ts](#)

Copy

Review

Accept

PatchFull file

```

- import { forceRemoveQueryActivity } from
+ import { forceRemoveQueryActivity } from
'../../../../../../../../extension/panels/sidebar/SidebarStore.queryWindow.mutations.manager.js';
- import {
-     sidebarStore,
-     updateSidebarStore,
- } from '../../../../../../../../extension/panels/sidebar/SidebarStore.core.manager.js';
+ import { forceRemoveQueryActivity } from
+ import { updateSidebarStore } from
'../../../../../../../../extension/panels/sidebar/SidebarStore.queryWindow.mutations.manager.js';
+ import { updateSidebarStore } from
'../../../../../../../../extension/panels/sidebar/SidebarStore.core.manager.js';
+

```

```

- const SOURCE_SETTLEMENT_TIMEOUT_MS = 5_000;
-
+ const SOURCE_SETTLEMENT_TIMEOUT_MS = 5_000;
+
+ /** Expected business rejection raised when an observer still owns the exact live-
+ session locator. */
+ class ChatDeleteBlockedByObserverError extends Error {
+     public constructor() {
+         super('ChatDeleteBlocked: query is running');
+         this.name = 'ChatDeleteBlockedByObserverError';
+     }
+ }
+

```

```

-
-     // Consult the authoritative store for the running flag of this session.
-     const activities =
-         sidebarStore.sidebarContentState.activityState.queryState.items;
-     const activity = activities.find(
-         (w) =>
-             w.sessionId === locator.sessionId &&
-             w.storageScopeRelPath === locator.storageScopeRelPath,
-     );
-     if (
-         activity?.status === 'running' &&
-         activity.ownership === 'observer'
-     ) {
-         // Display a VS Code system error and abort the operation.
-         void VsCode.window.showErrorMessage(
-             'Cannot delete while a query is running. Stop it first.',
-         );
-         throw new Error('ChatDeleteBlocked: query is running');
-     }
-
+

```

```

-     try {
-         const activeSourceRunRefs = liveSession
+
+         try {
+             // Re-read the process-local authority after claim acquisition. Sidebar
+             Activity is only a
+             // UI projection and may be missing or stale while an EasyAgents observer
+             still converges.
+             const activeSnapshots = liveSession
+

```

```

-             snapshot.runRef.locator.storageScopeRelPath ===
-             locator.storageScopeRelPath &&
-
+             snapshot.runRef.locator.storageScopeRelPath ===
+             locator.storageScopeRelPath,
+
+         );
+         const hasActiveObserver = activeSnapshots.some(
+             (snapshot) =>
+                 snapshot.ownership === 'observer' &&
+                 (snapshot.kind === 'reservation' ||
+                 !snapshot.handoffReady),
+         );
+         if (hasActiveObserver) {
+             // Observer-owned sources have product-specific settlement and must be
+             stopped explicitly.
+             throw new ChatDeleteBlockedByObserverError();

```

```

+         }
+         const activeSourceRunRefs = activeSnapshots
+           .filter(
+             (snapshot) =>
+
+             (snapshot.kind === 'reservation' ||
+              (snapshot.kind === 'record' &&
+               !snapshot.sourceSettled &&
+               !snapshot.handoffReady)),
+
+             (snapshot.kind === 'reservation' ||
+              !snapshot.sourceSettled),
+
+       } catch (err) {
+
+       } catch (err) {
+         if (err instanceof ChatDeleteBlockedByObserverError) {
+           // This is an expected operator action, not a filesystem/system
failure.
+           void VsCode.window.showErrorMessage(
+             'Cannot delete while a query is running. Stop it first.',
+           );
+           throw err;
+         }
+       }

```

[AICode/src/extension/src/slimbk-local/endpoints/ai/orch/chat/delete.endpoint.ts](#)

Copy

Review

Accept

  *patching file:*  [delete.endpoint.liveSession.test.ts](#)

[AICode/src/extension/src/slimbk-local/endpoints/ai/orch/chat/ test /delete.endpoint.liveSession.test.ts](#)

Copy

Review

Accept

PatchFull file

```
- } as const;
```

```
-
+ } as const;
+ const observerRunRef = {
+   ...runRef,
+   sourceId: runRef.turnId,
+ } as const;
+
+ 
```

```
-      it('blocks deletion from an observer-running Activity projection before acquiring
the claim', async () => {
```

```
+      it('blocks deletion from observer authority while an Activity projection is also
present', async () => {
```

```
-      panelState: 'open',
-      runRef,
-
+      panelState: 'open',
+      runRef: observerRunRef,
```

```

- ];
-
- await expect(
-
+ ];
+ listActiveSnapshotsMock.mockReturnValue([
+   {
+     kind: 'reservation',
+     runRef: observerRunRef,
+     viewerGroupId: 'easyAgents:batch',
+     ownership: 'observer',
+     replayMode: 'replaceTurnAssistants',
+     sourceStartedAtMs: 1,
+     abortRequested: false,
+     locatorEpoch: 1,
+   },
+ ]);
+
+ await expect(
+
- );
- expect(acquireMock).not.toHaveBeenCalled();
- expect(order).toEqual([]);
-
+ );
+ expect(acquireMock).toHaveBeenCalledWith(runRef.locator);
+ expect(releaseMock).toHaveBeenCalledWith(claim);
+ expect(order).toEqual(['release']);
+ });
+
+ it('blocks deletion from authoritative observer state even when Activity is
missing', async () => {
+   listActiveSnapshotsMock.mockReturnValue([
+     {
+       kind: 'record',
+       runRef: observerRunRef,
+       viewerGroupId: 'easyAgents:batch',
+       ownership: 'observer',
+       replayMode: 'replaceTurnAssistants',
+       sourceStartedAtMs: 1,
+       baselineReady: true,
+       sourceSettled: true,
+       handoffReady: false,
+       hasTerminalMeta: true,
+       abortRequested: false,
+       locatorEpoch: 1,
+     },
+   ],
+ );
+
+   await expect(
+     endpoint.syncHandler({
+       args: runRef.locator,
+     } as Parameters<typeof endpoint.syncHandler>[0]),
+   ).rejects.toThrow('query is running');
+
+   expect(acquireMock).toHaveBeenCalledWith(runRef.locator);
+   expect(releaseMock).toHaveBeenCalledWith(claim);
+   expect(showErrorMessageMock).toHaveBeenCalledWith(
+     'Cannot delete while a query is running. Stop it first.',
+   );
+   expect(deleteMock).not.toHaveBeenCalled();
+   expect(evictMock).not.toHaveBeenCalled();
+   expect(forceRemoveActivityMock).not.toHaveBeenCalled();

```

```

+         expect(reconcileActivityMock).not.toHaveBeenCalled();
+         expect(order).toEqual(['release']);
+
-
-     it('aggregates durable deletion and post-release Activity reconciliation failures
in deterministic order', async () => {
-
+
+     it('treats a stale observer Activity row as non-authoritative when the registry has
no observer', async () => {
+         listActiveSnapshotsMock.mockReturnValue([]);
+         sidebarStore.sidebarContentState.activityState.queryState.items = [
+             {
+                 status: 'running',
+                 ownership: 'observer',
+                 panelState: 'open',
+                 runRef: observerRunRef,
+                 sessionId: runRef.locator.sessionId,
+                 storageScopeRelPath: runRef.locator.storageScopeRelPath,
+                 title: 'Stale observer',
+                 liveQueryElapsedMs: 1,
+                 updatedAt: 1,
+                 openedAt: 1,
+             },
+         ];
+
+         await endpoint.syncHandler({
+             args: runRef.locator,
+         } as Parameters<typeof endpoint.syncHandler>[0]);
+
+         expect(order).toEqual([
+             'close',
+             'removeActivity',
+             'delete',
+             'evict',
+             'release',
+         ]);
+         expect(showErrorMessageMock).not.toHaveBeenCalled();
+     });
+
+     it('aggregates durable deletion and post-release Activity reconciliation failures
in deterministic order', async () => {
+

```

[AI Code/src/extension/src/slimbk-local/endpoints/ai/orch/chat/_test_/delete.endpoint.liveSession.test.ts](#)

Copy

Review

Accept

  **patching file:**  [preparePromptStreamingTurn.util.ts](#)

[AI Code/src/frontend/src/app/prompt/Prompt/preparePromptStreamingTurn.util.ts](#)

Copy

Review

Accept

PatchFull file

```

-
- /** Pure replace-style plan created before any Prompt runtime mutation. */
-
+
+ /** Pure immutable message plan created before any Prompt runtime mutation. */
+

```

```

-
- /** Allocate targets and caller-defined scaffold without touching React state or
mutable refs. */
-
+
+ /** Message-axis mutation selected explicitly by the caller when a prepared plan is
committed. */
+ export type PromptStreamingTurnMessageCommitMode = 'append' | 'replaceAll';
+
+ /** Allocate targets and caller-defined scaffold without touching React state or
mutable refs. */
+


---


- /**
-  * Prime the Prompt runtime for a brand-new streamed turn by reusing the exact same
contract for every entry path.
-
+ /**
+  * Commit one already-complete Prompt streaming plan after synchronous transport
acceptance.
+


---


- *
- * Responsibilities:
- * - clear the append-mode prefix so the new turn always starts from an empty assistant
buffer,
- * - register the stable turn/placeholder identities before any stream event arrives,
- * - append the caller-provided UI messages (user bubble when needed + canonical
assistant placeholders),
- * - reset every ephemeral streaming buffer and idempotence guard so old state never
leaks into the new turn.
- *
- * Why this helper exists:
- * - The normal Prompt send path, edit replay path, and EasyAgents live watcher must
all respect the exact same
- *   streaming bootstrap contract.
- * - Centralizing that contract keeps the live watcher attached to the existing Prompt
architecture instead of
- *   recreating a second pseudo-streaming pipeline.
-
+ *
+ * This function never allocates identities or calls a scaffold builder. Every
potentially failing
+ * preparation step therefore happens before `start()`, while all visible mutations
remain after it.
+


---


- */
- export function preparePromptStreamingTurn(args: {
-   turnId: string;
- }
+ */
+ export function commitPromptStreamingTurn(args: {
+   plan: PromptStreamingTurnPlan;
+   messageCommitMode?: PromptStreamingTurnMessageCommitMode;
+ }


---


-   setMessages: Dispatch<SetStateAction<PromptMessage[]>>;
-   buildMessages: (
-     targets: PromptStreamingTurnTargets,
-   ) => readonly PromptMessage[];
-
+   setMessages: Dispatch<SetStateAction<PromptMessage[]>>;
+


---



```

```

- }): PromptStreamingTurnTargets {
-   const plan = createPromptStreamingTurnPlan({
-     turnId: args.turnId,
-     buildMessages: args.buildMessages,
-   });
-
+ }): PromptStreamingTurnTargets {
+   const { plan } = args;
+


---


-   // The current turn id must be visible before the first thinking/message token is
processed.
-   args.currentTurnIdRef.current = args.turnId;
-
+   // The current turn id must be visible before the first thinking/message token is
processed.
+   args.currentTurnIdRef.current = targets.turnId;
+


---


-
-   // The structural UI shell for the streamed turn must exist before transport events
reach the Prompt.
-
+
+   // Install exactly the immutable message plan prepared before transport acceptance.
+


---


-   args.setMessages((previousMessages) => {
-     return [...previousMessages, ...plan.messages];
-
+   args.setMessages((previousMessages) => {
+     return args.messageCommitMode === 'replaceAll'
+       ? [...plan.messages]
+       : [...previousMessages, ...plan.messages];
+


---


-   return targets;
- }
-
+   return targets;
+ }
+
+ /**
+  * Prime the Prompt runtime for a brand-new streamed turn by reusing the exact same
contract for every entry path.
+  *
+  * Responsibilities:
+  * - clear the append-mode prefix so the new turn always starts from an empty assistant
buffer,
+  * - register the stable turn/placeholder identities before any stream event arrives,
+  * - append the caller-provided UI messages (user bubble when needed + canonical
assistant placeholders),
+  * - reset every ephemeral streaming buffer and idempotence guard so old state never
leaks into the new turn.
+  *
+  * Why this helper exists:
+  * - The normal Prompt send path, edit replay path, and EasyAgents live watcher must
all respect the exact same
+  *   streaming bootstrap contract.
+  * - Centralizing that contract keeps the live watcher attached to the existing Prompt
architecture instead of
+  *   recreating a second pseudo-streaming pipeline.
+  */
+ export function preparePromptStreamingTurn(args: {
+   turnId: string;

```

```

+   setMessages: Dispatch<SetStateAction<PromptMessage[]>>;
+   buildMessages: (
+     targets: PromptStreamingTurnTargets,
+   ) => readonly PromptMessage[];
+   currentTurnIdRef: MutableRefObject<string | undefined>;
+   streamingMessageUidRef: MutableRefObject<string | undefined>;
+   streamingThinkingUidRef: MutableRefObject<string | undefined>;
+   baseAssistantTextRef: MutableRefObject<string>;
+   setStreamingThinkingTokens: Dispatch<SetStateAction<string[]>>;
+   setStreamingTokens: Dispatch<SetStateAction<string[]>>;
+   lastConsumedRawIndexRef: MutableRefObject<number>;
+   toolCallNotice: Pick<ToolCallNoticeApi, 'reset'>;
+   metaOnlyFinalizedTurnIdRef: MutableRefObject<string | undefined>;
+   lastMetaChunkFinalizedRef: MutableRefObject<AiStreamedChunk | undefined>;
+   resetLastReceivedStreamKind: () => void;
+ }): PromptStreamingTurnTargets {
+   const plan = createPromptStreamingTurnPlan({
+     turnId: args.turnId,
+     buildMessages: args.buildMessages,
+   });
+   return commitPromptStreamingTurn({
+     plan,
+     setMessages: args.setMessages,
+     currentTurnIdRef: args.currentTurnIdRef,
+     streamingMessageUidRef: args.streamingMessageUidRef,
+     streamingThinkingUidRef: args.streamingThinkingUidRef,
+     baseAssistantTextRef: args.baseAssistantTextRef,
+     setStreamingThinkingTokens: args.setStreamingThinkingTokens,
+     setStreamingTokens: args.setStreamingTokens,
+     lastConsumedRawIndexRef: args.lastConsumedRawIndexRef,
+     toolCallNotice: args.toolCallNotice,
+     metaOnlyFinalizedTurnIdRef: args.metaOnlyFinalizedTurnIdRef,
+     lastMetaChunkFinalizedRef: args.lastMetaChunkFinalizedRef,
+     resetLastReceivedStreamKind: args.resetLastReceivedStreamKind,
+   });
+ }
+

```

[AI Code/src/frontend/src/app/prompt/Prompt/preparePromptStreamingTurn.util.ts](#)

Copy

Review

Accept

  **patching file:**  [usePromptRunCommands.hook.ts](#)

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

Copy

Review

Accept

PatchFull file

```

-   buildPromptStreamingAssistantMessages,
-   preparePromptStreamingTurn,
-
+   buildPromptStreamingAssistantMessages,
+   commitPromptStreamingTurn,
+   createPromptStreamingTurnPlan,
+

```

```

- } from './preparePromptStreamingTurn.util.js';
-
+ } from './preparePromptStreamingTurn.util.js';
+ import type {
+   PromptStreamingTurnMessageCommitMode,

```

```

+     PromptStreamingTurnPlan,
+ } from './preparePromptStreamingTurn.util.js';
+


---


- export { COMPACT_CONTEXT_NOW_USER_PROMPT };
-
+ export { COMPACT_CONTEXT_NOW_USER_PROMPT };
+
+ /** Explicit handling policy that forces every future AiQueryOptions field to receive a
+ Retry decision. */
+ const PROMPT_RETRY_OPTION_HANDLING = {
+   reasoning: 'preserveBase',
+   verbosity: 'preserveBase',
+   agenticOptions: 'dropOneShot',
+   userPromptAugmentation: 'dropOneShot',
+   attachments: 'restoreHistory',
+ } satisfies Readonly<
+   Record<
+     keyof AiQueryOptions,
+     'preserveBase' | 'dropOneShot' | 'restoreHistory'
+   >
+ >;
+
+ /** Build the complete Retry option snapshot without spreading one-shot execution-only
+ fields. */
+ function buildPromptRetryAiQueryOptions(args: {
+   baseOptions: AiQueryOptions;
+   attachments: readonly PromptAttachment[];
+ }): AiQueryOptions {
+   // Referencing the exhaustive policy keeps its compile-time coverage part of this
+ runtime path.
+   void PROMPT_RETRY_OPTION_HANDLING;
+   return {
+     reasoning: args.baseOptions.reasoning,
+     verbosity: args.baseOptions.verbosity,
+     agenticOptions: undefined,
+     userPromptAugmentation: undefined,
+     attachments:
+       args.attachments.length > 0
+         ? replacePromptAttachments(args.attachments)
+         : undefined,
+   };
+ }
+


---


-   []],
-   );
-
+   []],
+   );
+
+ /** Commit a fully prepared replace-style scaffold only after the stream single-
+ flight accepts it. */
+ const commitPreparedStreamingTurn = useCallback(
+   (
+     plan: PromptStreamingTurnPlan,
+     messageCommitMode: PromptStreamingTurnMessageCommitMode =
+       'append',
+   ): void => {
+     // The shared commit owns every Prompt mutation so command handlers cannot
+     partially arm a turn.
+     commitPromptStreamingTurn({
+       plan,
+       messageCommitMode,

```

```

+         setMessages,
+         currentTurnIdRef,
+         streamingMessageUidRef,
+         streamingThinkingUidRef,
+         baseAssistantTextRef,
+         setStreamingThinkingTokens,
+         setStreamingTokens,
+         lastConsumedRawIndexRef,
+         toolCallNotice,
+         metaOnlyFinalizedTurnIdRef,
+         lastMetaChunkFinalizedRef,
+         resetLastReceivedStreamKind,
+     });
+ },
+ [
+     baseAssistantTextRef,
+     currentTurnIdRef,
+     lastConsumedRawIndexRef,
+     lastMetaChunkFinalizedRef,
+     metaOnlyFinalizedTurnIdRef,
+     resetLastReceivedStreamKind,
+     setMessages,
+     setStreamingThinkingTokens,
+     setStreamingTokens,
+     streamingMessageUidRef,
+     streamingThinkingUidRef,
+     toolCallNotice,
+ ],
+ );
+

```

```

-         : { type: 'prompt', promptValue: text };
-
+         : { type: 'prompt', promptValue: text };
+     const attachmentsSnapshot =
+       normalizeAttachmentsSnapshot(attachments);
+     const plan = createPromptStreamingTurnPlan({
+       turnId: newTurnId,
+       buildMessages: (targets) => {
+         // Normal sends keep one user bubble plus the canonical assistant-
side placeholders.
+         const userMessage: PromptMessage = {
+           turnId: targets.turnId,
+           uid: uidGenerator.newUniqueId,
+           role: 'user',
+           text,
+           ...(attachmentsSnapshot
+             ? { attachments: attachmentsSnapshot }
+             : {}),
+         };
+         return [
+           userMessage,
+           ...buildPromptStreamingAssistantMessages(targets),
+         ];
+       },
+     });
+

```

```

-
-     // Prime the Prompt with the exact same streaming scaffold used by all
other Prompt entry paths.
-     preparePromptStreamingTurn({
-       turnId: newTurnId,
-       setMessages,

```

```

-         buildMessages: (targets) => {
-             const attachmentsSnapshot =
-                 normalizeAttachmentsSnapshot(attachments);
-
-             // Normal sends keep one user bubble plus the canonical assistant-
side placeholders.
-             const userMessage: PromptMessage = {
-                 turnId: targets.turnId,
-                 uid: uidGenerator.newUniqueId,
-                 role: 'user',
-                 text,
-                 ...(attachmentsSnapshot
-                     ? { attachments: attachmentsSnapshot }
-                     : {}),
-             };
-
-             return [
-                 userMessage,
-                 ...buildPromptStreamingAssistantMessages(targets),
-             ];
-         },
-         currentTurnIdRef,
-         streamingMessageUidRef,
-         streamingThinkingUidRef,
-         baseAssistantTextRef,
-         setStreamingThinkingTokens,
-         setStreamingTokens,
-         lastConsumedRawIndexRef,
-         toolCallNotice,
-         metaOnlyFinalizedTurnIdRef,
-         lastMetaChunkFinalizedRef,
-         resetLastReceivedStreamKind,
-     });
+
+     // Commit the prebuilt scaffold in the same stack as the accepted start
result.
+     commitPreparedStreamingTurn(plan);
+
-
-     [
-         setUiPending,
-         baseAssistantTextRef,
-         currentTurnIdRef,
-         setMessages,
-
+     [
+         commitPreparedStreamingTurn,
+
-
-         normalizeAttachmentsSnapshot,
-         streamingMessageUidRef,
-         streamingThinkingUidRef,
-         setStreamingThinkingTokens,
-         setStreamingTokens,
-         lastConsumedRawIndexRef,
-         toolCallNotice,
-         metaOnlyFinalizedTurnIdRef,
-         lastMetaChunkFinalizedRef,
-         resetLastReceivedStreamKind,
-
+         normalizeAttachmentsSnapshot,
+
-
-         start,

```

```

-         setForceScrollSeq,
-
+         setForceScrollSeq,
+
-         sessionLocator.storageScopeRelPath,
-         setForceScrollSeq,
-
+         sessionLocator.storageScopeRelPath,
+         setForceScrollSeq,
+         setUiPending,
+         start,
+
-         verbosity: aiQueryOptionsBase.verbosity,
-     };
-
+         verbosity: aiQueryOptionsBase.verbosity,
+     };
+     const plan = createPromptStreamingTurnPlan({
+         turnId: newTurnId,
+         buildMessages: (targets) => {
+             const label = `${specifUiLabelFromAction(aiQuery.action)} $
{aiQuery.specifName}`;
+             // The UI keeps a compact structured SPECIF bubble while the
expanded provider prompt stays backend-owned.
+             const userMessage: PromptMessage = {
+                 turnId: targets.turnId,
+                 uid: uidGenerator.newUniqueId,
+                 role: 'user',
+                 text: label,
+                 specifAction: {
+                     action: aiQuery.action,
+                     specifName: aiQuery.specifName,
+                     specifMarkdown: aiQuery.specifMarkdown,
+                     strictMode: aiQuery.strictMode,
+                     programmingDocEnabled:
+                         aiQuery.programmingDocEnabled,
+                     additionalInstructions:
+                         aiQuery.additionalInstructions,
+                 },
+             };
+             return [
+                 userMessage,
+                 ...buildPromptStreamingAssistantMessages(targets),
+             ];
+         },
+     });
+
-
-     // Prime the Prompt with the compact SPECIF user bubble and the standard
assistant-side scaffold.
-     preparePromptStreamingTurn({
-         turnId: newTurnId,
-         setMessages,
-         buildMessages: (targets) => {
-             const label = `${specifUiLabelFromAction(aiQuery.action)} $
{aiQuery.specifName}`;
-
-             // The UI keeps a compact structured SPECIF bubble while the
expanded provider prompt stays backend-owned.
-             const userMessage: PromptMessage = {
-                 turnId: targets.turnId,
-                 uid: uidGenerator.newUniqueId,

```

```

-         role: 'user',
-         text: label,
-         specifAction: {
-             action: aiQuery.action,
-             specifName: aiQuery.specifName,
-             specifMarkdown: aiQuery.specifMarkdown,
-             strictMode: aiQuery.strictMode,
-             programmingDocEnabled:
-                 aiQuery.programmingDocEnabled,
-             additionalInstructions:
-                 aiQuery.additionalInstructions,
-         },
-     };
-
-     return [
-         userMessage,
-         ...buildPromptStreamingAssistantMessages(targets),
-     ];
-
-     currentTurnIdRef,
-     streamingMessageUidRef,
-     streamingThinkingUidRef,
-     baseAssistantTextRef,
-     setStreamingThinkingTokens,
-     setStreamingTokens,
-     lastConsumedRowIndexRef,
-     toolCallNotice,
-     metaOnlyFinalizedTurnIdRef,
-     lastMetaChunkFinalizedRef,
-     resetLastReceivedStreamKind,
- });
+
+     // The structured user bubble and both assistant placeholders were fully
+     // prepared before start.
+     commitPreparedStreamingTurn(plan);
+
-     aiQueryOptionsBase.verbosity,
-     baseAssistantTextRef,
-     currentTurnIdRef,
-     lastConsumedRowIndexRef,
-     lastMetaChunkFinalizedRef,
-     metaOnlyFinalizedTurnIdRef,
-     resetLastReceivedStreamKind,
-
+     aiQueryOptionsBase.verbosity,
+     commitPreparedStreamingTurn,
+
-     setForceScrollSeq,
-     setMessages,
-     setStreamingThinkingTokens,
-     setStreamingTokens,
-
+     setForceScrollSeq,
+
-     start,
-     streamingMessageUidRef,
-     streamingThinkingUidRef,
-     toolCallNotice,
-
+     start,
+

```

```

-     const handleRetryQuery = useCallback(): AiStreamStartResult => {
-         // Build the complete retry transport plan before any visible mutation.
-
+     const handleRetryQuery = useCallback(): AiStreamStartResult => {
+         // Resolve the exact persisted user turn before allocating transport or UI
+         identities.
+

```

```

-         const plannedAttachments = plannedUser.attachments ?? [];
-         const plannedOptions: AiQueryOptions = {
-             reasoning: aiQueryOptionsBase.reasoning,
-             verbosity: aiQueryOptionsBase.verbosity,
-             agenticOptions: undefined,
-             userPromptAugmentation: undefined,
-             attachments:
-                 plannedAttachments.length > 0
-                 ? replacePromptAttachments(plannedAttachments)
-                 : undefined,
-         };
-
+         const plannedAttachments = plannedUser.attachments ?? [];
+         const plannedOptions = buildPromptRetryAiQueryOptions({
+             baseOptions: aiQueryOptionsBase,
+             attachments: plannedAttachments,
+         });
+

```

```

-             : { type: 'retryQuery', promptValue: plannedUser.text ?? '' };
-
+             : { type: 'retryQuery', promptValue: plannedUser.text ?? '' };
+         const retryTurnId = plannedUser.turnId;
+         const promptIndex = messages.findIndex(
+             (message) =>
+                 message.role === 'user' && message.turnId === retryTurnId,
+         );
+         if (promptIndex < 0) throw new Error('No user prompt to retry');
+         const plan = createPromptStreamingTurnPlan({
+             turnId: retryTurnId,
+             buildMessages: (targets) => {
+                 // Preserve the first user bubble, remove only this turn's old
+                 // regenerated content,
+                 // and keep any unexpected later messages belonging to other turns.
+                 const prefix = messages.slice(0, promptIndex + 1);
+                 const suffixOtherTurns = messages
+                     .slice(promptIndex + 1)
+                     .filter((message) => message.turnId !== retryTurnId);
+                 return [
+                     ...prefix,
+                     ...buildPromptStreamingAssistantMessages(targets),
+                     ...suffixOtherTurns,
+                 ];
+             },
+         });
+

```

```

-             kind: 'promptQuery',
-             turnId: plannedUser.turnId,
-
+             kind: 'promptQuery',
+             turnId: retryTurnId,
+

```

```

-         setUiPending(true);
-

```

```

- // Overwrite semantics => no base text
- baseAssistantTextRef.current = '';
-
- // Reset ephemeral buffers and indices
- setStreamingThinkingTokens([]);
- setStreamingTokens([]);
- lastConsumedRawIndexRef.current = 0;
- toolCallNotice.reset();
-
- // Reuse SAME turnId as the last user of current turn
- let lastTurnIdOfCurrentTurn: string | undefined = undefined;
- for (let i = messages.length - 1; i >= 0; i--) {
-   const m = messages[i];
-   if (m?.role === 'user') {
-     lastTurnIdOfCurrentTurn = m.turnId;
-     break;
-   }
- }
- const retryTurnId =
-   lastTurnIdOfCurrentTurn ?? `turn:${uidGenerator.newUniqueId}`;
- currentTurnIdRef.current = retryTurnId;
-
- /**
-  * Resolve the user prompt to retry.
-  *
-  * IMPORTANT:
-  * - The Prompt UI supports "attachment-only" submissions, i.e. `promptValue
=== ""` is valid.
-  * - Therefore we must NOT use a falsy check on the string itself (" is
falsy).
-  * - We only fail if we cannot find any user message for this turn at all.
-  */
- let foundPromptToRetry = false;
- for (let i = 0; i < messages.length; i++) {
-   const m = messages[i];
-   if (m.role === 'user' && m.turnId === retryTurnId) {
-     foundPromptToRetry = true;
-     break;
-   }
- }
- if (!foundPromptToRetry) throw new Error('No user prompt to retry');
-
- // Fresh stable UUIDs for regenerated run
- const newThinkingUid = uidGenerator.newUniqueId;
- const newAssistantUid = uidGenerator.newUniqueId;
-
- // Fix target refs BEFORE streaming
- streamingThinkingUidRef.current = newThinkingUid;
- streamingMessageUidRef.current = newAssistantUid;
-
- // Remove previous assistant bubbles for this turn, then insert fresh ones
- setMessages((prev) => {
-   /**
-    * Robust pruning for retry:
-    *
-    * We must not assume the tail of the message axis is "only assistant
bubbles".
-    * In practice, timing/hydration artifacts can create a non-assistant tail,
which would
-    * prevent a naive "drop trailing assistants" loop from removing the
aborted bubble.
-    *
-    * Invariant for retry:

```

```

-         * - Keep the first user prompt message of this turn (turnId),
-         * - Drop EVERYTHING else for this turn (assistant bubbles, aborted
placeholders, etc.),
-         * - Then regenerate fresh (think + assistant) placeholders.
-         */
-         let promptIdx = -1;
-         for (let i = 0; i < prev.length; i++) {
-             const m = prev[i];
-             if (m.role === 'user' && m.turnId === retryTurnId) {
-                 promptIdx = i;
-                 break;
-             }
-         }
-
-         // If the prompt is missing (should not happen), do not mutate state.
-         if (promptIdx < 0) return prev;
-
-         const prefix = prev.slice(0, promptIdx + 1);
-
-         // Preserve messages of other turns that may (unexpectedly) exist after the
retry prompt.
-         // We only drop items belonging to the retry turn.
-         const suffixOtherTurns = prev
-             .slice(promptIdx + 1)
-             .filter((m) => m.turnId !== retryTurnId);
-
-         // Insert new thinking bubble
-         const thinkMsg: PromptMessage = {
-             turnId: retryTurnId,
-             uid: newThinkingUid,
-             role: 'assistant',
-             text: '',
-             assistantBubbleKind: 'think',
-             thinkingInitialOpen: false,
-         };
-
-         // Insert new assistant placeholder
-         const assistantMsg: PromptMessage = {
-             turnId: retryTurnId,
-             uid: newAssistantUid,
-             role: 'assistant',
-             text: '',
-             fileEditsUiFinalized: false,
-         };
-         return [...prefix, thinkMsg, assistantMsg, ...suffixOtherTurns];
-     });
-
-     // Clear meta-only idempotence guards for this regenerated run
-     metaOnlyFinalizedTurnIdRef.current = undefined;
-     lastMetaChunkFinalizedRef.current = undefined;
-
-     // Reset kind trackers for the regenerated run
-     resetLastReceivedStreamKind();
-
-     setUiPending(true);
-     commitPreparedStreamingTurn(plan, 'replaceAll');
+
- }, [
-     setUiPending,
-     baseAssistantTextRef,
-     setStreamingThinkingTokens,
-     setStreamingTokens,
-     lastConsumedRawIndexRef,

```

```

-         toolCallNotice,
-         messages,
-         currentTurnIdRef,
-         streamingThinkingUidRef,
-         streamingMessageUidRef,
-         setMessages,
-         metaOnlyFinalizedTurnIdRef,
-         lastMetaChunkFinalizedRef,
-         resetLastReceivedStreamKind,
-         sessionLocator.storageScopeRelPath,
-         start,
-
+     }, [
+
- }, [
-     aiQueryOptionsBase,
-
+ }, [
+     aiQueryOptionsBase,
+     commitPreparedStreamingTurn,
+     messages,
+     sessionLocator.storageScopeRelPath,
+     setUiPending,
+     start,
+
-         attachments: readonly PromptAttachment[],
-     ): AiStreamStartResult => {
-
+         attachments: readonly PromptAttachment[],
+     ): AiStreamStartResult => {
+         const attachmentsSnapshot =
+             normalizeAttachmentsSnapshot(attachments);
+         const plan = createPromptStreamingTurnPlan({
+             turnId,
+             buildMessages: (targets) => {
+                 // Edit replay recreates the user bubble because the whole turn is
replayed from scratch.
+                 const userMessage: PromptMessage = {
+                     turnId: targets.turnId,
+                     uid: uidGenerator.newUniqueId,
+                     role: 'user',
+                     text,
+                     ...(attachmentsSnapshot
+                         ? { attachments: attachmentsSnapshot }
+                         : {}),
+                 };
+                 return [
+                     userMessage,
+                     ...buildPromptStreamingAssistantMessages(targets),
+                 ];
+             },
+         });
+
-
-         // Rebuild the replayed Prompt turn with the same bootstrap contract as a
normal streamed run.
-         preparePromptStreamingTurn({
-             turnId,
-             setMessages,
-             buildMessages: (targets) => {
-                 const attachmentsSnapshot =
-                     normalizeAttachmentsSnapshot(attachments);

```

```

-
- // Edit replay recreates the user bubble because the whole turn is
replayed from scratch.
-
-     const userMessage: PromptMessage = {
-         turnId: targets.turnId,
-         uid: uidGenerator.newUniqueId,
-         role: 'user',
-         text,
-         ...(attachmentsSnapshot
-             ? { attachments: attachmentsSnapshot }
-             : {}),
-     };
-
-     return [
-         userMessage,
-         ...buildPromptStreamingAssistantMessages(targets),
-     ];
-
-     },
-     currentTurnIdRef,
-     streamingMessageUidRef,
-     streamingThinkingUidRef,
-     baseAssistantTextRef,
-     setStreamingThinkingTokens,
-     setStreamingTokens,
-     lastConsumedRawIndexRef,
-     toolCallNotice,
-     metaOnlyFinalizedTurnIdRef,
-     lastMetaChunkFinalizedRef,
-     resetLastReceivedStreamKind,
- });
-
+
+ // Commit the complete edit replay plan only after the transport accepted
the attempt.
+
+     commitPreparedStreamingTurn(plan);
+
-
-     [
-         setUiPending,
-         baseAssistantTextRef,
-         setStreamingThinkingTokens,
-         setStreamingTokens,
-         lastConsumedRawIndexRef,
-         toolCallNotice,
-         currentTurnIdRef,
-         streamingThinkingUidRef,
-         streamingMessageUidRef,
-         setMessages,
-
+
+     [
+         commitPreparedStreamingTurn,
+
-
-         normalizeAttachmentsSnapshot,
-         metaOnlyFinalizedTurnIdRef,
-         lastMetaChunkFinalizedRef,
-         resetLastReceivedStreamKind,
-
+
+         normalizeAttachmentsSnapshot,
+
-
-         normalizeAttachmentsSnapshot,
-         sessionLocator.storageScopeRelPath,
-         start,
-

```

```

+         normalizeAttachmentsSnapshot,
+         sessionLocator.storageScopeRelPath,
+         setForceScrollSeq,
+         setUiPending,
+         start,
+
-
-         /**
-         * Start streaming right away.
-         *
-         * Note:
-         * - Attachments are intentionally not supported for SPECIF actions.
-         * - We still forward reasoning/verbosity base options to stay consistent
with the main Send flow.
-         */
-         // Force scroll.
-
+         // Attachments remain unsupported for SPECIF actions; only the accepted
structured scaffold is visible.
+

```

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

Copy

Review

Accept

  **patching file:**  [preparePromptStreamingTurn.util.test.ts](#)

[AI Code/src/frontend/src/app/prompt/Prompt/ test /preparePromptStreamingTurn.util.test.ts](#)

Copy

Review

Accept

PatchFull file

```

- import type { PromptMessage } from '../PromptHistory/promptHistoryTypes.type.js';
-
+ import type { PromptMessage } from '../PromptHistory/promptHistoryTypes.type.js';
+ import type { PromptStreamingTurnTargets } from
'../preparePromptStreamingTurn.util.js';
+
-
-     buildPromptStreamingAssistantMessages,
-
+     buildPromptStreamingAssistantMessages,
+     commitPromptStreamingTurn,
+
-
- /** Build one mutable ref-like object compatible with the helper contract. */
-
+
+ it('commits an accepted prebuilt plan without executing its scaffold builder again', ()
=> {
+     let messagesState: PromptMessage[] = [];
+     const setMessages = jest.fn<
      (update: SetStateAction<PromptMessage[]>) => void
+     >((update) => {
+         messagesState = applyStateUpdate(messagesState, update);
+     });
+     const buildMessages = jest.fn(
      (targets: PromptStreamingTurnTargets): readonly PromptMessage[] => [
+         {
+             turnId: targets.turnId,

```

```

+         uid: 'planned-user',
+         role: 'user',
+         text: 'Planned before start',
+     },
+     ...buildPromptStreamingAssistantMessages(targets),
+ ],
+ );
+ const plan = createPromptStreamingTurnPlan({
+     turnId: 'turn-prebuilt',
+     buildMessages,
+ });
+ buildMessages.mockClear();
+
+ commitPromptStreamingTurn({
+     plan,
+     setMessages,
+     currentTurnIdRef: createRef<string | undefined>(undefined),
+     streamingMessageUidRef: createRef<string | undefined>(undefined),
+     streamingThinkingUidRef: createRef<string | undefined>(undefined),
+     baseAssistantTextRef: createRef('stale'),
+     setStreamingThinkingTokens: jest.fn(() => {}),
+     setStreamingTokens: jest.fn(() => {}),
+     lastConsumedRowIndexRef: createRef(10),
+     toolCallNotice: { reset: jest.fn(() => {}) },
+     metaOnlyFinalizedTurnIdRef: createRef<string | undefined>('old'),
+     lastMetaChunkFinalizedRef: createRef<AiStreamedChunk | undefined>(
+         undefined,
+     ),
+     resetLastReceivedStreamKind: jest.fn(() => {}),
+ });
+
+ expect(buildMessages).not.toHaveBeenCalled();
+ expect(messagesState).toEqual(plan.messages);
+ });
+
+ /** Build one mutable ref-like object compatible with the helper contract. */
+

```

[AI Code/src/frontend/src/app/prompt/Prompt/ test /preparePromptStreamingTurn.util.test.ts](#)

Copy

Review

Accept

  **patching file:** [usePromptRunCommands.startAcceptance.test.tsx](#)

[AI Code/src/frontend/src/app/prompt/Prompt/ test /usePromptRunCommands.startAcceptance.test.tsx](#)

Copy

Review

Accept

PatchFull file

```

-     expect(toolCallNoticeMock.reset).toHaveBeenCalledTimes(1);
-     });
-
+     expect(toolCallNoticeMock.reset).toHaveBeenCalledTimes(1);
+     });
+
+ it('commits one prebuilt Retry replacement only after synchronous acceptance', ()
=> {
+     startMock.mockImplementation((args) => ({
+         accepted: true,
+         runRef: {
+             locator: sessionLocator,
+             turnId: args.turnId,

```

```

+         sourceId: 'retry-source',
+       },
+       streamInstanceId: 'retry-stream',
+     ));
+     const { result } = renderHook(() => useCommandHarness());
+     let retryResult: AiStreamStartResult | undefined;
+
+     act(() => {
+       retryResult = result.current.handleRetryQuery();
+     });
+
+     expect(retryResult).toEqual(
+       expect.objectContaining({
+         accepted: true,
+         streamInstanceId: 'retry-stream',
+       }),
+     );
+     expect(startMock).toHaveBeenCalledWith({
+       kind: 'promptQuery',
+       turnId: 'turn',
+       storageScopeRelPath: '',
+       aiQuery: { type: 'retryQuery', promptValue: 'Question' },
+       aiQueryOptions: {
+         reasoning: 'medium',
+         verbosity: 'low',
+         agenticOptions: undefined,
+         userPromptAugmentation: undefined,
+         attachments: undefined,
+       },
+     });
+     expect(result.current.uiPending).toBe(true);
+     expect(result.current.messages).toHaveLength(3);
+     expect(result.current.messages[0]).toEqual(
+       expect.objectContaining({
+         uid: 'user',
+         role: 'user',
+         text: 'Question',
+       }),
+     );
+     expect(result.current.messages[1]).toEqual(
+       expect.objectContaining({
+         role: 'assistant',
+         assistantBubbleKind: 'think',
+       }),
+     );
+     expect(result.current.messages[2]).toEqual(
+       expect.objectContaining({
+         role: 'assistant',
+         fileEditsUiFinalized: false,
+       }),
+     );
+     expect(
+       result.current.messages.some(
+         (message) => message.text === 'Partial answer',
+       ),
+     ).toBe(false);
+     expect(toolCallNoticeMock.reset).toHaveBeenCalledTimes(1);
+   });
+

```

[AI Code/src/frontend/src/app/prompt/Prompt/ test /usePromptRunCommands.startAcceptance.test.tsx](#)

Copy
Review
Accept

  reading

project:

 [preparePromptStreamingTurn.util.ts](#),
 [usePromptRunCommands.hook.ts](#),
 [SidebarStore.rehydrate.manager.ts](#),  [preparePromptStreamingTurn.util.test.ts](#) [L1-L100],  [delete.endpoint.ts](#) [L1-L210],  [SidebarStore.queryWindow.listeners.manager.ts](#) [L45-L120],  e "[Règle await boucle](#)",  e "[Règle boucle constante](#)",  e "[Règle void frontend](#)",  e "[Règle type assertions](#)",  e "[Préfixe running constant](#)"

  patching file:  [SidebarStore.rehydrate.manager.ts](#)

[AICode/src/extension/src/extension/panels/sidebar/SidebarStore.rehydrate.manager.ts](#)

Grouped

([what is it?](#))

  patching file:  [preparePromptStreamingTurn.util.test.ts](#)

[AICode/src/frontend/src/app/prompt/Prompt/_test_/preparePromptStreamingTurn.util.test.ts](#)

Grouped

([what is it?](#))

  patching file:  [usePromptRunCommands.hook.ts](#)

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

Grouped

([what is it?](#))

  scanning git:  "[Résumé des corrections](#)",  "[Qualité syntaxique diff](#)",  "[Diff fichiers corrigés](#)"

  reading project:  e "[Ancien cancel promotion](#)",  e "[Réhydratation multi mutations](#)",  e "[Plans post start restants](#)",  [tsconfig.json](#),  [tsconfig.json](#),  [tsconfig.jest.json](#),  [tsconfig.jest.json](#),  e "[Usage direct sidebarStore](#)"

  patching file:  [delete.endpoint.ts](#)

[AICode/src/extension/src/slimbk-local/endpoints/ai/orch/chat/delete.endpoint.ts](#)

Grouped

([what is it?](#))

  patching file:  [delete.endpoint.liveSession.test.ts](#)

[AICode/src/extension/src/slimbk-local/endpoints/ai/orch/chat/_test_/delete.endpoint.liveSession.test.ts](#)

Grouped

([what is it?](#))

  patching file:  [preparePromptStreamingTurn.util.test.ts](#)

[AICode/src/frontend/src/app/prompt/Prompt/_test_/preparePromptStreamingTurn.util.test.ts](#)

Grouped

([what is it?](#))

  reading project:  e "[Autres commits scaffold](#)",  [SidebarStore.rehydrate.manager.test.ts](#) [L60-L115],  [SidebarStore.rehydrate.manager.test.ts](#) [L430-L570],  [delete.endpoint.liveSession.test.ts](#) [L180-L330],  [LiveSessionRegistry.service.test.ts](#) [L105-L190],  [usePromptRunCommands.startAcceptance.test.tsx](#) [L105-L260],  [delete.endpoint.liveSession.test.ts](#) [L300-L403]

  patching file:  [LiveSessionRegistry.service.test.ts](#)

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

Grouped

([what is it?](#))

  reading project:  [LiveSessionRegistry.service.test.ts](#) [L155-L190]

  patching file:  [preparePromptStreamingTurn.util.ts](#)

[AICode/src/frontend/src/app/prompt/Prompt/preparePromptStreamingTurn.util.ts](#)

Grouped

([what is it?](#))

  patching file:  [delete.endpoint.liveSession.test.ts](#)

[AICode/src/extension/src/slimbk-local/endpoints/ai/orch/chat/_test_/delete.endpoint.liveSession.test.ts](#)

Grouped

([what is it?](#))

  reading project:  e "[Runtime Activity snapshot](#)",  [SidebarQueryRunningBridge.ts](#) [L150-

L215],  e "[Réhydratation tests indirects](#)",  h "[Sidebar reset](#)

[broadcasts](#)",  [SidebarStore.manager.queryRunningUpsert.test.ts](#) [L1-L100],  e "[Void garde typage](#)",  e "[Boucles revalidation](#)"

  patching file:  [delete.endpoint.liveSession.test.ts](#)

[AICode/src/extension/src/slimbk-local/endpoints/ai/orch/chat/_test_/delete.endpoint.liveSession.test.ts](#)

Error

([what is it?](#))

  reading project:  [delete.endpoint.liveSession.test.ts](#) [L20-L55],  [delete.endpoint.liveSession.test.ts](#) [L245-

L320],  [delete.endpoint.liveSession.test.ts](#) [L355-L390]

  patching file:  [delete.endpoint.liveSession.test.ts](#)

[AICode/src/extension/src/slimbk-local/endpoints/ai/orch/chat/_test_/delete.endpoint.liveSession.test.ts](#)

Grouped

([what is it?](#))

  scanning recent files

  reading project:  [delete.endpoint.ts](#) [L1-L205],  [delete.endpoint.liveSession.test.ts](#) [L25-

L50],  [delete.endpoint.liveSession.test.ts](#) [L250-L410],  [LiveSessionRegistry.service.ts](#) [L185-

L260],  [SidebarStore.rehydrate.manager.ts](#) [L250-L500],  [usePromptRunCommands.hook.ts](#) [L1-

L150],  [usePromptRunCommands.hook.ts](#) [L150-L570]

Corrections terminées 

Joel, les écarts identifiés lors de la vérification ont tous été corrigés. Le latch ne peut plus prendre la sortie de secours avant la fin de la FIFO — le petit fugitif est désormais sous haute surveillance. 

J'ai relu les documents d'architecture `liveSession`, Queue B et fallback persist avant l'implémentation. Les corrections restaurent leurs invariants sans modifier les algorithmes de streaming, les sanitizers ou les fichiers générés.

Résumé des changements

Domaine	Correction appliquée	Couverture ajoutée	État
Settlement source	La promotion consomme définitivement le handle de réservation. Une annulation générique post-promotion ne résout plus prématurément le latch.	Annulation normale, promotion, settlement après finalisation et éviction pré-promotion	
Réhydratation Activity	Les panneaux, sources et réservations sont recapturés après chaque lecture persistante. La projection finale est installée en une mutation atomique.	Changement d'autorité pendant un <code>await</code> , disparition de panneau et nouvelle source	

Domaine	Correction appliquée	Couverture ajoutée	État
Observer EasyAgents	Un observer <code>sourceSettled</code> reste <code>Running/observer</code> / <code>open</code> jusqu'au véritable <code>handoffReady</code> .	État intermédiaire <code>settlement</code> → <code>handoff</code>	✓
Delete	La décision destructive repose désormais sur le registre exact sous <code>claim</code> , et non sur la projection <code>Activity</code> .	Observer avec <code>Activity</code> , observer sans <code>Activity</code> et lignes <code>Activity</code> obsolètes	✓
Atomicité frontend	<code>Send</code> , <code>SPECIF</code> , <code>Retry</code> , <code>Edit</code> , <code>Compact</code> , <code>Reply</code> et <code>Continue</code> préparent leur plan avant <code>start()</code> , puis commettent après acceptation.	Refus sans mutation, <code>Retry</code> accepté et <code>scaffold</code> préconstruit	✓
Typage <code>Retry</code>	Chaque clé de <code>AiQueryOptions</code> doit recevoir une politique explicite. Toute future clé non traitée provoquera une erreur de typage.	Garde exhaustif statique restauré	✓

Détails techniques



Latch de settlement exact

Dans `LiveSessionRegistry.service.ts` :

- la promotion réservation → record supprime désormais le mapping du handle ;
- une annulation post-promotion devient un véritable no-op ;
- seule la finalisation post-forwarding peut alors résoudre le latch ;
- une réservation non promue puis évincée conserve son chemin de settlement par annulation exacte ;
- les handles consommés sont retirés de la `WeakMap`, évitant également leur maintien inutile.

La non-régression est couverte dans `LiveSessionRegistry.service.test.ts`.



Réhydratation `Activity` atomique

Dans `SidebarStore.rehydrate.manager.ts` :

- les panneaux et snapshots live sont capturés ;
- les métadonnées persistantes manquantes sont chargées ;
- après chaque attente, les autorités sont recapturées depuis zéro ;
- la projection complète est construite hors store ;
- la liste `Activity` est remplacée une seule fois.

Cette structure empêche désormais :

- une ancienne capture de supprimer une source démarrée pendant une lecture disque ;
- un panneau fermé pendant l'attente de réapparaître artificiellement ;
- l'exposition transitoire d'une liste vide ou partiellement reconstruite ;
- un ancien record terminal de masquer une source successive du même locator.

La mutation atomique et la resynchronisation des titres de panneaux sont centralisées dans `SidebarStore.queryWindow.listeners.manager.ts`.

Les scénarios concurrents sont ajoutés à `SidebarStore.rehydrate.manager.test.ts`.



Observer settlement avant handoff

La réhydratation n'élimine plus tous les records dès `sourceSettled=true`.

Le comportement est maintenant :

Ownership	État du record	Projection
Source	Active/non finalisée	<code>Running/source/open</code> <code>closed</code>
Observer	Source non settled	<code>Running/observer/open</code>
Observer	Source settled, handoff en attente	<code>Running/observer/open</code>
Source ou observer	Handoff prêt	<code>Idle/open</code> si le panneau existe
Observer	Panneau fermé	Aucune projection

Cela préserve correctement la phase de convergence Git/persistence EasyAgents située entre le settlement du transport et le handoff métier.



Delete fondé sur l'autorité du registre

Dans `delete.endpoint.ts` :

- le claim destructif est acquis avant la décision finale ;
- les snapshots live exacts sont relus sous ce claim ;

- toute réservation observer est bloquante ;
- tout record observer non handoff-ready est bloquant, même après `sourceSettled` ;
- une projection Activity observer obsolète n'est plus considérée comme une autorité ;
- seuls les runs source Prompt normaux sont abortés puis attendus ;
- un refus observer libère le claim sans fermer le panneau, supprimer les données ou déclencher une réconciliation destructive.

Les trois cas — observer projeté, observer absent d'Activity et Activity observer obsolète — sont couverts dans [delete.endpoint.liveSession.test.ts](#).



Plans frontend préparés avant Start

Dans [preparePromptStreamingTurn.util.ts](#) :

- la préparation pure et le commit sont désormais deux opérations distinctes ;
- le commit ne génère plus d'identité et ne rappelle aucun constructeur de scaffold ;
- le caller choisit explicitement entre ajout du scaffold et remplacement complet de l'axe des messages ;
- l'ancien helper reste disponible pour les chemins dont le transport a déjà été accepté.

Dans [usePromptRunCommands.hook.ts](#) :

- Send prépare son user message et ses deux placeholders avant `start()` ;
- SPECIF prépare son message structuré avant `start()` ;
- Edit prépare son tour reconstruit avant `start()` ;
- Retry prépare également l'axe final complet avant `start()` ;
- Compact et Reply réutilisent le chemin Send ;
- Continue conserve son plan spécialisé déjà pur ;
- aucun plan n'est committé lorsque le single-flight refuse la tentative.

Les contrats sont renforcés dans :

- [preparePromptStreamingTurn.util.test.ts](#)
- [usePromptRunCommands.startAcceptance.test.tsx](#)

Fichiers modifiés

Fichier

[LiveSessionRegistry.service.ts](#)

[LiveSessionRegistry.service.test.ts](#)

[SidebarStore.rehydrate.manager.ts](#)

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

[SidebarStore.rehydrate.manager.test.ts](#)

[delete.endpoint.ts](#)

[delete.endpoint.liveSession.test.ts](#)

[preparePromptStreamingTurn.util.ts](#)

[usePromptRunCommands.hook.ts](#)

[preparePromptStreamingTurn.util.test.ts](#)

[usePromptRunCommands.startAcceptance.test.tsx](#)

État final

Invariant

- Settlement uniquement après abandon réel de l'autorité source
- Aucun Delete pendant un observer actif
- Activity observer maintenue jusqu'au handoff métier
- Aucun snapshot pré-`await` réappliqué après changement d'autorité
- Une seule mutation Activity complète pendant la réhydratation
- Aucun scaffold Prompt sur refus de Start
- Aucun constructeur de scaffold exécuté après acceptation
- Toute future option Retry explicitement classifiée
- Fichiers générés préservés
- Ensemble des corrections demandées

Responsabilité

- Consommation du handle après promotion
- Barrières de settlement
- Recapture et projection Activity
- Remplacement Activity atomique
- Courses settlement/réhydratation
- Autorité destructive du registre
- Observers exacts et Activity obsolète
- Séparation plan/commit
- Plans transactionnels des commandes
- Plan préconstruit non réexécuté
- Refus sans mutation et Retry accepté

État



Complet

Nom du commit

text

Copy

fix(live-session): close settlement activity and delete races

🎵 Les sources attendent jusqu'au vrai signal 🗨️ Activity reste exacte, Delete devient loyal 🎵

</ASSISTANT>

<USER>

FAIL src/messageBus/handlers/ia/__test__/SidebarQueryRunningBridge.race.test.ts • Test suite failed to run SyntaxError: The requested module './SidebarStore.queryWindow.listeners.manager.js' does not provide an export named 'replaceQueryActivitiesForRehydration' 66 | 67 | const { SidebarQueryActivityRuntimeService } = > 68 | await import('../SidebarQueryRunningBridge.js'); | ^ 69 | 70 | const LOCATOR = { 71 | sessionId: 'a'.repeat(32), at EsmLoader.tryLoadGraphSync (node_modules/jest-runtime/build/index.js:725:18) at src/messageBus/handlers/ia/__test__/SidebarQueryRunningBridge.race.test.ts:68:5

</USER>

<ASSISTANT>

I'm thinking...

</ASSISTANT>

<ASSISTANT>

Diagnostic ciblé — LLM-DEBUG-3 🧑🔧

Le symptôme indique très probablement un **mock ESM incomplet dans le test**, pas un défaut du code applicatif : le module réel exporte la nouvelle fonction, mais un `unstable_mockModule(...)` peut remplacer entièrement ses exports et oublier celle-ci. Je vérifie la chaîne d'import et tous les mocks concernés avant de corriger le test proprement.

🔧 ✅ reading

project:

📄 [SidebarQueryRunningBridge.race.test.ts](#),

📄 [SidebarQueryRunningBridge.ts](#),

📄 [SidebarStore.queryWindow.listeners.manager.ts](#), 📄 [SidebarStore.rehydrate.manager.ts](#), 🗨️ e "[Mocks listeners Activity](#)", 📁 g "[Docs architecture proches](#)", 🗨️ "[Mocks ESM concernés](#)"

🔧 ✅ patching file: 📄 [SidebarQueryRunningBridge.race.test.ts](#)

[AICode/src/extension/src/messageBus/handlers/ia/__test__/SidebarQueryRunningBridge.race.test.ts](#)

Copy

Review

Accept

PatchFull file

```
-      panelOpened: async () => {},
-
+      panelOpened: async () => {},
+      // SidebarStore core loads the Activity rehydration module, which statically
+      imports this
+      // helper even though the isolated runtime-bridge scenarios never execute
+      rehydration.
+      replaceQueryActivitiesForRehydration: async () => {},
+
```

[AICode/src/extension/src/messageBus/handlers/ia/__test__/SidebarQueryRunningBridge.race.test.ts](#)

Copy

Review

Accept

🔧 ✅ reading project: 📄 [package.json](#)

Correction appliquée — LLM-DEBUG-3 ✅

Cause réelle

L'échec provenait **du test**, pas du code applicatif.

Le test `SidebarQueryRunningBridge.race.test.ts` remplace entièrement le module ESM `SidebarStore.queryWindow.listeners.manager.ts` avec `jest.unstable_mockModule()`. Depuis la correction précédente, `SidebarStore.rehydrate.manager.ts` importe statiquement le nouvel export `replaceQueryActivitiesForRehydration`. Le mock ESM du test ne le déclarait pas : Jest construisait donc un namespace de module incomplet et échouait **avant même l'exécution des tests**.

Correction

Le mock du test expose maintenant une implémentation asynchrone neutre de `replaceQueryActivitiesForRehydration`.

Cette fonction n'est pas exécutée par les scénarios du runtime bridge ; elle doit seulement être présente pour respecter le graphe statique des exports ESM. Aucun code applicatif n'a été modifié pour contourner le test.

Vérification	Résultat	Complétude
Origine du défaut identifiée	Mock ESM incomplet	✓
Code applicatif conservé	Aucun changement fonctionnel	✓
Test corrigé au bon niveau	Export manquant ajouté au mock	✓
Typage du mock	Fonction asynchrone cohérente avec le contrat réel	✓
Autres mocks examinés	Ils remplacent aussi le store central et ne chargent pas cette chaîne réelle	✓
Correction complète	Oui	✓
Test ciblé à relancer		
Depuis le package extension :		

```
bash
Copy
npm test -- SidebarQueryRunningBridge.race.test.ts
```

Si une autre erreur apparaît, envoie-moi la nouvelle sortie complète : j'appliquerai l'itération suivante de LLM-DEBUG-3 en déterminant de nouveau si elle vient du test ou du code applicatif.

Nom du commit

```
text
Copy
test(live-session): complete sidebar activity ESM mock
```

🎵 Le mock retrouve enfin son export 🧨 Jest peut remonter à bord, sans toucher au code qui dort 🎵