

Permalink URI:

vscode://joelabenhaim.aicode-ja/chat?sessionId=80cb43d4baf140db91a341b77940e988-VS-V%C3%A9rifier-Finalise

<USER>

[Verify]SPECIF_FINALISER_LIVESESSION_POUR_LES_SESSIONS_PROMPT_NORMALES_20260712_204731

</USER>

<ASSISTANT>

I'm thinking...

</ASSISTANT>

<ASSISTANT>

🔧 ☒ scanning recent files

🔧 ☒ scanning git: 🗑️ "État Git de la tranche", 🗑️ "Fichiers modifiés", 🗑️ "Statistiques du diff"

🔧 ☒ reading project: 📄 [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: 🗑️ "Commit courant", 🗑️ "Historique récent", 🗑️ "Inventaire depuis la base", 🗑️ "Ampleur depuis la base"

🔧 ☒ reading project: 📄 h "Contrats liveSession", 📄 h "Registre liveSession", 📄 h "Source et transport", 📄 h "Panneaux et activité", 📄 h "Frontend live Prompt", 📄 "Preuves des invariants", 📄 [iaStreamHandlers.ts](#) [L1-L340], 📄 [liveSessionStreamHandlers.ts](#) [L1-L300], 📄 [stream.endpoint.ts](#) [L1-L260], 📄 [LiveSessionSourceLifecycle.service.ts](#) [L1-L220], 📄 [LiveSessionOpen.service.ts](#) [L1-L330], 📄 [PromptLiveSessionPanelQuery.util.ts](#) [L1-L220], 📄 [PromptLiveSessionDetachedStop.service.ts](#) [L1-L100], 📄 [openPanel.util.ts](#) [L1-L340], 📄 [SidebarQueryRunningBridge.ts](#) [L1-L470], 📄 [SidebarStore.queryWindow.listeners.manager.ts](#) [L1-L620], 📄 [SidebarStore.queryWindow.mutations.manager.ts](#) [L1-L230], 📄 [SidebarStore.rehydrate.manager.ts](#) [L1-L620], 📄 [PromptThreadCleanup.manager.ts](#) [L1-L300], 📄 [PromptQueryDoneSound.manager.ts](#) [L1-L360], 📄 [openPanel.util.ts](#) [L330-L678], 📄 [PromptLiveSessionPanelCoordinator.service.ts](#) [L190-L340], 📄 [useAiStream.hook.ts](#), 📄 [usePromptLiveSession.hook.ts](#), 📄 [PromptLiveSessionBootstrap.adapter.ts](#), 📄 [NormalPromptLiveSession.adapter.ts](#), 📄 [applyHydratedPromptSessionState.util.ts](#), 📄 [AiStreamStartArgs.type.ts](#), 📄 [PromptLiveSession.types.ts](#), 📄 e "Composer après échec", 📄 e "Stop après échec", 📄 e "Queue completion error", 📄 h "Barrières Queue A B", 📄 [Prompt.tsx](#), 📄 [usePromptRunCommands.hook.ts](#), 📄 [usePromptRunController.hook.ts](#), 📄 [usePromptRunController.finalization.hook.ts](#), 📄 [usePromptCancelController.hook.ts](#), 📄 [preparePromptStreamingTurn.util.ts](#), 📄 [preparePromptContinuationStreaming.util.ts](#), 📄 [usePromptSessionLocator.hook.ts](#), 📄 [usePromptSessionHydration.hook.ts](#), 📄 [EasyAgentsBatchLiveSession.adapter.ts](#), 📄 [PromptInput.tsx](#), 📄 [SidebarActivityQuery.tsx](#), 📄 [SidebarStore.handlers.manager.ts](#), 📄 [SidebarUiMessages.ts](#), 📄 [MessageRegistry.ts](#), 📄 [VscButton.tsx](#), 📄 [extTypedMessageBus.util.ts](#), 📄 [extMessageBus.util.ts](#), 📄 [abortSource.endpoint.ts](#), 📄 [abort.endpoint.ts](#), 📄 [open.endpoint.ts](#), 📄 [stream.endpoint.ts](#), 📄 [aiStopSuffix.util.ts](#), 📄 [OrchMessagesMemoryAssistantPatch.ts](#), 📄 [Slimbk.ts](#) [L80-L170], 📄 [SlimbkResult.type.ts](#), 📄 [slimbkBuildHeader.ts](#) [L1-L130], 📄 [slimbkBuildHeader.ts](#) [L350-

L436],  [Slimbk.streamingErrorHandler.test.ts](#),  [slimbkBuildHeader.errorCode.test.ts](#),  e "[Header erreur streaming](#)",  [Slimbk.ts](#) [L220-L330],  [LiveSessionRegistry.service.ts](#),  [LiveSessionService.ts](#),  h "[Replay liveSession](#)"

  scanning git:  ["Diff du registre"](#)
  reading

project:

 [LiveSessionReplay.service.ts](#),
 [delete.endpoint.ts](#),
 [EasyAgentsBatchRunRetry.service.ts](#),
 [EasyAgentsBatchLiveSessionAdapter.service.ts](#),
 [EasyAgentsBatchRunChildExecution.service.ts](#),
 [EasyAgentsBatchRunExecution.service.ts](#),
 [EasyAgentsBatchRunManager.service.ts](#),
 [SidebarStore.core.manager.ts](#),  r "[deleteSessionDirsBySessionId](#)",  e "[Suppression durable chat](#)",
 [ChatPersistenceDelete.service.ts](#),  [useAiStreamQueue.hook.ts](#),  [usePromptStreamDispatchGateQueue.hook.ts](#)
  scanning git:  ["Diff Queue A"](#),  ["Diff Queue B"](#),  ["Diff verrou Prompt"](#),  ["Diff sanitizers"](#),  ["Contrôle du diff"](#),  ["Occurrences any"](#),  ["Suppressions lint TODO"](#),  ["Imports dynamiques"](#),  ["Appels require"](#), ["Doubles assertions"](#), ["Marqueurs TODO"](#), ["Assertions openPanel test"](#), ["Assertions rehydrate test"](#), ["Assertions origine test"](#), ["Assertions endpoint test"](#), ["Assertions commandes"](#)
  reading project:  h "[Échecs de queue testés](#)",  e "[Stop détaché refusé](#)",  h "[Réactivation Stop détaché](#)", h "[Suppression source active](#)",  [SidebarActivityQuery.test.tsx](#) [L1-L240],  [delete.endpoint.liveSession.test.ts](#) [L1-L303],  [useStreamListener.hook.ts](#)
  scanning git:  ["Volume source du commit"](#),  ["Fichiers source du commit"](#)
  reading project:  e "[Couverture detach item](#)",  e "[Couverture historique Batch](#)",  e "[Couverture onglet ciblé](#)", e "[Couverture cleanup sonore](#)", e "[Types any liveSession](#)", e "[Types any frontend Prompt](#)", e "[Types any flux frontend](#)", e "[Types any panneaux](#)", e "[Types any handlers](#)"
  scanning git:  ["Diff test avec any"](#)
  reading project:  h "[Concurrence brouillon source](#)",  [ChatPersistenceWriter.service.ts](#) [L150-L250],  e "[SourceId fourni au start](#)",  [useAiStream.hook.promptSourceIdentity.test.ts](#) [L130-L230],  [LiveSessionOpenResult.type.ts](#) [L109-L170],  [PromptLiveSessionPanelCoordinator.service.ts](#) [L340-L430],  r "[onStart](#)",  e "[Façade onStart restante](#)",  e "[Méthode compatibilité](#)",  e "[Déclaration onStart](#)", e "[Type état stream actif](#)", e "[Usage replayMode frontend](#)", e "[Wrapper disposeForAgent](#)", e "[Helper viewer mode](#)", h "[Échecs coordinateur](#)",  [PromptLiveSessionPanelCoordinator.service.test.ts](#) [L470-L524],  [frontMessageBus.util.ts](#) [L1-L220],  [frontMessageBus.util.ts](#) [L220-L350],  h "[Persistance avant émission](#)",  h "[Écriture mémoire en streaming](#)", e "[Barrière fallback émission](#)"

Rapport de vérification

Joel, c'est un travail colossal — cette tranche possède assez de barrières anti-course pour sécuriser une centrale nucléaire... mais j'ai tout de même trouvé quelques électrons fuyeurs 🧐 😊.

Périmètre vérifié

- **État courant** : dépôt propre, HEAD = 3f04a59be8f5f9eb99af7fdee1364f2bc3ebd432.
- **Commit courant** : feat(live-session): keep prompt runs alive across panel close and reopen.
- **Couverture du manifeste** : les **160 chemins numérotés de la spécification** sont bien présents dans le diff d'implémentation.
- **Diff réel** : **182 fichiers source**, soit **22 fichiers supplémentaires** par rapport à la liste dite « exacte » de la spécification.
- Ces fichiers supplémentaires correspondent principalement :
 - à la propagation d'identité dans [Queue A](#) et [Queue B](#) ;
 - aux adaptations du framework et du générateur Slimbk dans [le framework Slimbk](#) et [le générateur de client Slimbk](#) ;
 - aux sorties générées et à plusieurs harness/tests supplémentaires.
- Le diff est propre concernant les erreurs de whitespace et n'ajoute pas de TODO, de suppression ESLint ni d'import dynamique de production.

J'ai également lu les documents de maintenance demandés : [architecture liveSession](#), [architecture EasyAgents](#), [invariant d'isomorphisme](#), [invariants thinking](#), [invariants Queue B](#) et [barrière fallback persist](#).

1) Toute la spécification est-elle implémentée ?

Conclusion : ❌ Non, pas entièrement au sens strict — estimation 92 %

L'architecture principale est bien présente, les 160 fichiers annoncés ont été traités, les chemins normaux de fermeture/réouverture sont cohérents et une couverture de tests très importante a été ajoutée.

La validation ne peut cependant pas être donnée à **100 %**, principalement à cause de :

1. Une course critique lors de la suppression d'une session dont la source normale tourne encore, pouvant recréer les données après leur suppression.
2. Une perte des erreurs internes de Queue A/Queue B, suivie d'un faux résultat `completed`.
3. Une fuite process-lifetime possible de l'état sonore pour une finalisation `stale`.
4. Un bouton Stop détaché qui reste définitivement désactivé si le RPC est refusé ou échoue.
5. Un contrat `sourceId` relâché, alors que l'identité devait être exclusivement générée après acceptation.
6. Le résultat d'acceptation de Reply masqué par le type public du contrôleur.
7. Du code de compatibilité intermédiaire inutilisé, contraire au caractère atomique/final de la tranche.
8. Une régression de couverture de plusieurs comportements historiques du panneau central.
9. Une liste de fichiers de spécification inexacte, puisqu'elle omet 22 fichiers réellement nécessaires.

Axe de vérification	Estimation	Verdict
Présence des 160 fichiers annoncés	100 %	✓
Architecture globale demandée	97 %	✓
Fonctionnement nominal fermeture/réouverture	96 %	✓
Robustesse aux courses et erreurs internes	84 %	⚠
Respect strict des contrats de types	91 %	⚠
Couverture de tests	89 %	⚠
Conformité globale estimée	92 %	⚠

2) L'implémentation actuelle fait-elle ce qu'elle est censée faire selon la spécification ? Pour chaque fonctionnalité technique demandée, est-elle complète ?

Conclusion : ✓ Le scénario nominal est largement réalisé, mais plusieurs garanties de panne ou de destruction restent incomplètes

Fonctionnalité technique demandée	État	Estimation	V
LiveSessionRunRef avec locator + turnId + sourceId et clés structurelles	✓	100 %	Contrat strict et clé JSON stru
Modes replaceTurnAssistants et continueAssistant	✓	100 %	Contrat partagé et application
Descripteurs stricts EasyAgents/normalPrompt	✓	100 %	Ligne EasyAgents obligatoire de fermeture exacte et égalité
Détection des paramètres partiels, inconnus et dupliqués	✓	100 %	Défense extension dans le par frontend dans le dispatcher bo
Snapshot brut de query figé et frontière React externe	✓	100 %	Le snapshot repose bien sur u invalide ne monte ni contrôleur dans Prompt .
Désactivation de l'hydratation classique et des auto-runs pour toute query live explicite	✓	100 %	L'effet classique est désactivé immédiatement.
Contexte exact de webview extension-only	✓	100 %	Contexte non sérialisé transm avant réservation dans les deu
Démarrage synchrone, single-flight et sans mutation sur refus	✓	98 %	<code>activeIdentityRef</code> est ac commandes appellent <code>start</code> toutefois relâché par <code>source</code>
Identité <code>streamInstanceId</code> à travers callbacks, Queue A et Queue B	✓	98 %	Les entrées, suffixes, chunks c <code>await</code> sont gardés par l'ident
Aucun changement d'algorithme de Queue A/B, du verrou ou des sanitizers	✓	100 %	Le diff de Queue A/B ajoute l les sanitizers n'ont pas été mo
Réservations synchrones, promotion par handle opaque et abort précoce	✓	100 %	Implémentés dans le registre l

Fonctionnalité technique demandée	État	Estimation	V
Époques, retraites structurées, tombstones et génération destructive	✓	98 %	Les primitives et revalidations
Claims destructifs exacts et wake-ups immuables	✓	98 %	Acquisition/libération, invalidation pour l'éviction et isolation des
Finalisation <code>reservationCancelled</code> / <code>handoffReady</code> / <code>missingBaseline</code> / <code>missingTerminal</code> / <code>stale</code>	✓	100 %	Les cinq résultats sont présents
Lifecycle lazy <code>Readable</code> partagé	✓	100 %	Priming avant callback, consommation observation de la première lecture lifecycle source partagé .
Persistance de tout <code>meta.error</code> avant append et forwarding	✓	100 %	La séquence est bien persistée <code>forward</code> dans la source Prom
Synthèse <code>SYSTEM_ERROR/INTERRUPTED</code> sans remplacement d'un terminal antérieur	✓	98 %	Les gardes sont présentes et un Orch ne crée pas un second terminal
Protocole V1 premier-header et erreurs tardives dans le body	✓	100 %	Encoder/décoder stricts, consommation header d'erreur et erreur tardive
Filtrage des enveloppes Slimbk post-header	✓	100 %	Le parseur est appelé avant le <code>cast/application AiStreamed</code> extension.
FIFO extension avant finalisation registre	✓	96 %	L'ordre normal est correct, mais peut ensuite être perdue par Q
Open live-safe sans reset Orch	✓	100 %	Open ne réhydrate pas la mémoire persistant récent.
Replay backlog + tail complet et non compressé	✓	100 %	Buffer en tableau sans limite anti-missed-wakeup.
Handoff direct normalPrompt sans second Open	✓	100 %	Exige terminal, <code>completed</code> , et <code>finalizedStreamInsta</code>
Retry normalPrompt <code>preserveLocal</code> sans lecture des champs draft	✓	100 %	Construction d'un objet assaini pas <code>promptDraft</code> , <code>promptD</code> <code>composerInteractive=tr</code>
Composer editable après armement du replay, Send verrouillé	✓	100 %	à <code>liveSessionStreaming</code> ,
Stop unique dans le compositeur	✓	100 %	Le bandeau normalPrompt ne le <code>VscTextarea</code> reste l'unique
Barrière fallback corrélée à <code>turnId</code> + <code>streamInstanceId</code>	⚠	94 %	L'ack est correctement corrélé sans effacer ; une fin complète toujours la finalisation fronter
Continuation avec UID assistant réutilisé et thinking séparé par tentative	✓	98 %	Le plan est préparé avant <code>sta</code>
Projection Activité Idle/open, Running/observer/open, Running/source/open ou closed	✓	100 %	Union partagée et mutations e
Stop détaché uniquement pour source fermée	⚠	94 %	Projection et autorisation exacte refus/échec du RPC.
Son exact par run et tentative	⚠	92 %	Déduplication correcte, mais l
Coordination panneau/source/cleanup par locator	✓	97 %	Files par locator, suppression cleanup et wake-ups sont prés
Suppression durable sous claim	✗	75 %	Le claim et l'éviction sont pré définitivement incapable de ré
Retry EasyAgents sous claim	✓	97 %	Double guard, claim adjacent, en <code>finally</code> et reconstruction <code>sourceId=turnId</code> , lifecycle
Préservation du produit EasyAgents	✓	97 %	kill et UX read-only sont cons
Programming docs	✓	100 %	Les six documents attendus or

Fonctionnalité technique demandée	État	Estimation	
Tests exhaustifs de la tranche	△	89 %	Très vaste couverture nouvelle panneau ont été supprimés sans

3) Y a-t-il des bugs ?

Conclusion : ❌ Oui — quatre bugs fonctionnels confirmés, deux violations contractuelles et deux problèmes de qualité

● Bug 1 — Une suppression réussie peut être suivie d’une résurrection du chat

Sévérité : critique — **confiance :** 97 %

Le chemin de suppression autorise une activité `source`, puis exécute :

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

Copy

```
// The dispose coordinator sees the claim and defers every destructive cleanup until
release.
closePromptPanel(locator);
...
await new ChatPersistence().deleter.deleteSessionDirsBySessionId(locator);

// Durable success retires live authority before any auxiliary Activity broadcast can
fail.
liveSession.evictLocator(locator, destructionClaim);
```

La source n’est donc pas arrêtée et drainée avant le premier delete. Son arrêt est différé au wake-up suivant la libération du claim.

Or la source applique volontairement la persistance avant l’append registre :

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

Copy

```
// Every Orch terminal error is durable before it becomes replayable or frontend-visible.
if (chunk.type === 'meta' && chunk.value.metaType === 'error') {
    await this.persistError(args.runRef, chunk.value.value);
}
this.deps.liveSession.appendChunk({
    runRef: args.runRef,
    chunk,
});
```

Après l’éviction :

1. un chunk tardif peut encore être produit par Orch ;
2. un `meta.error` est persisté **avant** que `appendChunk()` découvre que le record a été évincé ;
3. même un chunk non terminal peut provoquer l’append refusé, puis la synthèse d’un `INTERRUPTED`, lui aussi persisté avant le nouvel append ;
4. le fichier supprimé peut donc être recréé.

Une variante existe lorsqu’une réservation est encore seulement « staged » : l’éviction du registre ne retire pas immédiatement le handle de la map process-locale ; l’endpoint peut encore le consommer et primer `askStream()` avant que `registerRun()` rejette la réservation devenue stale.

Conséquence possible : l’utilisateur voit une suppression réussie, puis le chat réapparaît partiellement ou est réécrit par la source arrêtée trop tard.

● Bug 2 — Une erreur de Queue A/Queue B peut être perdue et classée `completed`

Sévérité : élevée — **confiance :** 95 %

Dans `Queue A`, une erreur de `runMerged()` rejette la chaîne interne, mais `whenIdle()` ne conserve ni ne réémet cette erreur : il ne regarde que la longueur de la file et le drapeau de drainage.

Dans `le hook de streaming`, l’erreur est ensuite seulement loguée et le résultat devient malgré tout `completed` :

[AICode/src/frontend/src/messageBus/useAiStream.hook.ts](#)

Copy

```
try {
  await streamQueue.whenIdle();
  await dispatchGateQueue.whenIdle();
  await streamQueue.waitForCommit();
} catch (error) {
  console.error(
    '[useAiStream.onComplete] queue finalization failed:',
    error,
  );
} finally {
  // ...
  setOutcome('completed');
```

Conséquences possibles :

- faux outcome `completed` alors qu'un chunk n'a pas été appliqué ;
- terminal ou ack fallback perdu ;
- handoff fondé sur un état frontend incomplet ;
- Promise interne rejetée sans consommateur ;
- absence de `finalizedStreamInstanceId` si aucun token/thinking/meta n'a survécu ;
- masque du compositeur potentiellement non libéré dans ce chemin extrême.



Bug 3 — L'état sonore exact peut fuir après une finalisation `stale`

Sévérité : moyenne — confiance : 98 %

`onAiStreamStop()` retire seulement l'identifiant de tentative de l'ensemble actif ; il ne supprime pas l'entrée par run de la table d'état sonore.

Or le coordinateur de panneau ne nettoie le run sonore que pour quatre résultats non-stale. Le test associé exige même explicitement l'absence de cleanup en cas `stale`.

Scénario concret :

- source classique démarrée dans son panneau d'origine ;
- suppression destructive pendant l'exécution ;
- le descripteur du coordinateur peut être absent, car ce panneau avait été ouvert classiquement avant le démarrage ;
- éviction du run ;
- handler source finalisé avec `stale` ;
- tentative sonore retirée, mais entrée de run jamais supprimée.

Le cleanup par run est exact et ne pourrait pas supprimer l'état d'un successeur ayant un autre `sourceId` : il peut donc être effectué sans danger.



Bug 4 — Le Stop détaché devient irrécupérable après refus ou erreur du RPC

Sévérité : moyenne — confiance : 100 %

Le frontend déclenche le RPC sans observer son résultat :

[AICode/src/frontend/src/app/sidebar/sidebar-activity/SidebarActivityQuery.tsx](#)

Copy

```
const handleStop = useCallback(() : void => {
  void bus.call('sidebar->ext:stopDetachedQuery', {
    runRef: item.runRef,
  });
}, [item.runRef]);
...
disableAfterClick
```

Le composant de design system rend `disableAfterClick` permanent pour l'instance montée.

Ainsi :

- si le backend répond { `accepted: false` }, `stopRequested` reste faux mais le bouton local reste désactivé ;

- si le RPC rejette, la Promise fire-and-forget n'a pas de `catch` et le bouton reste désactivé ;
- le bouton ne redevient actif qu'après disparition/remontage du sous-arbre ou remplacement de `sourceId`.



Bug contractuel 5 — `sourceId` reste injectable par un caller Prompt

Sévérité contractuelle : moyenne — confiance : 100 %

La spécification exige que `sourceId` soit créé **après** acceptation par `start()`.

Le contrat autorise pourtant un override :

[AICode/src/frontend/src/messageBus/AiStreamStartArgs.type.ts](#)

Copy

```
| {  
  kind: 'promptQuery';  
  turnId: string;  
  sourceId?: string;  
  storageScopeRelPath: ChatSessionLocator['storageScopeRelPath'];  
  aiQuery: AiQuery;  
  aiQueryOptions: AiQueryOptions;  
}
```

Et `start()` utilise réellement cette valeur si elle est fournie.

Les call sites de production actuels ne l'utilisent pas, mais le type public permet à un futur caller :

- de réutiliser un `sourceId` ;
- d'affaiblir le contrat « une invocation acceptée = une source fraîche » ;
- de rencontrer artificiellement une tombstone process-lifetime.

C'est précisément une **relaxation d'une contrainte de type stricte**, qui doit être signalée selon la politique de vérification.



Bug contractuel 6 — Reply masque son résultat d'acceptation

Sévérité contractuelle : faible à moyenne — confiance : 100 %

Le hook de commandes renvoie bien `AiStreamStartResult`, mais l'interface publique de `usePromptRunController` déclare :

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

Copy

```
handleReplyButton: (label: string) => void;
```

La spécification exigeait pourtant que `Send`, `SPECIF`, `Retry`, `Edit`, `Compact`, `Reply` et `Continue` retournent tous le résultat strict au caller.

Le comportement runtime est actuellement correct parce que la valeur existe, mais **le contrat TypeScript public la supprime**.



Problème de qualité 7 — Compatibilité intermédiaire et type inutilisé

Confiance : 100 %

- [le runtime Activité](#) conserve une méthode `onStart()` explicitement commentée comme façade de compatibilité « jusqu'à la migration step-2 », sans call site trouvé.
- [les types de démarrage du stream](#) exporte `AiStreamActiveState`, mais aucun call site ne l'utilise ; le hook redéfinit d'autres types actifs internes.

Cela contredit le caractère **atomique/final** de la tranche et l'interdiction de laisser du code de transition ou de la dette technique.



Problème de couverture 8 — Plusieurs protections historiques du panneau ne sont plus testées

Confiance : 100 %

Le test [openPanel.util.test.ts](#) a été fortement réécrit. Les anciens scénarios portant notamment sur :

- l'ouverture et la réutilisation des panneaux EasyAgents Item ;
- le retargeting d'onglet ;
- `detachItemView` ;

- `invalidateItemBootstrap` ;
 - `detachHistoryView` ;
 - certains ordres de cleanup ;
- ont disparu de ce test. Les fonctions sont encore mockées, mais ne sont plus affirmées dans la version courante.
Les tests dédiés au nouveau coordinateur sont riches, mais ils ne remplacent pas totalement cette couverture non-Prompt.

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

Conclusion sur le code source : très bonne architecture, mais livraison encore trop risquée sans corrections

Critère	Note	Avis
Architecture et séparation des responsabilités	9/10	Les nouvelles classes source, coordinateur, parseur, adaptateurs et services de claims sont bien séparées.
Exactitude des identités asynchrones	9/10	<code>sourceId</code> , <code>streamInstanceId</code> , <code>replayOpenAttemptId</code> , <code>runId</code> , époque et génération destructive sont bien distingués.
Lisibilité et commentaires	9/10	Les invariants critiques sont abondamment matérialisés dans le code.
Robustesse nominale	9/10	Le chemin fermer → continuer → rouvrir → replay → handoff est cohérent.
Robustesse destructive	6/10	La suppression d'une source active contient une vraie fenêtre de résurrection.
Robustesse frontend aux erreurs internes	7/10	Les erreurs de queue ne remontent pas correctement au résultat de tentative.
Typage strict	8/10	Globalement excellent, avec deux relâchements précis signalés. Plusieurs hubs restent très volumineux : Prompt , useAiStream et le registre
Maintenabilité	7/10	liveSession dépassent chacun largement le millier de lignes ou s'en approchent.
Couverture de tests	8/10	Impressionnante sur les nouvelles courses, mais quelques garanties historiques ont été retirées.

Conclusion sur la spécification : exceptionnellement précise, mais trop rigide et pas réellement auto-cohérente

Points très positifs :

- excellente définition des identités et des propriétaires ;
- scénarios de course explicités ;
- séparation entre terminal data et transport ;
- exigences d'isomorphisme très fortes ;
- exigences précises sur les claims, wake-ups et finalisations ;
- risque de régression de **92 %** correctement assumé.

Points faibles :

1. La « liste exacte » de 160 fichiers est factuellement incomplète : l'implémentation correcte des identités de queue et du transport Slimbk a nécessité **22 fichiers source supplémentaires**.
2. La tranche est si large qu'une erreur locale peut traverser cinq couches avant d'être visible.
3. La suppression active décrit bien l'exclusion mutuelle, mais pas une barrière définitive empêchant les écritures post-delete.
4. La spécification insiste sur la finalisation frontend exacte, mais ne définit pas clairement le comportement d'une Queue A/B rejetée ou d'un stream accepté entièrement vide.
5. Le niveau de détail de la liste de fichiers donne une illusion d'exhaustivité alors que les invariants et tests de propriétés sont ici plus fiables que le manifeste.

Mon avis global : **le design est excellent et nettement supérieur à une implémentation opportuniste, mais cette ampleur exige encore une passe corrective dédiée aux chemins destructifs et aux échecs internes.**

5) Pouvez-vous résumer le rapport dans un tableau ?

TL-DR

Question	Réponse courte	Estimation	Verdict
Toute la spécification est-elle implémentée ?	Tous les fichiers demandés sont présents, mais certains contrats et garanties restent	92 %	⚠

Question	Réponse courte	Estimation	Verdict
Le scénario nominal fonctionne-t-il ?	incomplets. Oui : fermeture, poursuite extension, Activité, réouverture, replay, Stop et handoff sont câblés.	96 %	✓
Les contrats exacts sont-ils respectés ?	Presque ; <code>sourceId?</code> et le type public de Reply sont deux écarts confirmés.	91 %	⚠
Les courses principales sont-elles contrôlées ?	La majorité oui ; la suppression d'une source active reste dangereuse.	84 %	✗
Le replay est-il sans perte ni doublon dans le chemin nominal ?	Oui, avec buffer complet, curseur, exact lease et gardes de tentative.	97 %	✓
Queue A/Queue B restent-elles algorithmiquement intactes ?	Oui, les modifications portent bien sur l'identité et les resets de génération.	100 %	✓
Les erreurs Queue A/B sont-elles correctement propagées ?	Non, une erreur peut être loguée puis classée <code>completed</code> .	65 %	✗
La suppression durable est-elle sûre ?	Non lorsque la source normale est encore active : une écriture tardive peut recréer le chat.	75 %	✗
Le Stop détaché est-il fiable ?	Exact en nominal, mais non récupérable après refus/erreur RPC.	90 %	⚠
Le son est-il correctement déduplicué ?	Oui en nominal ; un run <code>stale</code> peut rester en mémoire.	92 %	⚠
EasyAgents est-il préservé ?	Oui dans les chemins inspectés.	97 %	✓
Les tests sont-ils suffisants ?	Très larges, mais avec une perte de couverture historique sur le panneau central.	89 %	⚠
Livraison production recommandée maintenant ?	Pas avant correction des bugs de suppression et de queue.	—	✗

6) Quelle est la conclusion en une phrase ?

L'implémentation réalise remarquablement bien l'architecture `liveSession` demandée, mais elle ne doit pas encore être considérée comme totalement conforme ni sûre en production tant que la résurrection post-delete et la perte des erreurs de queue ne sont pas corrigées.

7) Si des corrections sont nécessaires, quels lots de corrections peut-on coder directement ?

Lot 1 — 🔥 Sécuriser la suppression destructive

- Empêcher toute source ou réservation staged de démarrer ou de continuer à persister après le commit logique de suppression.
- Fermer la fenêtre entre le premier delete et l'arrêt effectif de la source.
- Invalider la réservation staged avant `askStream()`.
- Attendre la convergence de la source, puis effectuer une suppression finale garantissant l'absence de réécriture.
- Ajouter les tests de :
 - réservation staged pendant delete ;
 - chunk tardif après éviction ;
 - terminal synthétique post-éviction ;
 - absence de résurrection sur disque.

Lot 2 — 🗨️ Fiabiliser Queue A, Queue B et la finalisation frontend

- Conserver la première erreur de drain dans chaque queue.
- Faire rejeter `whenIdle()` lorsque la tentative propriétaire a connu une erreur.
- Ne jamais publier `completed` après une erreur de queue.
- Router cette issue vers `transportError` avec terminal unique.
- Publier `finalizedStreamInstanceId` pour toute tentative acceptée, y compris sans token/thinking/meta.
- Tester erreur de Queue A, erreur de Queue B, stream vide et démasquage fallback.

Lot 3 — 🔗 Corriger les contrats, le Stop et le cleanup sonore

- Retirer `sourceId?` de la branche `promptQuery` et toujours générer cette identité après acceptation.

- Exposer `AiStreamStartResult` dans le type public de `handleReplyButton`.
- Attendre le résultat du `Stop` détaché et réactiver le bouton après refus/erreur.
- Éliminer les rejets `fire-and-forget` non observés.
- Nettoyer l'état sonore exact même lorsque la finalisation est `stale`.
- Retirer la façade `onStart()` inutilisée et résoudre le type mort `AiStreamActiveState`.

Lot 4 — 🧩 Restaurer la couverture de non-régression

- Restaurer ou recréer les assertions historiques du panneau central :
- panneaux `EasyAgents Item` ;
- `retargeting` d'onglet ;
- panneau `Batch history` ;
- `detach/invalidation` ;
- ordres de `cleanup`.
- Ajouter les tests des quatre bugs fonctionnels ci-dessus.
- Conserver les nouveaux tests du coordinateur sans remplacer les garanties historiques.

Lot 5 — 🛠️ Correction complète recommandée

- Appliquer les lots 1 à 4 ensemble.
- Mettre à jour les `programming docs` affectées.
- Supprimer le code mort et les compatibilités intermédiaires.
- Conserver les algorithmes existants de `Queue A/Queue B` ; ne modifier que leur propagation d'erreurs et leur finalisation exacte.

🔥 Coder la suppression atomique

🔄 Coder la fiabilité du stream

🔗 Coder contrats `Stop` et son

🧪 Restaurer la couverture tests

🛠️ Coder toutes les corrections

</ASSISTANT>

<USER>

[Coder toutes les corrections]

</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](#), 📄 [isomorphism-invariant-rule-between-streaming-and-rehydration-of-chat-sessions.programming-doc-readme.md](#)

🔧 ✅ scanning git: 🗑️ "État Git avant correction"

🔧 ✅ scanning recent files

🔧 ✅ reading project: 🔍 h "[API destructive live](#)", 🔍 h "[Suppression chat active](#)", 🔍 h "[Lifecycle handler source](#)", 🔍 h "[Tests suppression destructive](#)", 🗑️ "[Contrats destructifs](#)", 📄 [PromptThreadCleanup.manager.ts](#) [L1-L240], 🔍 h "[AwaitStop semantics](#)", 🔍 e "[Cleanup export usages](#)", 🔍 e "[Detach prompt usages](#)", 📄 [LiveSessionRecord.type.ts](#) [L1-L220], 📄 [LiveSessionRegistry.service.ts](#) [L1-L360],

📄 [runWithTimeout.util.ts](#),

📄 [useAiStreamQueue.hook.ts](#), 📄 [usePromptStreamDispatchGateQueue.hook.ts](#), 📄 [useAiStream.hook.ts](#) [L1-

L260], 📄 [useAiStream.hook.ts](#) [L600-L1020], 📄 [usePromptRunController.finalization.hook.ts](#) [L1-

L260], 📄 [usePromptRunController.hook.ts](#) [L1-L170], 📄 [usePromptRunController.hook.ts](#) [L240-

L430], 📄 [usePromptRunController.hook.ts](#) [L430-L649], 📄 [useAiStream.hook.ts](#) [L250-L620], 🔍 e "[SourceId prompt](#)

[caller](#)", 🔍 e "[État actif mort](#)", 🔍 e "[Reply public type](#)", 🔍 e "[Facade onStart](#)", 📄 [SidebarActivityQuery.tsx](#) [L1-

L240], 📄 [PromptLiveSessionPanelCoordinator.service.ts](#) [L180-L290], 📄 [openPanel.util.test.ts](#)

🔧 ✅ scanning git: 🗑️ "[Ancien test panneaux](#)"


```

- /** Exact, process-local live-session registry with leases, epochs, replay cursors, and
- retention. */
-
+
+ /** Create one exact source-settlement latch whose Promise executor installs its
+ resolver synchronously. */
+ function createSourceSettlementLatch(): LiveSessionSourceSettlementLatch {
+   let resolveSettlement: (() => void) | undefined;
+   const promise = new Promise<void>((resolve) => {
+     // Promise executors run synchronously, so the resolver is available before
+     this helper returns.
+     resolveSettlement = resolve;
+   });
+   if (!resolveSettlement) {
+     throw new Error(
+       'Live-session source settlement resolver was not initialized.',
+     );
+   }
+   return Object.freeze({ promise, resolve: resolveSettlement });
+ }
+
+ /** Exact, process-local live-session registry with leases, epochs, replay cursors, and
+ retention. */
+
-   private readonly recordsByRunKey = new Map<string, LiveSessionRecord>();
-
+   private readonly recordsByRunKey = new Map<string, LiveSessionRecord>();
+   /** Settlement latches outlive destructive eviction until the source handler itself
+ finishes. */
+   private readonly sourceSettlementByRunKey = new Map<
+     string,
+     LiveSessionSourceSettlementLatch
+   >();
+
-   this.activeRunKeyByLocatorKey.set(locatorKey, runKey);
-   this.bumpLocatorEpoch(locatorKey);
-
+   this.activeRunKeyByLocatorKey.set(locatorKey, runKey);
+   this.bumpLocatorEpoch(locatorKey);
+   const sourceSettlement = createSourceSettlementLatch();
+
-   };
-   this.retiredRunKeys.add(runKey);
-
+   };
+   this.retiredRunKeys.add(runKey);
+   this.sourceSettlementByRunKey.set(runKey, sourceSettlement);
+
-   if (this.reservationsByRunKey.get(runKey) !== reservation) {
-
+   if (this.reservationsByRunKey.get(runKey) !== reservation) {
+     // The exact owner may report cancellation after destructive eviction
+     removed its indexes.
+     this.resolveSourceSettlement(reservation.runRef);
+
-   this.rememberRetirement(reservation.runRef, 'reservationCancelled');
-
+   this.rememberRetirement(reservation.runRef, 'reservationCancelled');
+   // Cancellation is the terminal authority transition for a source that never
+   promoted.

```

```

+         this.resolveSourceSettlement(reservation.runRef);
+
-         const retirement = this.retirementByRunKey.get(runKey);
-         return retirement &&
-
+         const retirement = this.retirementByRunKey.get(runKey);
+         const outcome =
+             retirement &&
+
-             this.getLocatorEpoch(locatorKey)
-             ? retirement.outcome
-             : 'stale';
-
+             this.getLocatorEpoch(locatorKey)
+             ? retirement.outcome
+             : 'stale';
+         // A source handler may finish after destructive eviction; that completion
+         still releases waiters.
+         this.resolveSourceSettlement(runRef);
+         return outcome;
+
-         this.retireRecord(record, 'retiredMissingBaseline');
-         return 'retiredMissingBaseline';
-
+         this.retireRecord(record, 'retiredMissingBaseline');
+         this.resolveSourceSettlement(runRef);
+         return 'retiredMissingBaseline';
+
-         this.retireRecord(record, 'retiredMissingTerminal');
-         return 'retiredMissingTerminal';
-
+         this.retireRecord(record, 'retiredMissingTerminal');
+         this.resolveSourceSettlement(runRef);
+         return 'retiredMissingTerminal';
+
-         this.signal(record);
-         return 'handoffReady';
-
+         this.signal(record);
+         // All source persistence and forwarding completed before this exact
+         finalization boundary.
+         this.resolveSourceSettlement(runRef);
+         return 'handoffReady';
+
-         this.retireRecord(record, 'retiredMissingBaseline');
-         if (observerGroupInactive) this.emitObserverInactive(record);
-
+         this.retireRecord(record, 'retiredMissingBaseline');
+         this.resolveSourceSettlement(runRef);
+         if (observerGroupInactive) this.emitObserverInactive(record);
+
-         this.retireRecord(record, 'retiredMissingTerminal');
-         if (observerGroupInactive) this.emitObserverInactive(record);
-
+         this.retireRecord(record, 'retiredMissingTerminal');
+         this.resolveSourceSettlement(runRef);
+         if (observerGroupInactive) this.emitObserverInactive(record);
+

```

```

-         }
-         this.signal(record);
-
+         }
+         this.signal(record);
+         // Observer transports use the same latch so destructive callers can await
generic source release.
+         this.resolveSourceSettlement(runRef);
+

```

```

-
-     /** Read one isolated structured retirement for exact fallback decisions. */
-
+
+     /** Await the exact source transport relinquishing persistence authority, or
resolve immediately if already gone. */
+     public async waitForSourceSettlement(
+         runRef: LiveSessionRunRef,
+     ): Promise<void> {
+         this.sweep();
+         const runKey = buildLiveSessionRunKey(runRef);
+         const sourceSettlement = this.sourceSettlementByRunKey.get(runKey);
+         if (!sourceSettlement) return;
+
+         // The same latch survives reservation promotion, closing the reserve/register
observation gap.
+         await sourceSettlement.promise;
+     }
+
+     /** Read one isolated structured retirement for exact fallback decisions. */
+

```

```

-
-     /** Evict one exact record while conditionally clearing only its own active locator
pointer. */
-
+
+     /** Resolve and retire the exact source completion latch after every persistence
path has ended. */
+     private resolveSourceSettlement(runRef: LiveSessionRunRef): void {
+         const runKey = buildLiveSessionRunKey(runRef);
+         const sourceSettlement = this.sourceSettlementByRunKey.get(runKey);
+         if (!sourceSettlement) return;
+         this.sourceSettlementByRunKey.delete(runKey);
+         sourceSettlement.resolve();
+     }
+
+     /** Evict one exact record while conditionally clearing only its own active locator
pointer. */
+

```

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

Copy

Review

Accept

  **patching file:**  [LiveSessionService.ts](#)

[AICode/src/extension/src/aiTools/orch/liveSession/core/LiveSessionService.ts](#)

Copy

Review

Accept

PatchFull file

```

-         return this.registry.hasExactReservation(runRef);

```

```

-    }
-
+    return this.registry.hasExactReservation(runRef);
+    }
+
+    /** Await one exact source transport releasing every remaining persistence and
+    forwarding right. */
+    public async waitForSourceSettlement(
+        runRef: LiveSessionRunRef,
+    ): Promise<void> {
+        await this.registry.waitForSourceSettlement(runRef);
+    }
+

```

[AI Code/src/extension/src/aiTools/orch/liveSession/core/LiveSessionService.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 { SlimbkEndpoint } from
'../../../../../../sharedlib/slimbk/framework/SlimbkEndpoint.js';
-
+ import { SlimbkEndpoint } from
'../../../../../../sharedlib/slimbk/framework/SlimbkEndpoint.js';
+ import { runWithTimeout } from
'../../../../../../sharedlib/lib/concurrencyUtil/runWithTimeout.util.js';
+
- } from '../../../../../../sharedlib/types/chatTypes/ChatSessionLocator.type.js';
- import { getLiveSessionServiceForProd } from
'../../../../../../aiTools/orch/liveSession/core/LiveSessionService.js';
-
+ } from '../../../../../../sharedlib/types/chatTypes/ChatSessionLocator.type.js';
+ import type { LiveSessionRunRef } from
'../../../../../../sharedlib/types/liveSessionTypes/LiveSessionRun.type.js';
+ import {
+     getLiveSessionServiceForProd,
+     type LiveSessionService,
+ } from '../../../../../../aiTools/orch/liveSession/core/LiveSessionService.js';
+
- import { reconcileClosedQueryActivityFromLiveAuthority } from
'../../../../../../extension/panels/sidebar/SidebarStore.rehydrate.manager.js';
-
+ import { reconcileClosedQueryActivityFromLiveAuthority } from
'../../../../../../extension/panels/sidebar/SidebarStore.rehydrate.manager.js';
+
+ const SOURCE_SETTLEMENT_TIMEOUT_MS = 5_000;
+
-     return error instanceof Error ? error : new Error(String(error));
- }
-
+     return error instanceof Error ? error : new Error(String(error));
+ }
+

```

```

+ /** Abort every exact normal-Prompt source synchronously, then wait until none can
+ persist after deletion. */
+ async function stopAndAwaitSourceSettlement(args: {
+   liveSession: LiveSessionService;
+   runRefs: readonly LiveSessionRunRef[];
+ }): Promise<void> {
+   if (args.runRefs.length === 0) return;
+
+   // Claim and invoke every abort without an intervening await so reservations cannot
+   promote unchecked.
+   for (const runRef of args.runRefs) {
+     const abortClaim = args.liveSession.claimSourceAbort(runRef);
+     abortClaim?.abort();
+   }
+
+   // Wait for registry finalization/cancellation before touching disk. This is the
+   write barrier that
+   // prevents a late meta.error or synthetic terminal from recreating a successfully
+   deleted chat.
+   const settlement = await runWithTimeout(async () => {
+     await Promise.all(
+       args.runRefs.map(async (runRef) => {
+         await args.liveSession.waitForSourceSettlement(runRef);
+       })
+     );
+   }, SOURCE_SETTLEMENT_TIMEOUT_MS);
+   if (settlement.status === 'error') {
+     throw normalizeError(settlement.error);
+   }
+   if (settlement.status === 'timeout') {
+     throw new Error(
+       'ChatDeleteBlocked: active source did not relinquish persistence authority
+ in time',
+     );
+   }
+ }
+ }
+

```

```

- * Behavior:
- * - Deletion is forbidden while the related Prompt window is running.
-
+ * Behavior:
+ * - Observer-owned runs must be stopped explicitly; source-owned Prompt runs are
+ aborted and drained here.
+

```

```

-   try {
-     sourceWasActive = liveSession
-
+   try {
+     const activeSourceRunRefs = liveSession
+

```

```

-     .listActiveSnapshots()
-     .some(
-
+     .listActiveSnapshots()
+     .filter(
+

```

```

-       locator.storageScopeRelPath &&
-
+       locator.storageScopeRelPath &&
+       snapshot.ownership === 'source' &&
+

```

```

-         (snapshot.kind === 'reservation' ||
-           (snapshot.ownership === 'source' &&
-
+           (snapshot.kind === 'reservation' ||
+           (snapshot.kind === 'record' &&
+
-           !snapshot.handoffReady)),
-         );
-
+           !snapshot.handoffReady)),
+         )
+         .map((snapshot) => snapshot.runRef);
+         sourceWasActive = activeSourceRunRefs.length > 0;
+
-         closePromptPanel(locator);
-
+         closePromptPanel(locator);
+
+         // A destructive delete must stop and drain the source before the first
+         disk removal. Eviction
+         // still remains post-delete so a filesystem failure does not falsely
+         retire durable authority.
+         await stopAndAwaitSourceSettlement({
+           liveSession,
+           runRefs: activeSourceRunRefs,
+         });
+

```

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

Copy

Review

Accept

🔧 ☒ reading project: 📄 [delete.endpoint.ts](#) [L1-L160]

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

```
AICode/src/extension/src/aiTools/orch/liveSession/registry/ test /LiveSessionRegistry.service.test.ts
```

Copy

Review

Accept

PatchFull file

```
- it('applies exact abort claims once for reservations, sources, and observer
leases', () => {
-
+
+   it('releases exact source-settlement waiters on cancellation, finalization, and
post-eviction handler completion', async () => {
+     const cancelledRegistry = new LiveSessionRegistryService();
+     const cancelledReservation = reserve(cancelledRegistry, RUN_A);
+     const cancelledWait = cancelledRegistry.waitForSourceSettlement(RUN_A);
+     expect(
+       cancelledRegistry.cancelSourceReservation(cancelledReservation),
+     ).toBe(true);
+     await expect(cancelledWait).resolves.toBeUndefined();
+
+     const finalizedRegistry = new LiveSessionRegistryService();
+     const finalizedReservation = reserve(finalizedRegistry, RUN_A);
+     promote(finalizedRegistry, RUN_A, finalizedReservation);
+     finalizedRegistry.appendChunk({
+       runRef: RUN_A,
```

```

+         chunk: terminal(RUN_A),
+     });
+     const finalizedWait = finalizedRegistry.waitForSourceSettlement(RUN_A);
+     expect(finalizedRegistry.finalizeSourceAfterForwarding(RUN_A)).toBe(
+         'handoffReady',
+     );
+     await expect(finalizedWait).resolves.toBeUndefined();
+
+     const evictedRegistry = new LiveSessionRegistryService();
+     reserve(evictedRegistry, RUN_A);
+     const evictedWait = evictedRegistry.waitForSourceSettlement(RUN_A);
+     const claim = evictedRegistry.acquireLocatorDestructionClaim(LOCATOR);
+     if (!claim) throw new Error('Expected destruction claim.');
```

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

Copy

Review

Accept

  patching file: [LiveSessionService.test.ts](#)

[AI Code/src/extension/src/aiTools/orch/liveSession/core/_test_/LiveSessionService.test.ts](#)

Copy

Review

Accept

PatchFull file

```

-         expect(service.listActiveSnapshots()).toHaveLength(1);
-
+         expect(service.listActiveSnapshots()).toHaveLength(1);
+         const sourceSettlement = service.waitForSourceSettlement(RUN);
+
+
-         expect(service.finalizeSourceAfterForwarding(RUN)).toBe('handoffReady');
```

[AI Code/src/extension/src/aiTools/orch/liveSession/core/_test_/LiveSessionService.test.ts](#)

Copy

Review

Accept

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

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

Copy

Review

Accept

PatchFull file

```

-     jest.fn<LiveSessionService['listActiveSnapshots']>();
-
+     jest.fn<LiveSessionService['listActiveSnapshots']>();
```

```

+ const claimSourceAbortMock = jest.fn<LiveSessionService['claimSourceAbort']>();
+ const waitForSourceSettlementMock =
+   jest.fn<LiveSessionService['waitForSourceSettlement']>();
+
-   listActiveSnapshots: listActiveSnapshotsMock,
-
+   listActiveSnapshots: listActiveSnapshotsMock,
+   claimSourceAbort: claimSourceAbortMock,
+   waitForSourceSettlement: waitForSourceSettlementMock,
+
-   },
-   ]);
-
+   },
+   ]);
+   claimSourceAbortMock.mockReturnValue({
+     runRef,
+     abort: () => {
+       order.push('abort');
+     },
+   });
+   waitForSourceSettlementMock.mockImplementation(async () => {
+     order.push('waitForSourceSettlement');
+   });
+
-   'close',
-   'delete',
-
+   'close',
+   'abort',
+   'waitForSourceSettlement',
+   'delete',
+
-   expect(reconcileActivityMock).toHaveBeenCalledWith(runRef.locator);
-   expect(order).toEqual(['close', 'delete', 'release', 'reconcile']);
-
+   expect(reconcileActivityMock).toHaveBeenCalledWith(runRef.locator);
+   expect(order).toEqual([
+     'close',
+     'abort',
+     'waitForSourceSettlement',
+     'delete',
+     'release',
+     'reconcile',
+   ]);
+
-   'reconcile',
-   ]);
-   });
-
+   'reconcile',
+   ]);
+   });
+
+   it('never starts durable deletion when the active source cannot relinquish write
authority', async () => {
+     waitForSourceSettlementMock.mockRejectedValueOnce(
+       new Error('source settlement failed'),
+     );
+

```

```

+      await expect(
+        endpoint.syncHandler({
+          args: runRef.locator,
+        }) as Parameters<typeof endpoint.syncHandler>[0]),
+      ).rejects.toThrow('source settlement failed');
+
+      expect(deleteMock).not.toHaveBeenCalled();
+      expect(evictMock).not.toHaveBeenCalled();
+      expect(releaseMock).toHaveBeenCalledWith(claim);
+      expect(reconcileActivityMock).toHaveBeenCalledWith(runRef.locator);
+      expect(order).toEqual([
+        'close',
+        'abort',
+        'release',
+        'reconcile',
+      ]);
+    });
+  });
+
-      expect(evictMock).not.toHaveBeenCalled();
-      expect(order).toEqual(['close', 'delete', 'release']);
-
+      expect(evictMock).not.toHaveBeenCalled();
+      expect(order).toEqual([
+        'close',
+        'abort',
+        'waitForSourceSettlement',
+        'delete',
+        'release',
+      ]);
+

```

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

Copy

Review

Accept

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

[AI Code/src/frontend/src/messageBus/useAiStreamQueue.hook.ts](#)

Copy

Review

Accept

PatchFull file

```

-      /** Resolve when the queue is completely drained and no drain is in progress. */
-      whenIdle: () => Promise<void>;
-
+      /** Resolve when the queue is completely drained and no drain is in progress. */
+      whenIdle: (
+        attemptIdentity?: AiStreamAttemptIdentity,
+      ) => Promise<void>;
+      /** Forget the retained failure only after the owning attempt published its
terminal outcome. */
+      forgetAttemptFailure: (
+        attemptIdentity: AiStreamAttemptIdentity,
+      ) => void;
+
-
-      /** Compare the exact source and stream identities that delimit Queue A coalescence. */
-
+      /** One exact idle waiter that must observe the retained failure of its own stream
attempt. */

```

```

+ interface QueueIdleWaiter {
+     readonly attemptIdentity?: AiStreamAttemptIdentity;
+     readonly resolve: () => void;
+     readonly reject: (error: unknown) => void;
+ }
+
+ /** Build one collision-free key for Queue A failure ownership. */
+ function buildAttemptKey(attemptIdentity: AiStreamAttemptIdentity): string {
+     return JSON.stringify([
+         attemptIdentity.sourceId,
+         attemptIdentity.streamInstanceId,
+     ]);
+ }
+
+ /** Compare the exact source and stream identities that delimit Queue A coalescence. */
+
+
+
+ // Set of "idle" resolvers awaiting an empty queue and no active drain.
+ const idleResolversRef = useRef<Set<() => void>>(new Set());
+
+
+ // Exact failures remain observable after the internal drain promise is recovered
+ for future work.
+ const failureByAttemptKeyRef = useRef<Map<string, Error>>(new Map());
+
+ // Set of "idle" waiters awaiting an empty queue and no active drain.
+ const idleResolversRef = useRef<Set<QueueIdleWaiter>>(new Set());
+
+
+ const mergerRef = useRef<StreamChunkMerger>(new StreamChunkMerger());
+
+ const mergerRef = useRef<StreamChunkMerger>(new StreamChunkMerger());
+
+ /** Return the first failure owned by the requested attempt, or by any attempt for
+ legacy callers. */
+ const getRetainedFailure = useCallback(
+     (
+         attemptIdentity?: AiStreamAttemptIdentity,
+     ): Error | undefined => {
+         if (attemptIdentity) {
+             return failureByAttemptKeyRef.current.get(
+                 buildAttemptKey(attemptIdentity),
+             );
+         }
+         for (const failure of failureByAttemptKeyRef.current.values()) {
+             return failure;
+         }
+         return undefined;
+     },
+     [],
+ );
+
+ /** Preserve the first Queue A failure for one attempt while allowing the drain
+ scheduler to recover. */
+ const runMergedForAttempt = useCallback(
+     async (
+         chunk: AiStreamedChunk,
+         attemptIdentity: AiStreamAttemptIdentity,
+     ): Promise<void> => {
+         try {
+             await opts.runMerged(chunk, attemptIdentity);
+         } catch (error) {
+             const key = buildAttemptKey(attemptIdentity);

```

```

+         const currentAttempt = opts.getAttemptIdentity();
+         if (
+             currentAttempt !== undefined &&
+             isSameAttempt(currentAttempt, attemptIdentity) &&
+             !failureByAttemptKeyRef.current.has(key)
+         ) {
+             failureByAttemptKeyRef.current.set(
+                 key,
+                 error instanceof Error
+                     ? error
+                     : new Error(String(error)),
+             );
+         }
+         throw error;
+     }
+ },
+ [opts],
+ );
+
-         // Apply the merged item with the identity shared by every consumed
-         entry.
-         await opts.runMerged(
+         // Apply the merged item with the identity shared by every consumed
+         entry.
+         await runMergedForAttempt(
+
-         if (!item) break;
-         await opts.runMerged(item.chunk, item.attemptIdentity);
+
+         if (!item) break;
+         await runMergedForAttempt(
+             item.chunk,
+             item.attemptIdentity,
+         );
+
-         ) {
-         for (const resolve of idleResolversRef.current) {
+         ) {
+         for (const waiter of idleResolversRef.current) {
+
-         try {
-             resolve();
+
+         try {
+             const failure = getRetainedFailure(
+                 waiter.attemptIdentity,
+             );
+             if (failure !== undefined) waiter.reject(failure);
+             else waiter.resolve();
+
-         }
-         }, [clearDrainTimer, opts, throttleEnabled, throttleMs]);
+
+         }, [
+             clearDrainTimer,
+             getRetainedFailure,
+             runMergedForAttempt,

```

```

+         throttleEnabled,
+         throttleMs,
+     ]));
+
-     drainScheduledRef.current = true;
-     tailRef.current = tailRef.current.then(runDrain, runDrain);
-
+     drainScheduledRef.current = true;
+     const previousTail = tailRef.current;
+     tailRef.current = (async (): Promise<void> => {
+         // Internal drain rejection is represented by failureByAttemptKeyRef, not
by an unobserved tail.
+         try {
+             await previousTail;
+         } catch {
+             // A previous implementation may have left a rejected tail; recovery
remains deterministic.
+         }
+         try {
+             await runDrain();
+         } catch {
+             // The exact failure is retained for whenIdle(attemptIdentity).
+         }
+     })();
+
- /** Resolve when all queued items are fully applied and no drain is in progress. */
- const whenIdle = useCallback(() : Promise<void> => {
-     if (queueRef.current.length === 0 && !drainingRef.current) {
-         return Promise.resolve();
-     }
-     return new Promise<void>((resolve) => {
-         idleResolversRef.current.add(resolve);
-     });
- }, []);
-
+ /** Resolve when all queued items are fully applied and no drain is in progress. */
+ const whenIdle = useCallback(
+     (attemptIdentity?: AiStreamAttemptIdentity): Promise<void> => {
+         if (queueRef.current.length === 0 && !drainingRef.current) {
+             const failure = getRetainedFailure(attemptIdentity);
+             return failure === undefined
+                 ? Promise.resolve()
+                 : Promise.reject(failure);
+         }
+         return new Promise<void>((resolve, reject) => {
+             idleResolversRef.current.add({
+                 attemptIdentity,
+                 resolve,
+                 reject,
+             });
+         });
+     },
+     [getRetainedFailure],
+ );
+
+ /** Retire one exact failure only after the stream owner consumed it into an
outcome. */
+ const forgetAttemptFailure = useCallback(
+     (attemptIdentity: AiStreamAttemptIdentity): void => {
+         failureByAttemptKeyRef.current.delete(
+             buildAttemptKey(attemptIdentity),
+         );
+     };

```

```

+         },
+         [],
+     );
+
-         whenIdle,
-
+         whenIdle,
+         forgetAttemptFailure,
+
-     }),
-     [enqueueChunk, enqueueChunkForAttempt, whenIdle],
-
+     }),
+     [
+         enqueueChunk,
+         enqueueChunkForAttempt,
+         forgetAttemptFailure,
+         whenIdle,
+     ],
+

```

[AICode/src/frontend/src/messageBus/useAiStreamQueue.hook.ts](#)

Copy

Review

Accept

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

[AICode/src/frontend/src/messageBus/usePromptStreamDispatchGateQueue.hook.ts](#)

Error

([what is it?](#))

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

[AICode/src/frontend/src/messageBus/usePromptStreamDispatchGateQueue.hook.ts](#)

Error

([what is it?](#))

  reading project:  [usePromptStreamDispatchGateQueue.hook.ts](#) [L250-L330]

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

[AICode/src/frontend/src/messageBus/usePromptStreamDispatchGateQueue.hook.ts](#)

Copy

Review

Accept

PatchFull file

```

-
-     /** Resolve when the queue is fully drained and no drain is in progress (test
-     helper). */
-     whenIdle: () => Promise<void>;
-
+
+     /** Resolve when the queue is fully drained and surface the retained failure of the
+     requested attempt. */
+     whenIdle: (
+         attemptIdentity?: AiStreamAttemptIdentity,
+     ) => Promise<void>;
+
+     /** Forget the retained failure only after the owning attempt published its
+     terminal outcome. */

```

```

+ forgetAttemptFailure: (
+     attemptIdentity: AiStreamAttemptIdentity,
+ ) => void;
+
-
- /** Narrowed meta.tool_calls chunk for safe access inside the gate. */
-
+
+ /** One exact idle waiter that must observe the retained Queue B failure of its stream
+ attempt. */
+ interface GateIdleWaiter {
+     readonly attemptIdentity?: AiStreamAttemptIdentity;
+     readonly resolve: () => void;
+     readonly reject: (error: unknown) => void;
+ }
+
+ /** Build one collision-free key for Queue B failure ownership. */
+ function buildAttemptKey(attemptIdentity: AiStreamAttemptIdentity): string {
+     return JSON.stringify([
+         attemptIdentity.sourceId,
+         attemptIdentity.streamInstanceId,
+     ]);
+ }
+
+ /** Narrowed meta.tool_calls chunk for safe access inside the gate. */
+
-
- // Set of "idle" resolvers awaiting an empty queue and no active drain.
- const idleResolversRef = useRef<Set<() => void>>(new Set());
-
+
+ // Exact dispatch failures survive the recovered internal tail until the stream
owner classifies them.
+ const failureByAttemptKeyRef = useRef<Map<string, Error>>(new Map());
+
+ // Set of "idle" waiters awaiting an empty queue and no active drain.
+ const idleResolversRef = useRef<Set<GateIdleWaiter>>(new Set());
+
-
- PromptStreamLockStore.getLockedCallId(),
- );
-
+
+ PromptStreamLockStore.getLockedCallId(),
+ );
+
+ /** Return the first failure owned by the requested attempt, or by any attempt for
legacy callers. */
+ const getRetainedFailure = useCallback(
+     (
+         attemptIdentity?: AiStreamAttemptIdentity,
+     ): Error | undefined => {
+         if (attemptIdentity) {
+             return failureByAttemptKeyRef.current.get(
+                 buildAttemptKey(attemptIdentity),
+             );
+         }
+         for (const failure of failureByAttemptKeyRef.current.values()) {
+             return failure;
+         }
+         return undefined;
+     },
+     [],
+ );

```

```

+
+ /** Dispatch one exact Queue B item while retaining the first failure for terminal
+ classification. */
+ const dispatchForAttempt = useCallback(
+   async (
+     chunk: AiStreamedChunk,
+     attemptIdentity: AiStreamAttemptIdentity,
+   ): Promise<void> => {
+     try {
+       await opts.dispatchMerged(chunk, attemptIdentity);
+     } catch (error) {
+       const key = buildAttemptKey(attemptIdentity);
+       if (
+         isSameAttemptIdentity(
+           opts.getAttemptIdentity(),
+           attemptIdentity,
+         ) &&
+         !failureByAttemptKeyRef.current.has(key)
+       ) {
+         failureByAttemptKeyRef.current.set(
+           key,
+           error instanceof Error
+             ? error
+             : new Error(String(error)),
+         );
+       }
+       throw error;
+     }
+   },
+   [opts],
+ );
+

```

```

-   syntheticCloseStateRef.current = null;
-   for (const resolve of idleResolversRef.current) {
-     resolve();
-

```

```

+   syntheticCloseStateRef.current = null;
+   for (const waiter of idleResolversRef.current) {
+     waiter.resolve();
+

```

```

-   }
-   idleResolversRef.current.clear();
-
+   }
+   idleResolversRef.current.clear();
+   failureByAttemptKeyRef.current.clear();
+

```

```

-   ) {
-     for (const resolve of idleResolversRef.current) {
-
+   ) {
+     for (const waiter of idleResolversRef.current) {
+

```

```

-       try {
-         resolve();
-
+       try {
+         const failure = getRetainedFailure(
+           waiter.attemptIdentity,
+         );
+         if (failure !== undefined) waiter.reject(failure);
+

```

```

+         else waiter.resolve();
+
-         idleResolversRef.current.clear();
-     }
- }, []);
+         idleResolversRef.current.clear();
+     }
+ }, [getRetainedFailure]);
+
-
-     await opts.dispatchMerged(gated, pending.attemptIdentity);
-
+     await dispatchForAttempt(gated, pending.attemptIdentity);
+
-
-     await opts.dispatchMerged(deferred.chunk, deferred.attemptIdentity);
-
+     try {
+         await dispatchForAttempt(
+             deferred.chunk,
+             deferred.attemptIdentity,
+         );
+     } catch (error) {
+         // A failed deferred replay has already released Queue A backpressure.
Remove the exact
+         // failed entry so whenIdle can surface its retained error instead of
hanging forever.
+         if (deferredWhileLockedRef.current[0] === deferred) {
+             deferredWhileLockedRef.current.shift();
+         }
+         throw error;
+     }
+
-
-     await opts.dispatchMerged(gated, head.attemptIdentity);
-
+     await dispatchForAttempt(gated, head.attemptIdentity);
+
-
-     }
-     await opts.dispatchMerged(head.chunk, head.attemptIdentity);
-
+     }
+     await dispatchForAttempt(head.chunk, head.attemptIdentity);
+
-
-     deferHeadItem,
-
+     deferHeadItem,
+     dispatchForAttempt,
+
-
-     // Serialize drains by chaining onto the tail promise.
-     tailRef.current = tailRef.current.then(runDrain, runDrain);
-
+

```

```

+         // Serialize drains while converting internal rejection into the retained exact
failure channel.
+         const previousTail = tailRef.current;
+         tailRef.current = (async (): Promise<void> => {
+             try {
+                 await previousTail;
+             } catch {
+                 // A previous implementation may have left a rejected tail; recovery
remains deterministic.
+             }
+             try {
+                 await runDrain();
+             } catch {
+                 // The exact failure is retained for whenIdle(attemptIdentity).
+             }
+         })();
+
-
-     /** Resolve when all queued items are fully dispatched and no drain is in progress.
*/
-     const whenIdle = useCallback(() : Promise<void> => {
-         if (
-             queueRef.current.length === 0 &&
-             deferredWhileLockedRef.current.length === 0 &&
-             !drainingRef.current &&
-             pendingToolCallsSuffixRef.current === null
-         ) {
-             return Promise.resolve();
-         }
-         return new Promise<void>((resolve) => {
-             idleResolversRef.current.add(resolve);
-         });
-     }, []);
+
+     /** Resolve when all queued items are fully dispatched, or reject with the exact
retained failure. */
+     const whenIdle = useCallback(
+         (attemptIdentity?: AiStreamAttemptIdentity): Promise<void> => {
+             if (
+                 queueRef.current.length === 0 &&
+                 deferredWhileLockedRef.current.length === 0 &&
+                 !drainingRef.current &&
+                 pendingToolCallsSuffixRef.current === null
+             ) {
+                 const failure = getRetainedFailure(attemptIdentity);
+                 return failure === undefined
+                     ? Promise.resolve()
+                     : Promise.reject(failure);
+             }
+             return new Promise<void>((resolve, reject) => {
+                 idleResolversRef.current.add({
+                     attemptIdentity,
+                     resolve,
+                     reject,
+                 });
+             });
+         },
+         [getRetainedFailure],
+     );
+
+     /** Retire one exact failure only after the stream owner consumed it into an
outcome. */

```


Copy
Review
Accept

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

[AI Code/src/frontend/src/messageBus/useAiStream.hook.ts](#)

Copy
Review
Accept

PatchFull file

```
- type ActiveAiStreamArgs =
-   | (Omit<Extract<AiStreamStartArgs, { kind: 'promptQuery' }>, 'sourceId') & {
-
+ type ActiveAiStreamArgs =
+   | (Extract<AiStreamStartArgs, { kind: 'promptQuery' }> & {
+
-   );
- }
-
+   );
+ }
+
+ /** Normalize queue and transport failures before they are retained in a deterministic
+ terminal outcome. */
+ function normalizeStreamFailure(error: unknown): Error {
+   return error instanceof Error ? error : new Error(String(error));
+ }
+
+ /** Render ordered queue failures into one durable diagnostic without losing individual
+ stacks. */
+ function formatStreamFailures(errors: readonly Error[]): string {
+   return errors
+     .map((error, index) => {
+       const details = error.stack ?? error.message;
+       return `Failure ${index + 1}: ${details}`;
+     })
+     .join('\n');
+ }
+
-   streamQueueRef.current = streamQueue;
-
+   streamQueueRef.current = streamQueue;
+
+   /** Drain both serialized queues and retain every exact failure for terminal
+ outcome classification. */
+   const drainAttemptQueues = useCallback(
+     async (
+       attemptIdentity: AiStreamAttemptIdentity,
+     ): Promise<Error[]> => {
+       const errors: Error[] = [];
+       try {
+         await streamQueue.whenIdle(attemptIdentity);
+       } catch (error) {
+         errors.push(normalizeStreamFailure(error));
+       }
+       try {
+         await dispatchGateQueue.whenIdle(attemptIdentity);
+       } catch (error) {
+         errors.push(normalizeStreamFailure(error));
+       }
+     }
  )
```

```

+         return errors;
+     },
+     [dispatchGateQueue, streamQueue],
+ );
+
+ /** Clear only the exact consumed failure so successor attempts cannot inherit
+ stale queue state. */
+ const forgetAttemptQueueFailures = useCallback(
+     (attemptIdentity: AiStreamAttemptIdentity): void => {
+         streamQueue.forgetAttemptFailure(attemptIdentity);
+         dispatchGateQueue.forgetAttemptFailure(attemptIdentity);
+     },
+     [dispatchGateQueue, streamQueue],
+ );
+
+ /** Route one synthetic transport terminal through Queue A/B after their retained
+ failure is acknowledged. */
+ const enqueueSyntheticTransportError = useCallback(
+     (args: {
+         identity: ActiveAiStreamIdentity;
+         details: string;
+     }): void => {
+         streamQueue.enqueueChunkForAttempt(
+             {
+                 turnId: args.identity.runRef.turnId,
+                 type: 'meta',
+                 value: {
+                     metaType: 'error',
+                     value: {
+                         reason: AiStopErrorReason.SYSTEM_ERROR,
+                         message: SYSTEM_ERROR_CANONICAL_TEXT,
+                         details: args.details,
+                     },
+                 },
+             },
+             args.identity.attemptIdentity,
+         );
+     },
+     [streamQueue],
+ );
+
+ const reset = useCallback(() : void => {
+
+ const reset = useCallback(() : void => {
+     const activeAttemptIdentity =
+         activeIdentityRef.current?.attemptIdentity;
+     if (activeAttemptIdentity) {
+         // Manual hydration/reset consumes the old attempt without leaving retained
+         queue errors behind.
+         forgetAttemptQueueFailures(activeAttemptIdentity);
+     }
+
+ }, [
+     dispatchGateQueue,
+
+ }, [
+     dispatchGateQueue,
+     forgetAttemptQueueFailures,
+
+
+     turnId: args.turnId,
+     sourceId:
+     args.sourceId ??

```

```

-         `source:${uidGenerator.newUniqueId}` ,
-
+         turnId: args.turnId,
+         // Normal Prompt callers cannot inject source identity; every
accepted invocation is fresh.
+         sourceId: `source:${uidGenerator.newUniqueId}` ,
+
-
-         // Finalize only after the queue drained and one commit turn has
passed.
-         void (async () => {
-
+         // Finalize only after the queue drained and one commit turn has
passed.
+         void (async () => {
+             const queueFailures = await drainAttemptQueues(
+                 identityAtEntry.attemptIdentity,
+             );
+             if (
+                 queueFailures.length > 0 &&
+                 !terminalDataSeenRef.current &&
+                 isSameStreamAttempt(
+                     activeIdentityRef.current,
+                     identityAtEntry.attemptIdentity,
+                 )
+             ) {
+                 // A failed queue item is already removed. Consume its retained
failure, then route one
+                 // canonical terminal through the same queues without
duplicating prior terminal data.
+                 forgetAttemptQueueFailures(
+                     identityAtEntry.attemptIdentity,
+                 );
+                 enqueueSyntheticTransportError({
+                     identity: identityAtEntry,
+                     details: formatStreamFailures(queueFailures),
+                 });
+                 queueFailures.push(
+                     ...(await drainAttemptQueues(
+                         identityAtEntry.attemptIdentity,
+                     )),
+                 );
+             }
+
-
-         try {
-             await streamQueue.whenIdle();
-             await dispatchGateQueue.whenIdle();
-
+         try {
-
-         } catch (error) {
-             console.error(
-                 '[useAiStream.onComplete] queue finalization failed:',
-                 error,
-             );
-
+         } catch (error) {
+             queueFailures.push(normalizeStreamFailure(error));
+
-
-         // Only the still-owning attempt may reset global lock/gate
state and publish completion.
-

```

```

+ // Only the still-owning attempt may reset global lock/gate
state and publish completion.
+ forgetAttemptQueueFailures(
+     identityAtEntry.attemptIdentity,
+ );
+
- dispatchGateQueue.resetInternalState();
- setOutcome('completed');
-
+ dispatchGateQueue.resetInternalState();
+ if (queueFailures.length > 0) {
+     console.error(
+         '[useAiStream.onComplete] queue finalization
failed:',
+         new AggregateError(
+             queueFailures,
+             'Stream completion queues failed for the exact
accepted attempt.',
+         ),
+     );
+ }
+ setOutcome(
+     queueFailures.length > 0
+     ? 'transportError'
+     : 'completed',
+ );
+
- return;
- const errMsg: string = `[useAiStream] stream error: ${JSON.stringify(
-     err,
- )}`;
-
+ return;
+ const transportFailure = normalizeStreamFailure(err);
+ const errMsg: string = `[useAiStream] stream error: ${
+     transportFailure.stack ?? transportFailure.message
+ }`;
+
-
- // Publish a meta:error via the queue so the runner can append the
canonical suffix exactly once.
- const currentTurnId = identityAtEntry.runRef.turnId;
-
-
+
- // Wait until all items are applied and one commit turn passed, then
flip streaming off.
- void (async () => {
-
+ // Wait until all items are applied and one commit turn passed, then
flip streaming off.
+ void (async () => {
+     // Drain terminal data already accepted by Queue A/B before
deciding whether a synthetic error is needed.
+     const queueFailures = await drainAttemptQueues(
+         identityAtEntry.attemptIdentity,
+     );
+
- try {

```

```

- // Drain terminal data already accepted by Queue A/B before
deciding whether a synthetic error is needed.
-         await streamQueue.whenIdle();
-         await dispatchGateQueue.whenIdle();
-
+         try {
+
-             if (!terminalDataSeenRef.current) {
-                 streamQueue.enqueueChunkForAttempt(
-                     {
-                         turnId: currentTurnId,
-                         type: 'meta',
-                         value: {
-                             metaType: 'error',
-                             value: {
-                                 reason: AiStopErrorReason.SYSTEM_ERROR,
-                                 message:
-                                     SYSTEM_ERROR_CANONICAL_TEXT,
-                                 details: errMsg,
-                             },
-                         },
-                     },
-                 ),
-
+                 if (!terminalDataSeenRef.current) {
+                     // Queue failures are consumed before retrying the
canonical transport terminal.
+                     forgetAttemptQueueFailures(
+
-                     );
-                     await streamQueue.whenIdle();
-                     await dispatchGateQueue.whenIdle();
-
+                     );
+                     enqueueSyntheticTransportError({
+                         identity: identityAtEntry,
+                         details: errMsg,
+                     });
+                     queueFailures.push(
+                         ...(await drainAttemptQueues(
+                             identityAtEntry.attemptIdentity,
+                         )),
+                     );
+
-                 } catch (error) {
-                     console.error(
-                         '[useAiStream.onError] queue finalization failed:',
-                         error,
-                     );
-
+                 } catch (error) {
+                     queueFailures.push(normalizeStreamFailure(error));
+
-                     // Stale errors cannot reset successor queues or append a
second terminal.
-
+                     // Stale errors cannot reset successor queues or append a
second terminal.
+                     forgetAttemptQueueFailures(
+                         identityAtEntry.attemptIdentity,
+                     );
+

```

```

-         dispatchGateQueue.resetInternalState();
-         setOutcome('transportError');
-
+         dispatchGateQueue.resetInternalState();
+         if (queueFailures.length > 0) {
+             console.error(
+                 '[useAiStream.onError] queue finalization failed:',
+                 new AggregateError(
+                     queueFailures,
+                     'Transport-error queues failed for the exact
accepted attempt.',
+                 ),
+             );
+         }
+         setOutcome('transportError');
+

```

```

-         [
-             dispatchGateQueue,
-
+         [
+             dispatchGateQueue,
+             drainAttemptQueues,
+             enqueueSyntheticTransportError,
+             forgetAttemptQueueFailures,
+

```

[AI Code/src/frontend/src/messageBus/useAiStream.hook.ts](#)

Copy

Review

Accept

  reading project:  [useAiStream.hook.ts](#) [L520-L1060],  [useAiStream.hook.ts](#) [L1060-L1165]

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

[AI Code/src/frontend/src/messageBus/useAiStream.hook.ts](#)

Grouped

([what is it?](#))

  patching file:  [usePromptRunController.finalization.hook.ts](#)

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

Copy

Review

Accept

PatchFull file

```

- import { useEffect } from 'react';
- import type { Dispatch, RefObject, SetStateAction } from 'react';
-
+ import { useCallback, useEffect, useRef } from 'react';
+ import type { Dispatch, RefObject, SetStateAction } from 'react';
+

```

```

- }: UsePromptRunControllerFinalizationParams): void {
-
+ }: UsePromptRunControllerFinalizationParams): void {
+     const lastPublishedStreamInstanceIdRef = useRef<string | undefined>(
+         undefined,
+     );
+
+     /** Publish each accepted stream finalization exactly once, including empty
completed transports. */

```


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

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

Copy

Review

Accept

PatchFull file

```
-      handleStopFromComposer: () => Promise<boolean>;
-      handleReplyButton: (label: string) => void;
-
+      handleStopFromComposer: () => Promise<boolean>;
+      /** Submit one canned Reply button label and expose the same synchronous acceptance
+       * contract as Send. */
+      handleReplyButton: (label: string) => AiStreamStartResult;
+
```

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

Copy

Review

Accept

  patching file:  [useAiStream.hook.promptSourceIdentity.test.ts](#)

[AICode/src/frontend/src/messageBus/ test /useAiStream.hook.promptSourceIdentity.test.ts](#)

Error

([what is it?](#))

  patching file:  [useAiStream.hook.promptSourceIdentity.test.ts](#)

[AICode/src/frontend/src/messageBus/ test /useAiStream.hook.promptSourceIdentity.test.ts](#)

Copy

Review

Accept

PatchFull file

```
-      waitForCommit: async () => {},
-
+      waitForCommit: async () => {},
+      forgetAttemptFailure: () => {},
+
-    },
-    whenIdle: async () => {},
-
+    },
+    whenIdle: async () => {},
+    forgetAttemptFailure: () => {},
+
-      sourceId: 'source-shared',
-      storageScopeRelPath: '',
-      aiQuery: { type: 'prompt', promptValue: 'A' },
-
+      storageScopeRelPath: '',
+      aiQuery: { type: 'prompt', promptValue: 'A' },
+
-      refusedStart = api.start({
-        kind: 'promptQuery',
-        turnId: 'turn-shared',
-        sourceId: 'source-shared',
-
+      refusedStart = api.start({
```

```

+         kind: 'promptQuery',
+         turnId: 'turn-shared',
+
-         if (!currentStart.accepted) throw new Error('Expected accepted start.');
```

```

+         if (!currentStart.accepted) throw new Error('Expected accepted start.');
```

```

+         const firstSourceId = currentStart.runRef.sourceId;
+
-         turnId: 'turn-shared',
-         sourceId: 'source-shared',
-
+         turnId: 'turn-shared',
+
-         throw new Error('Expected the successor start to be accepted.');
```

```

+         throw new Error('Expected the successor start to be accepted.');
```

```

+         expect(currentStart.runRef.sourceId).not.toBe(firstSourceId);
+
-         turnId: 'turn-retry-stop',
-         sourceId: 'source-retry-stop',
-
+         turnId: 'turn-retry-stop',
+

```

[AI Code/src/frontend/src/messageBus/ test /useAiStream.hook.promptSourceIdentity.test.ts](#)

Copy

Review

Accept



✅ **patching file:** [useAiStream.hook.liveSessionObserver.test.ts](#)

[AI Code/src/frontend/src/messageBus/ test /useAiStream.hook.liveSessionObserver.test.ts](#)

Copy

Review

Accept

PatchFull file

```

- let queueRunMerged: QueueRunMerged | undefined;
-
+ let queueRunMerged: QueueRunMerged | undefined;
+ let queueAIdleFailure: Error | undefined;
+ let queueBIdleFailure: Error | undefined;
+
-         },
-         whenIdle: async (): Promise<void> => {},
-
+         },
+         whenIdle: async (): Promise<void> => {
+             if (queueAIdleFailure) throw queueAIdleFailure;
+         },
+
-         waitForCommit: async (): Promise<void> => {},
-
+         waitForCommit: async (): Promise<void> => {},
+         forgetAttemptFailure: (): void => {
+             queueAIdleFailure = undefined;
+         },
+
-         },
-         whenIdle: async (): Promise<void> => {},

```

```

+      },
+      whenIdle: async (): Promise<void> => {
+        if (queueBIdleFailure) throw queueBIdleFailure;
+      },
+      forgetAttemptFailure: (): void => {
+        queueBIdleFailure = undefined;
+      },
+    },
+  },
+
+  queueRunMerged = undefined;
+
+  queueRunMerged = undefined;
+  queueAIdleFailure = undefined;
+  queueBIdleFailure = undefined;
+
+  expect(harnessApiRef.current?.terminalDataSeen).toBe(true);
+});
+
+  expect(harnessApiRef.current?.terminalDataSeen).toBe(true);
+});
+
+it('classifies a retained Queue A failure as transportError instead of completed',
+async () => {
+  jest.spyOn(console, 'error').mockImplementation(() => {});
+  await act(async () => {
+    root.render(React.createElement(Harness));
+  });
+  const api = harnessApiRef.current;
+  if (!api) throw new Error('Hook harness was not initialized.');
```

```

+  let startResult: AiStreamStartResult | undefined;
+  act(() => {
+    startResult = api.start({
+      kind: 'liveSessionObserver',
+      runRef: RUN_REF,
+      viewerLeaseId: 'lease',
+      replayMode: 'continueAssistant',
+      stopPolicy: 'source',
+    });
+  });
+  if (!startResult?.accepted) {
+    throw new Error('Expected the observer start to be accepted.');
```

```

+  }
+  const registration = registrationsByChannel.get(
+    'proxylocalapi:liveSessionStream',
+  );
+  if (!registration) throw new Error('Observer listener was not registered.');
```

```

+  queueAIdleFailure = new Error('Queue A dispatch failed');
```

```

+  act(() => registration.handlers.onComplete());
```

```

+  await waitFor(() => {
+    expect(harnessApiRef.current?.outcome).toBe('transportError');
```

```

+  });
+  expect(harnessApiRef.current?.isStreaming).toBe(false);
+  expect(queuedChunks).toEqual([
+    expect.objectContaining({
+      chunk: expect.objectContaining({
+        type: 'meta',
+        value: expect.objectContaining({ metaType: 'error' }),
+      }),
+      attemptIdentity: {
```

Copy
Review
Accept

Copy
Review
Accept

```
-   ]);
-   });
-
+   ]);
+   });
+
+   it('retains a runner failure for the exact attempt until terminal classification
consumes it', async () => {
+     const apiRef: { current: StreamSerializedQueueApi | null } = {
+       current: null,
+     };
+     const failure = new Error('Queue A runner failed');
+
+     function Harness(): JSX.Element {
+       const runMerged = React.useCallback(async (): Promise<void> => {
+         throw failure;
+       }, []);
+       const queueOptions = React.useMemo(
+         () => ({ runMerged }),
+         [runMerged],
+       );
+       const queue = useSerializedStreamQueue(queueOptions);
+       apiRef.current = queue;
```

```

+         return React.createElement(React.Fragment, null);
+     }
+
+     renderToStaticMarkup(React.createElement(Harness));
+     const api = apiRef.current;
+     if (!api) throw new Error('StreamSerializedQueueApi not initialized');
+
+     api.enqueueChunkForAttempt(token('failure'), ATTEMPT_A);
+     await expect(api.whenIdle(ATTEMPT_A)).rejects.toBe(failure);
+
+     api.forgetAttemptFailure(ATTEMPT_A);
+     await expect(api.whenIdle(ATTEMPT_A)).resolves.toBeUndefined();
+ });
+

```

[AI Code/src/frontend/src/messageBus/_test_/AiStreamMergedQueue.test.ts](#)

Copy

Review

Accept



patching file: [PromptStreamDispatchGateQueue.test.ts](#)

[AI Code/src/frontend/src/messageBus/_test_/PromptStreamDispatchGateQueue.test.ts](#)

Copy

Review

Accept

PatchFull file

```

-     });
- });
-
+     });
+
+     it('retains a downstream dispatch failure for the exact attempt until the stream
owner consumes it', async () => {
+     PromptStreamLockStore.reset();
+     const container = document.createElement('div');
+     document.body.appendChild(container);
+     const root = createRoot(container);
+     const failure = new Error('Queue B dispatch failed');
+     let gate: PromptStreamDispatchGateQueueApi | undefined;
+
+     function FailureHarness(): JSX.Element | null {
+         const dispatchMerged = useCallback(async (): Promise<void> => {
+             throw failure;
+         }, []);
+         const queueOptions = useMemo(
+             () => ({
+                 dispatchMerged,
+                 getAttemptIdentity: getAttemptA,
+             }),
+             [dispatchMerged],
+         );
+         const api = usePromptStreamDispatchGateQueue(queueOptions);
+         useEffect(() => {
+             gate = api;
+         }, [api]);
+         return null;
+     }
+
+     await act(async () => {
+         root.render(React.createElement(FailureHarness));
+     });
+     const queue = gate;

```

```

+         if (!queue) throw new Error('Gate queue API not initialized');
+
+         let enqueueFailure: unknown;
+         await act(async () => {
+             try {
+                 await queue.enqueueMergedForAttempt(
+                     tokenChunk('failure'),
+                     ATTEMPT_A,
+                 );
+             } catch (error) {
+                 enqueueFailure = error;
+             }
+         });
+         expect(enqueueFailure).toBe(failure);
+         await expect(queue.whenIdle(ATTEMPT_A)).rejects.toBe(failure);
+
+         queue.forgetAttemptFailure(ATTEMPT_A);
+         await expect(queue.whenIdle(ATTEMPT_A)).resolves.toBeUndefined();
+
+         await act(async () => {
+             root.unmount();
+         });
+         document.body.removeChild(container);
+     });
+
+     it('removes a failed deferred replay so exact whenIdle rejects instead of hanging',
+     async () => {
+         PromptStreamLockStore.reset();
+         const container = document.createElement('div');
+         document.body.appendChild(container);
+         const root = createRoot(container);
+         const failure = new Error('Deferred Queue B dispatch failed');
+         let gate: PromptStreamDispatchGateQueueApi | undefined;
+
+         function DeferredFailureHarness(): JSX.Element | null {
+             const dispatchMerged = useCallback(
+                 async (chunk: AiStreamedChunk): Promise<void> => {
+                     if (chunk.type === 'token') throw failure;
+                 },
+                 [],
+             );
+             const queueOptions = useMemo(
+                 () => ({
+                     dispatchMerged,
+                     getAttemptIdentity: getAttemptA,
+                 }),
+                 [dispatchMerged],
+             );
+             const api = usePromptStreamDispatchGateQueue(queueOptions);
+             useEffect(() => {
+                 gate = api;
+             }, [api]);
+             return null;
+         }
+
+         await act(async () => {
+             root.render(React.createElement(DeferredFailureHarness));
+         });
+         const queue = gate;
+         if (!queue) throw new Error('Gate queue API not initialized');
+
+         const lockedCall: AiStreamedToolCall = {
+             callId: 'deferred-failure-lock',

```



```

-
+      stopMock.mockClear();
+      mockedIsStreaming = false;
+      mockedStreamInstanceId = undefined;
+
-    ]);
-  });
-
+    ]);
+  });
+
+    it('publishes finalization for an accepted stream that completed without visible
chunks or meta', async () => {
+      mockedStreamInstanceId = 'stream-empty-complete';
+
+      await act(async () => {
+        root.render(React.createElement(Harness));
+      });
+
+      expect(controllerApiRef.current?.finalizedStreamInstanceId).toBe(
+        'stream-empty-complete',
+      );
+    });
+

```

[AICode/src/frontend/src/app/prompt/Prompt/ test /usePromptRunController.liveSessionObserver.test.tsx](#)

Copy

Review

Accept

  *reading project:*  e ["Mocks Queue A restants"](#),  e ["Mocks Queue B restants"](#),  e ["SourceId prompt résiduel"](#),  e ["SourceId retry résiduel"](#),  [useAiStream.hook.promptSourceIdentity.test.ts](#) [L145-L340],  e ["VscButton onClick type"](#),  e ["Async VscButton callbacks"](#)

  *patching file:*  [SidebarActivityQuery.tsx](#)

[AICode/src/frontend/src/app/sidebar/sidebar-activity/SidebarActivityQuery.tsx](#)

Copy

Review

Accept

PatchFull file

```

-
- import { Fragment, type JSX, useCallback, useMemo } from 'react';
-
+
+ import {
+   Fragment,
+   type JSX,
+   useCallback,
+   useMemo,
+   useRef,
+   useState,
+ } from 'react';
+
- }): JSX.Element {
-
+ }): JSX.Element {
+   const [stopPending, setStopPending] = useState(false);
+   const stopPendingRef = useRef(false);
+   const stopPayload = useMemo(
+     () => ({ runRef: item.runRef }),
+     [item.runRef],

```

```

+    );
+
+    /** Consume the local click immediately, then restore retryability only when the
+    exact RPC was refused. */
+    const requestStop = useCallback(async (): Promise<void> => {
+        if (stopPendingRef.current || item.stopRequested) return;
+
+        // Local pending state closes the same-render double-click window before the
+        Sidebar broadcast returns.
+        stopPendingRef.current = true;
+        setStopPending(true);
+        try {
+            const result = await bus.call(
+                'sidebar->ext:stopDetachedQuery',
+                stopPayload,
+            );
+            if (!result.accepted) {
+                // The backend did not consume this source Stop capability, so the
+                operator may retry.
+                stopPendingRef.current = false;
+                setStopPending(false);
+            }
+        } catch (error) {
+            // Transport failure is also retryable because no accepted backend abort
+            was confirmed.
+            stopPendingRef.current = false;
+            setStopPending(false);
+            console.error(
+                '[SidebarActivityQuery] detached source Stop request failed:',
+                error,
+            );
+        }
+    }, [item.stopRequested, stopPayload]);
+
+    /** Keep the visual button callback synchronous while the fully handled RPC
+    continues asynchronously. */
+

```

```

-     const handleStop = useCallback(()> void => {
-         void bus.call('sidebar->ext:stopDetachedQuery', {
-             runRef: item.runRef,
-         });
-     }, [item.runRef]);
-
+     const handleStop = useCallback(()> void => {
+         void requestStop();
+     }, [requestStop]);
+

```

```

-         variant="danger"
-         enabled={!item.stopRequested}
-         disableAfterClick
-
+         variant="danger"
+         enabled={!item.stopRequested && !stopPending}
+

```

[AI Code/src/frontend/src/app/sidebar/sidebar-activity/SidebarActivityQuery.tsx](#)

Copy

Review

Accept

  patching file:  [SidebarActivityQuery.test.tsx](#)

[AICode/src/frontend/src/app/sidebar/sidebar-activity/ test /SidebarActivityQuery.test.tsx](#)

Copy

Review

Accept

PatchFull file

```
-      // Hooks must not return Jest's mock object as a test completion value.
-      busCallMock.mockClear();
-
+      // Hooks must not return Jest's mock object as a test completion value.
+      busCallMock.mockReset();
+      busCallMock.mockResolvedValue({ accepted: true });
+
-
-      const stop = screen.getByTestId('query-activity-stop-source-a');
-      await act(async () => fireEvent.click(stop));
-
+      const stop = screen.getByTestId('query-activity-stop-source-a');
+      await act(async () => {
+        fireEvent.click(stop);
+        fireEvent.click(stop);
+      });
+      expect(busCallMock).toHaveBeenCalledTimes(1);
+
-
-      it('never renders detached Stop for Idle, observer-owned, or open source
projections', () => {
-
+
+      it('restores detached Stop after the backend refuses the exact source abort', async
() => {
+        busCallMock.mockResolvedValueOnce({ accepted: false });
+        const runRef = {
+          locator: LOCATOR,
+          turnId: 'turn-refused',
+          sourceId: 'source-refused',
+        };
+        render(
+          <SidebarActivityQuery
+            queryState={{
+              items: [
+                {
+                  status: 'running',
+                  ownership: 'source',
+                  panelState: 'closed',
+                  runRef,
+                  ...LOCATOR,
+                  title: 'Refused source',
+                  liveQueryElapsedMs: 1,
+                  stopRequested: false,
+                  updatedAt: 1,
+                  openedAt: 1,
+                },
+              ],
+            }}
+          />,
+        );
+        const stop = screen.getByTestId(
+          'query-activity-stop-source-refused',
+        );
+
+        await act(async () => {
+          fireEvent.click(stop);
+          await Promise.resolve();
+        });
+      });
+    });
+  });
+});
```

```

+         });
+
+         expect(stop).not.toBeDisabled();
+     });
+
+     it('restores detached Stop after an RPC failure without leaking an unhandled
rejection', async () => {
+         jest.spyOn(console, 'error').mockImplementation(() => {});
+         busCallMock.mockRejectedValueOnce(new Error('temporary RPC failure'));
+         const runRef = {
+             locator: LOCATOR,
+             turnId: 'turn-rpc-failure',
+             sourceId: 'source-rpc-failure',
+         };
+         render(
+             <SidebarActivityQuery
+                 queryState={{
+                     items: [
+                         {
+                             status: 'running',
+                             ownership: 'source',
+                             panelState: 'closed',
+                             runRef,
+                             ...LOCATOR,
+                             title: 'RPC failure source',
+                             liveQueryElapsedMs: 1,
+                             stopRequested: false,
+                             updatedAt: 1,
+                             openedAt: 1,
+                         },
+                     ],
+                 }}
+             />,
+         );
+         const stop = screen.getByTestId(
+             'query-activity-stop-source-rpc-failure',
+         );
+
+         await act(async () => {
+             fireEvent.click(stop);
+             await Promise.resolve();
+         });
+
+         expect(stop).not.toBeDisabled();
+     });
+
+     it('never renders detached Stop for Idle, observer-owned, or open source
projections', () => {
+

```

[AI Code/src/frontend/src/app/sidebar/sidebar-activity/ test /SidebarActivityQuery.test.tsx](#)

Copy

Review

Accept



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

[AI Code/src/extension/src/extension/panels/prompt/liveSession/PromptLiveSessionPanelCoordinator.service.ts](#)

Copy

Review

Accept

PatchFull file

```

-         await this.enqueue(args.runRef.locator, async () => {

```

```

-         await sidebarQueryRunningBridge.transitionSourceSettled(
-             args.runRef,
-             args.runId,
-         );
-         if (
-             args.finalization === 'handoffReady' ||
-             args.finalization === 'retiredMissingBaseline' ||
-             args.finalization === 'retiredMissingTerminal' ||
-             args.finalization === 'reservationCancelled'
-         ) {
-             await this.cleanupDetachedIfAllowed(args.runRef.locator);
-         }
+         await this.enqueue(args.runRef.locator, async () => {
+             try {
+                 await sidebarQueryRunningBridge.transitionSourceSettled(
+                     args.runRef,
+                     args.runId,
+                 );
+                 if (
+                     args.finalization === 'handoffReady' ||
+                     args.finalization === 'retiredMissingBaseline' ||
+                     args.finalization === 'retiredMissingTerminal' ||
+                     args.finalization === 'reservationCancelled'
+                 ) {
+                     await this.cleanupDetachedIfAllowed(args.runRef.locator);
+                 }
+             } finally {
+                 // Sound state is keyed by the exact runRef, so stale cleanup cannot
+                 // affect a successor source.
+             }
+         }

```

[AI Code/src/extension/src/extension/panels/prompt/liveSession/PromptLiveSessionPanelCoordinator.service.ts](#)

Copy

Review

Accept



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

[AI Code/src/extension/src/extension/panels/prompt/liveSession/ test /](#)

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

Copy

Review

Accept

PatchFull file

```

-
-         it('keeps stale source finalization from retiring sound or detached cleanup', async
-             () => {
-
-
+             it('retires exact stale source sound without running detached cleanup', async () =>
+             {
+
-
-             expect(transitionSourceSettledMock).toHaveBeenCalledWith(RUN, 13);
-             expect(cleanupSoundRunMock).not.toHaveBeenCalled();
-
+             expect(transitionSourceSettledMock).toHaveBeenCalledWith(RUN, 13);
+             expect(cleanupSoundRunMock).toHaveBeenCalled();
+             expect(cleanupOnPromptDisposeMock).not.toHaveBeenCalled();
+         });
+
+         it('retires exact source sound even when the Activity transition fails', async ()
=> {

```

```

+         transitionSourceSettledMock.mockRejectedValueOnce(
+             new Error('Activity transition failed'),
+         );
+
+         await expect(
+             createCoordinator().onSourceSettled({
+                 runRef: RUN,
+                 runId: 14,
+                 finalization: 'stale',
+             }),
+         ).rejects.toThrow('Activity transition failed');
+
+         expect(cleanupSoundRunMock).toHaveBeenCalledWith(RUN);
+

```

[AI Code/src/extension/src/extension/panels/prompt/liveSession/_test_/PromptLiveSessionPanelCoordinator.service.test.ts](#)

Copy

Review

Accept



patching file: [SidebarQueryRunningBridge.ts](#)

[AI Code/src/extension/src/messageBus/handlers/ia/SidebarQueryRunningBridge.ts](#)

Copy

Review

Accept

PatchFull file

```

- import type { LiveSessionRunRef } from
- '../../../../../sharedlib/types/liveSessionTypes/LiveSessionRun.type.js';
- import type { ChatSessionLocator } from
- '../../../../../sharedlib/types/chatTypes/ChatSessionLocator.type.js';
-
+ import type { LiveSessionRunRef } from
+ '../../../../../sharedlib/types/liveSessionTypes/LiveSessionRun.type.js';
+
-
- /** Compatibility facade for observer call sites until their step-2 handler
- migration completes. */
- public onStart(args: {
-     threadId: string;
-     storageScopeRelPath: ChatSessionLocator['storageScopeRelPath'];
-     turnId: string;
-     sourceId: string;
- }): SidebarQueryActivityAllocation {
-     return this.allocateStart({
-         runRef: {
-             locator: {
-                 sessionId: args.threadId,
-                 storageScopeRelPath: args.storageScopeRelPath,
-             },
-             turnId: args.turnId,
-             sourceId: args.sourceId,
-         },
-         ownership: 'observer',
-         sourceStartedAtMs: Date.now(),
-         panelState: 'open',
-     });
- }
-
+

```

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

Copy

Review

Accept

  reading project:  [EasyAgentsItemPanel.util.ts](#) [L1-L170]

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

[AICode/src/extension/src/extension/panels/core/_test_/openPanel.util.test.ts](#)

Copy

Review

Accept

PatchFull file

```
- import type { LiveSessionRunRef } from
'../../../../../sharedlib/types/liveSessionTypes/LiveSessionRun.type.js';
-
+ import type { LiveSessionRunRef } from
'../../../../../sharedlib/types/liveSessionTypes/LiveSessionRun.type.js';
+ import {
+   EASY_AGENT_ITEM_TARGET_TAB_QUERY_PARAM,
+   EasyAgentItemTabId,
+ } from ' ../../../../../sharedlib/types/easyAgentsTypes/EasyAgentItemTabId.type.js';
+ import { buildEasyAgentsBatchPanelKey } from '../EasyAgentsBatchPanel.util.js';
+ import { buildEasyAgentsItemPanelKey } from '../EasyAgentsItemPanel.util.js';
+

```

```
-   >();
-
+   >();
+ const postMessageMock = jest.fn<(message: unknown) => Promise<boolean>>(
+   async () => true,
+ );
+ const detachItemViewMock = jest.fn<
+   (args: { agentId: string }) => Promise<void>
+ >(async () => {});
+ const invalidateItemBootstrapMock = jest.fn<
+   (args: { agentId: string }) => Promise<void>
+ >(async () => {});
+ const detachHistoryViewMock = jest.fn<
+   (args: { batchParentLocator: ChatSessionLocator }) => Promise<void>
+ >(async () => {});
+

```

```
-   active: false,
-   webview: { postMessage: async () => true },
-
+   active: false,
+   webview: { postMessage: postMessageMock },
+

```

```
-   getEasyAgentsBatchRunManagerForProd: () => ({
-     detachItemView: async () => {},
-     invalidateItemBootstrap: async () => {},
-     detachHistoryView: async () => {},
-
+   getEasyAgentsBatchRunManagerForProd: () => ({
+     detachItemView: detachItemViewMock,
+     invalidateItemBootstrap: invalidateItemBootstrapMock,
+     detachHistoryView: detachHistoryViewMock,
+

```

```
-   revealPanelMock.mockReset();
-
```

```

+         revealPanelMock.mockReset();
+         postMessageMock.mockReset();
+         postMessageMock.mockResolvedValue(true);
+         detachItemViewMock.mockClear();
+         invalidateItemBootstrapMock.mockClear();
+         detachHistoryViewMock.mockClear();
+
-
-     it('propagates source-settlement coordinator failures to the exact source handler',
async () => {
-
+
+     it('reuses and retargets the exact EasyAgents item panel without creating a
duplicate', async () => {
+         const agentId = 'c'.repeat(32);
+         await openCentralPanel('easyAgentsItem', [
+             { name: 'agentId', value: agentId },
+         ]);
+
+         const key = buildEasyAgentsItemPanelKey(agentId);
+         expect(openedPanelsRegistry.get(key)).toBeDefined();
+
+         await openCentralPanel('easyAgentsItem', [
+             { name: 'agentId', value: agentId },
+             {
+                 name: EASY_AGENT_ITEM_TARGET_TAB_QUERY_PARAM,
+                 value: EasyAgentItemTabId.Form,
+             },
+         ]);
+         await Promise.resolve();
+
+         expect(loadPanelMock).toHaveBeenCalledTimes(1);
+         expect(revealPanelMock).toHaveBeenCalledTimes(1);
+         expect(postMessageMock).toHaveBeenCalledWith({
+             type: 'ext->easyAgentsItem:selectTab',
+             payload: { targetTab: EasyAgentItemTabId.Form },
+         });
+     });
+
+     it('detaches and invalidates the exact EasyAgents item runtime on panel disposal',
async () => {
+         const agentId = 'd'.repeat(32);
+         await openCentralPanel('easyAgentsItem', [
+             { name: 'agentId', value: agentId },
+         ]);
+         const panel = openedPanelsRegistry.get(
+             buildEasyAgentsItemPanelKey(agentId),
+         );
+         if (!panel) throw new Error('Expected EasyAgents item panel.');
```

```

+         panel.dispose();
+         await Promise.resolve();
+         await Promise.resolve();
+
+         expect(detachItemViewMock).toHaveBeenCalledWith({ agentId });
+         expect(invalidateItemBootstrapMock).toHaveBeenCalledWith({ agentId });
+     });
+
+     it('detaches the exact EasyAgents Batch history locator on panel disposal', async
() => {
+         const locator = {
+             sessionId: 'e'.repeat(32),
+             storageScopeRelPath: 'batch/history',

```

```

+     } as const;
+     await openCentralPanel('easyAgentsBatch', [
+       { name: 'sessionId', value: locator.sessionId },
+       {
+         name: 'storageScopeRelPath',
+         value: locator.storageScopeRelPath,
+       },
+     ]);
+     const panel = openedPanelsRegistry.get(
+       buildEasyAgentsBatchPanelKey(locator),
+     );
+     if (!panel) throw new Error('Expected EasyAgents Batch panel.');
```

```

+     panel.dispose();
+     await Promise.resolve();
+     await Promise.resolve();
+
+     expect(detachHistoryViewMock).toHaveBeenCalledWith({
+       batchParentLocator: locator,
+     });
+   });
+
+   it('propagates source-settlement coordinator failures to the exact source handler',
+     async () => {
```

[AI/Code/src/extension/src/extension/panels/core/_test_/openPanel.util.test.ts](#)

Copy

Review

Accept

🔧 ☒ reading project:  [usePromptStreamDispatchGateQueue.hook.ts](#) [L700-L840],  [usePromptStreamDispatchGateQueue.hook.ts](#) [L840-L930]

🔧 ☒ patching file:  [usePromptStreamDispatchGateQueue.hook.ts](#)
[AICode/src/frontend/src/messageBus/usePromptStreamDispatchGateQueue.hook.ts](#)

Grouped
([what is it?](#))

- 🔧 ☒ reading project: [useAiStreamQueue.hook.ts](#) [L35-L145], [useAiStreamQueue.hook.ts](#) [L180-L360], [useAiStreamQueue.hook.ts](#) [L520-L614], [usePromptStreamDispatchGateQueue.hook.ts](#) [L220-L370], [e "Reply callback implementation"](#)
- 🔧 ☒ scanning git: ["SourceId optionnel résid "](#), ["État actif mort résiduel"](#), ["Facade onStart résiduelle"](#)
- 🔧 ☒ reading project: [AiStreamStartArgs.type.ts](#) [L14-L70], [e "Reservation literals"](#), [e "Record literals"](#), [usePromptRunController.finalization.hook.ts](#) [L1-L240], [LiveSessionRegistry.service.ts](#) [L45-L230], [LiveSessionRegistry.service.ts](#) [L270-L355], [LiveSessionRegistry.service.ts](#) [L785-L930], [LiveSessionRegistry.service.ts](#) [L1030-L1085], [LiveSessionRegistry.service.ts](#) [L210-L250], [LiveSessionRegistry.service.ts](#) [L345-L390], [LiveSessionRegistry.service.ts](#) [L930-L980]
- 🔧 ☒ patching file: [LiveSessionRegistry.service.ts](#)

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

([what is it?](#))



patching file: [LiveSessionRecord.type.ts](#)

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

Grouped

([what is it?](#))



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

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

Grouped

([what is it?](#))



reading project: [LiveSessionRegistry.service.test.ts](#) [L105-L175]



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

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

Grouped

([what is it?](#))



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

[AICode/src/frontend/src/messageBus/useAiStreamQueue.hook.ts](#)

Grouped

([what is it?](#))



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

[AICode/src/frontend/src/messageBus/usePromptStreamDispatchGateQueue.hook.ts](#)

Grouped

([what is it?](#))



patching file: [PromptStreamDispatchGateQueue.test.ts](#)

[AICode/src/frontend/src/messageBus/_test_/PromptStreamDispatchGateQueue.test.ts](#)

Grouped

([what is it?](#))



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

[AICode/src/frontend/src/messageBus/useAiStreamQueue.hook.ts](#)

Grouped

([what is it?](#))



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

[AICode/src/frontend/src/messageBus/usePromptStreamDispatchGateQueue.hook.ts](#)

Grouped

([what is it?](#))



patching file: [liveSession.programming-doc-readme.md](#)

[AICode/src/extension/src/aiTools/orch/liveSession/liveSession.programming-doc-readme.md](#)

Copy

Review

Accept

PatchFull file

- A run is identified by the complete chat locator plus logical `turnId` and execution-specific `sourceId`.
 - EasyAgents deliberately sets `sourceId === turnId`; normal Prompt allocates a fresh source id per accepted invocation.
 - A run is identified by the complete chat locator plus logical `turnId` and execution-specific `sourceId`.
 - EasyAgents deliberately sets `sourceId === turnId`; normal Prompt allocates a fresh source id only after synchronous stream acceptance. Prompt callers cannot provide or reuse that identity.
-

Run epochs protect normal source replacement. A separate destructive generation protects Open against acquire/release claim ABA cycles. Durable Delete and EasyAgents Retry must acquire an opaque locator destruction claim, keep it through their destructive work, pass it to eviction, and release it in `finally`. Detached Prompt cleanup acquires the same locator-wide claim before its first asynchronous Abort/Await/ClearMemory step and keeps it through orphan cleanup, sound cleanup, trivial deletion, and Sidebar convergence. This mutual exclusion prevents a later Delete, Retry, Open, or source reservation from overtaking an already-started cleanup. The coordinator suppresses only the synchronous wake-up emitted by its own cleanup-claim release so a persistent cleanup failure cannot self-retry forever; external Delete/Retry releases still wake deferred cleanup. Eviction never releases the claim, and release retires both directions of the opaque claim lookup before emitting its wake-up.

Run epochs protect normal source replacement. A separate destructive generation protects Open against acquire/release claim ABA cycles. Durable Delete and EasyAgents Retry must acquire an opaque locator destruction claim, keep it through their destructive work, pass it to eviction, and release it in `finally`. Durable Delete additionally claims every exact normal-Prompt source abort and awaits its source-settlement latch before the first disk removal. The latch survives reservation promotion and destructive eviction until the source handler itself completes, so a late `meta.error` or synthetic terminal cannot recreate a successfully deleted session. Detached Prompt cleanup acquires the same locator-wide claim before its first asynchronous Abort/Await/ClearMemory step and keeps it through orphan cleanup, sound cleanup, trivial deletion, and Sidebar convergence. This mutual exclusion prevents a later Delete, Retry, Open, or source reservation from overtaking an already-started cleanup. The coordinator suppresses only the synchronous wake-up emitted by its own cleanup-claim release so a persistent cleanup failure cannot self-retry forever; external Delete/Retry releases still wake deferred cleanup. Eviction never releases the claim, and release retires both directions of the opaque claim lookup before emitting its wake-up.

Abort is claimed atomically for either an active observer lease or an exact source-owned reservation/record. Claiming synchronously latches `abortRequested`; the returned callback must be invoked without an intervening `await`. A refused or failed frontend abort restores the exact still-active Stop capability and releases the local abort handshake so the operator may retry. When rendering already became idle but the transport capability remains stoppable, the handshake stays armed during the RPC and closes on its accepted response because no later `isStreaming` transition exists. A normal-Prompt source Stop latch is also released when replay converges to a stable failure, because that surface must keep Retry usable after a pre-baseline source retirement. Reservation snapshots expose their real source/observer ownership so panel coordination and Activity rehydration never reinterpret an EasyAgents reservation as a detached normal Prompt source. The registry emits deeply immutable advisory wake-ups only after complete observer-inactive and destruction-claim-release mutations, and isolates listener failures. When the last observer lease disappeared before EasyAgents completed its business handoff, the handoff emits the observer-inactive wake-up again so deferred panel cleanup is retried after the final authority transition.

Abort is claimed atomically for either an active observer lease or an exact source-owned reservation/record. Claiming synchronously latches `abortRequested`; the returned callback must be invoked without an intervening `await`. A refused or failed frontend abort restores the exact still-active Stop capability and releases the local abort handshake so the operator may retry. The detached Activity Stop follows the same rule

with local pending state instead of an irreversible click disable. When rendering already became idle but the transport capability remains stoppable, the handshake stays armed during the RPC and closes on its accepted response because no later `isStreaming` transition exists. A normal-Prompt source Stop latch is also released when replay converges to a stable failure, because that surface must keep Retry usable after a pre-baseline source retirement. Reservation snapshots expose their real source/observer ownership so panel coordination and Activity rehydration never reinterpret an EasyAgents reservation as a detached normal Prompt source. The registry emits deeply immutable advisory wake-ups only after complete observer-inactive and destruction-claim-release mutations, and isolates listener failures. When the last observer lease disappeared before EasyAgents completed its business handoff, the handoff emits the observer-inactive wake-up again so deferred panel cleanup is retried after the final authority transition. Exact sound state is retired for every source-handler finalization, including `stale`, because the full runRef prevents cleanup from touching a successor source.

[AICode/src/extension/src/aiTools/orch/liveSession/liveSession.programming-doc-readme.md](#)

Copy

Review

Accept



patching file: [prompt-stream-dispatch-lock.programming-doc-readme.md](#)

[AICode/src/frontend/src/messageBus/prompt-stream-dispatch-lock.programming-doc-readme.md](#)

Copy

Review

Accept

PatchFull file

The reset/finalization boundary is guarded by `sourceId + streamInstanceId`. Each mounted transport listener freezes that identity in its callback object, so an old bus callback cannot relabel its frames with the current stream ref. A stale callback, continuation after `await`, or Queue A `runMerged` item from an older accepted stream cannot enter gating, acquire a lock, reset Queue B, or mutate the successor stream. `completed` and `transportError` are published only after Queue A idle, Queue B idle, and the final React commit barrier. If terminal data was already consumed before transport failure, no second error meta or suffix is injected.

The reset/finalization boundary is guarded by `sourceId + streamInstanceId`. Each mounted transport listener freezes that identity in its callback object, so an old bus callback cannot relabel its frames with the current stream ref. A stale callback, continuation after `await`, or Queue A `runMerged` item from an older accepted stream cannot enter gating, acquire a lock, reset Queue B, or mutate the successor stream. Queue A and Queue B retain their first downstream failure per exact attempt while recovering their internal scheduling tails; `whenIdle(attemptIdentity)` rejects with that retained failure instead of silently reporting success. The stream owner consumes and forgets the failure only after publishing the attempt outcome. `completed` and `transportError` are published only after Queue A idle, Queue B idle, and the final React commit barrier. Any queue or commit failure forces `transportError`; if no terminal data was consumed, one canonical error terminal is retried through the recovered queues. If terminal data was already consumed before transport failure, no second error meta or suffix is injected.

[AICode/src/frontend/src/messageBus/prompt-stream-dispatch-lock.programming-doc-readme.md](#)

Copy

Review

Accept



patching file: [fallback-persist-and-title.programming-doc-readme.md](#)

[AICode/src/extension/src/aiTools/orch/ask/stream/core/fallback-persist-and-title.programming-doc-readme.md](#)

Copy

Review

Accept

PatchFull file

The frontend arms the clear barrier only after synchronous stream acceptance and correlates it with `{ turnId, streamInstanceId }`. If the accepted attempt finalizes without receiving its matching `fallback\_persisted` ack, Prompt removes the transactional mask and restores focus but preserves prompt text, attachments, `nextTurnId`, edit state, and maximized state. A refused start creates no scaffold and arms no barrier.

The frontend arms the clear barrier only after synchronous stream acceptance and correlates it with `{ turnId, streamInstanceId }`. If the accepted attempt finalizes without receiving its matching `fallback\_persisted` ack, Prompt removes the transactional mask and restores focus but preserves prompt text, attachments, `nextTurnId`, edit state, and maximized state. Finalization is published for every accepted stream instance, including a transport that completed without visible token, thinking, or meta data, so an empty failure path cannot leave the composer masked indefinitely. A refused start creates no scaffold and arms no barrier.

[AICode/src/extension/src/aiTools/orch/ask/stream/core/fallback-persist-and-title.programming-doc-readme.md](#)

Copy

Review

Accept

  patching file:  [isomorphism-invariant-rule-between-streaming-and-rehydration-of-chat-sessions.programming-doc-readme.md](#)

[AICode/src/frontend/src/app/prompt/hydration/isomorphism-invariant-rule-between-streaming-and-rehydration-of-chat-sessions.programming-doc-readme.md](#)

Copy

Review

Accept

PatchFull file

- Every mounted listener plus every Queue A/Queue B entry carries the exact `sourceId + streamInstanceId`; stale callbacks and post-`await` continuations cannot mutate a successor retry.
- Every mounted listener plus every Queue A/Queue B entry carries the exact `sourceId + streamInstanceId`; stale callbacks and post-`await` continuations cannot mutate a successor retry.
- Queue A/Queue B retain exact attempt failures until finalization classifies them as `transportError`; a consumed source terminal remains unique, while a terminal is synthesized through the recovered queues only when none was previously applied.
- Every accepted stream instance publishes frontend finalization even when it completed without visible chunks, ensuring fallback masks and direct-handoff barriers cannot wait forever on an empty transport.

[AICode/src/frontend/src/app/prompt/hydration/isomorphism-invariant-rule-between-streaming-and-rehydration-of-chat-sessions.programming-doc-readme.md](#)

Copy

Review

Accept

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

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

Grouped

([what is it?](#))

  reading project:  e "[Queue A API literals](#)",  e "[Queue B API literals](#)",  e "[Queue idle function types](#)"

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

  reading project:  [usePromptStreamDispatchGateQueue.hook.ts](#) [L35-L130],  [usePromptStreamDispatchGateQueue.hook.ts](#) [L1000-L1195],  [useAiStreamQueue.hook.ts](#) [L390-L520],  [useAiStreamQueue.hook.ts](#) [L555-L620],  [LiveSessionRegistry.service.ts](#) [L55-L245],  [LiveSessionRegistry.service.ts](#) [L285-L380],  [LiveSessionRegistry.service.ts](#) [L790-L830],  [LiveSessionRegistry.service.ts](#) [L1040-L1095],  e ["Mocks LiveSessionService"](#),  e ["LiveSessionService partials"](#),  e ["Service method mocks"](#),  [useAiStream.hook.promptSourceIdentity.test.ts](#) [L30-L85]

  patching file:  [SidebarActivityQuery.tsx](#)
[AICode/src/frontend/src/app/sidebar/sidebar-activity/SidebarActivityQuery.tsx](#)
Grouped
([what is it?](#))

  reading project:  h ["Async onClick production"](#),  [FilenameBrowseButton.tsx](#) [L1-L80]
  patching file:  [SidebarActivityQuery.tsx](#)
[AICode/src/frontend/src/app/sidebar/sidebar-activity/SidebarActivityQuery.tsx](#)
Grouped
([what is it?](#))

  reading project:  [useAiStream.hook.liveSessionObserver.test.ts](#) [L35-L105],  [useAiStream.hook.liveSessionObserver.test.ts](#) [L315-L390],  [delete.endpoint.liveSession.test.ts](#) [L1-L250],  [usePromptStreamDispatchGateQueue.hook.ts](#) [L180-L225]
  patching file:  [AiStreamStartArgs.type.ts](#)
[AICode/src/frontend/src/messageBus/AiStreamStartArgs.type.ts](#)
Grouped
([what is it?](#))

  patching file:  [usePromptRunController.hook.ts](#)
[AICode/src/frontend/src/app/prompt/Prompt/usePromptRunController.hook.ts](#)
Grouped
([what is it?](#))

  patching file:  [PromptLiveSessionPanelCoordinator.service.test.ts](#)
[AICode/src/extension/src/extension/panels/prompt/liveSession/_test_/PromptLiveSessionPanelCoordinator.service.test.ts](#)
Grouped
([what is it?](#))

  patching file:  [LiveSessionRegistry.service.ts](#)
[AICode/src/extension/src/aiTools/orch/liveSession/registry/LiveSessionRegistry.service.ts](#)
Grouped
([what is it?](#))

  reading project:  e ["runWithTimeout imports"](#)
  patching file:  [useAiStream.hook.ts](#)

[AICode/src/frontend/src/messageBus/useAiStream.hook.ts](#)

Grouped
([what is it?](#))

-   reading project:  [PromptLiveSessionPanelCoordinator.service.ts](#) [L330-L405]
-   patching file:  [useAiStreamQueue.hook.ts](#)

[AICode/src/frontend/src/messageBus/useAiStreamQueue.hook.ts](#)

Grouped
([what is it?](#))

-   patching file:  [usePromptStreamDispatchGateQueue.hook.ts](#)
- [AICode/src/frontend/src/messageBus/usePromptStreamDispatchGateQueue.hook.ts](#)

Grouped
([what is it?](#))

-   scanning recent files
 -   patching file:  [useAiStreamQueue.hook.ts](#)
- [AICode/src/frontend/src/messageBus/useAiStreamQueue.hook.ts](#)

Grouped
([what is it?](#))

-   patching file:  [usePromptStreamDispatchGateQueue.hook.ts](#)
- [AICode/src/frontend/src/messageBus/usePromptStreamDispatchGateQueue.hook.ts](#)

Grouped
([what is it?](#))

-   patching file:  [useAiStreamQueue.hook.ts](#)
- [AICode/src/frontend/src/messageBus/useAiStreamQueue.hook.ts](#)
- Grouped
([what is it?](#))

-   reading project:  [SidebarActivityQuery.tsx](#) [L35-L100]
 -   patching file:  [SidebarActivityQuery.tsx](#)
- [AICode/src/frontend/src/app/sidebar/sidebar-activity/SidebarActivityQuery.tsx](#)
- Grouped
([what is it?](#))

-   patching file:  [SidebarActivityQuery.test.tsx](#)
- [AICode/src/frontend/src/app/sidebar/sidebar-activity/ test /SidebarActivityQuery.test.tsx](#)
- Grouped
([what is it?](#))

Corrections complètes appliquées 

Joel, les huit ensembles de problèmes identifiés pendant la vérification ont été corrigés dans **29 fichiers**, sans modifier de fichier généré. La tranche a maintenant quelques ceintures de sécurité supplémentaires — le code asynchrone pourra encore courir, mais beaucoup moins avec des ciseaux ✂.

Domaine	Correction réalisée	Couverture associée	Complétude
Suppression destructive	La source normale est arrêtée puis son règlement exact est attendu avant la première suppression disque	Réservation, promotion, annulation,	

Domaine	Correction réalisée	Couverture associée	Complétude
Queue A / Queue B	Première erreur conservée par tentative exacte et exposée par <code>whenIdle()</code>	finalisation tardive, échec du règlement Erreur Queue A, erreur Queue B, échec d'un chunk différé	✓
Résultat de stream	Une erreur interne de queue produit désormais <code>transportError</code> , jamais un faux <code>completed</code>	Terminal existant conservé, terminal synthétique seulement s'il manque	✓
Finalisation vide	Toute tentative acceptée publie son <code>finalizedStreamInstanceId</code> , même sans token, thinking ou meta	Transport accepté puis terminé sans contenu visible	✓
Identité source	Un caller Prompt ne peut plus injecter ou réutiliser <code>sourceId</code>	Source fraîche pour chaque démarrage accepté	✓
Stop détaché	Double clic bloqué synchroniquement ; refus ou erreur RPC rend le bouton réutilisable	Acceptation, refus, rejet transport, remplacement de source	✓
Son	L'état sonore exact est supprimé même après une finalisation <code>stale</code> ou une transition Activity en erreur	Source stale et erreur de transition	✓
Contrats publics	Reply retourne maintenant le résultat strict d'acceptation	Contrat identique à Send, Retry, Continue et Edit	✓
Dette intermédiaire	Façade Activity temporaire et type de stream inutilisé supprimés	Absence de call sites résiduels	✓
Non-régression panneaux	Couverture historique EasyAgents restaurée	Réutilisation, retargeting, detach item, invalidation et Batch history	✓
Documentation	Invariants de suppression, queues, fallback et isomorphisme actualisés	Quatre programming docs synchronisées	✓

Suppression sans résurrection 🗑️

Le cœur de la correction se trouve dans :

- [le registre liveSession](#)
- [les contrats internes du registre](#)
- [la façade liveSession](#)
- [l'endpoint de suppression](#)

Le nouveau comportement est :

1. Le claim destructif bloque toute nouvelle réservation.
2. Toutes les sources Prompt normales exactes sont identifiées.
3. Leur abort est réclamé et invoqué sans `await` intermédiaire.
4. Un latch exact suit la source à travers la réservation puis la promotion.
5. La suppression attend que le handler source ait réellement abandonné tout droit de persistance et de forwarding.
6. Une limite temporelle empêche la suppression de rester bloquée indéfiniment.
7. Le disque n'est supprimé qu'après cette barrière.
8. L'éviction reste postérieure au succès disque.
9. Une erreur ou un timeout ne touche pas aux données persistées.

Le latch survit également à une éviction destructive jusqu'à la vraie fin du handler : une finalisation tardive `stale` libère correctement les waiters sans prétendre que la source avait déjà fini lors de l'éviction.

Erreurs Queue A / Queue B correctement propagées 🔄

Les corrections sont réparties entre :

- [Queue A](#)
- [Queue B](#)
- [le hook de transport Prompt](#)

Chaque queue conserve maintenant sa **première erreur par** `{ sourceId, streamInstanceId }` :

- la chaîne interne de scheduling est récupérée afin d'éviter une Promise rejetée sans observateur ;
- `whenIdle(attemptIdentity)` rejette avec l'erreur exacte au lieu de signaler un faux succès ;
- une erreur provenant d'une tentative stale n'est pas conservée ;
- le propriétaire du stream efface l'erreur uniquement après l'avoir convertie en outcome ;
- un échec de replay différé dans Queue B retire le chunk défaillant avant de notifier l'attente, évitant un `whenIdle()` bloqué indéfiniment.

Dans le hook de transport :

- une erreur de queue ou de commit transforme la tentative en `transportError` ;
- si un terminal data a déjà été appliqué, il demeure l'unique terminal ;
- sinon, un terminal `SYSTEM_ERROR` canonique est réinjecté à travers les queues récupérées ;
- la tentative ne peut plus être publiée comme `completed` après une erreur interne.

Finalisation frontend totale 🌟

La finalisation dans [le hook de finalisation Prompt](#) publie maintenant chaque `streamInstanceId` terminé exactement une fois, y compris lorsque le transport ne contient :

- aucun token ;
- aucun thinking token ;
- aucun meta ;
- aucun contenu assistant visible.

Cela permet notamment à la barrière fallback de retirer son masque transactionnel et de restaurer le composer sans effacer le texte ou les attachments lorsque l'ack durable n'est jamais arrivé.

Contrats stricts renforcés 🔗

Dans [les contrats de démarrage du stream](#) :

- `sourceId` a été retiré de la branche `promptQuery` ;
- il est exclusivement généré après l'acceptation synchrone ;
- le type intermédiaire inutilisé `AiStreamActiveState` a été supprimé.

Dans [le contrôleur de run Prompt](#), `handleReplyButton` expose désormais `AiStreamStartResult`, comme les six autres commandes transactionnelles.

La façade temporaire `onStart()` a également été retirée de [le runtime Activity](#).

Stop détaché récupérable 🛑

Dans [l'activité Query de la Sidebar](#) :

- une ref synchrone ferme la fenêtre de double clic avant le premier `await` ;
- l'état React désactive immédiatement le bouton ;
- une réponse `{ accepted: false }` réactive le bouton ;
- une erreur RPC est absorbée, journalisée et rend également le bouton réutilisable ;
- une acceptation conserve le bouton désactivé jusqu'à la projection `stopRequested` ou la disparition de la source ;
- le sous-arbre reste keyé par `sourceId`, donc une source successive reçoit un état local neuf.

Nettoyage sonore exact 🔊

[le coordinateur liveSession des panneaux Prompt](#) retire maintenant l'état sonore dans un `finally` :

- après une finalisation normale ;
- après une retraite sans baseline ou terminal ;
- après une annulation de réservation ;
- après une finalisation `stale` ;
- même si la transition Activity échoue.

Le nettoyage reste sans danger pour une source successive, car il utilise le `runRef` complet incluant `sourceId`.

Couverture de non-régression restaurée 🛠️

Les tests ont été renforcés dans les zones suivantes :

- [tests du registre liveSession](#) : règlement après annulation, finalisation et finalisation post-éviction.

- [tests de suppression destructive](#) : ordre close → abort → settlement → delete → eviction, refus de suppression si le règlement échoue.
- [tests Queue A](#) : conservation et consommation d'une erreur exacte.
- [tests Queue B](#) : erreur downstream et échec d'un replay différé sans deadlock.
- [tests du transport liveSession frontend](#) : classification `transportError` et émission du terminal canonique.
- [tests d'identité source Prompt](#) : source fraîche après chaque nouvelle acceptation.
- [tests du Stop Activity](#) : double clic, refus et erreur RPC.
- [tests du coordinateur panneau](#) : cleanup sonore stale et cleanup malgré une erreur Activity.
- [tests historiques du panneau central](#) : réutilisation et retargeting EasyAgents Item, detach/invalidation et Batch history.
- [tests du contrôleur Prompt](#) : publication d'une finalisation de stream vide.

Documentation synchronisée

Les invariants corrigés sont maintenant décrits dans :

- [documentation liveSession](#)
- [documentation Queue B](#)
- [documentation fallback persist](#)
- [documentation d'isomorphisme](#)

Nom du commit

text

Copy

fix(live-session): harden deletion, stream failures, and detached controls



Les sources se ferment sans ressusciter 🗡️ Les queues savent enfin protester 🎵 Le liveSession peut respirer !

</ASSISTANT>

SHOW ALLPage 1 / 1