Permalink URI:
vscode://joelabenhaim.aicode-ja/chat?sessionId=d4dccaa5865a4b68810a6b54cbd7abb5-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*: ᛘ *"État Git courant"*, ᛘ *"Fichiers changés"*, ᛘ *"Volume des changements"*
🔧 ✅ *reading project*: 🗀 g *"Docs liveSession proches"*, 🗀 g *"Docs invariants proches"*, 🗀 g *"Docs flux Prompt proches"*, 🗀 g *"Docs EasyAgents proches"*, 🗎 *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*, 🗀 g *"Doc fallback persist"*
🔧 ✅ *scanning git*: ᛘ *"Historique récent"*, ᛘ *"Commit courant"*
🔧 ✅ *reading project*: 🗎 *fallback-persist-and-title.programming-doc-readme.md*
🔧 ✅ *scanning git*: ᛘ *"Inventaire du commit"*, ᛘ *"Statistiques du commit"*, ᛘ *"Résumé du commit"*
🔧 ✅ *reading project*: 🔎 h *"Identité liveSession"*, 🔎 h *"Registre liveSession"*, 🔎 h *"Source Prompt normale"*, 🔎 h *"Transport et origine"*, 🔎 h *"Frontend liveSession"*, 🔎 h *"Activité panneau Stop"*, 🔎 e *"Ancien canAbortSource"*, 🔎 e *"Ancien query windows"*, 🔎 e *"Ancien streamKey"*, 🔎 e *"Ancien type Activity"*, 👉 *"Contrats et registre"*, 👉 *"Source et transport"*, 👉 *"Handlers et activité"*, 👉 *"Frontend transactionnel"*, 🔎 e *"Définition reserveSource"*, 🔎 e *"Définition registerRun"*, 🔎 e *"Finalisation source"*, 🔎 e *"Classification Open"*, 🗎 *Prompt.tsx* [L620-L735], 🗎 *SidebarStore.rehydrate.manager.ts* [L110-L250], 🗎 *PromptLiveSessionPanelCoordinator.service.ts* [L1-L140], 🗎 *extTypedMessageBus.util.ts* [L120-L220],
🗎 *usePromptLiveSession.normalPrompt.test.tsx*,
🗎 *usePromptLiveSession.normalPromptIsomorphism.test.tsx*,
🗎 *usePromptLiveSession.hydration.test.tsx*,
🗎 *usePromptLiveSession.handoff.test.tsx*,
🗎 *usePromptRunCommands.startAcceptance.test.tsx*,
🗎 *Prompt.liveSessionBootstrap.test.tsx*,
🗎 *PromptLiveSessionPanelCoordinator.service.test.ts*,
🗎 *PromptLiveSessionPanelQuery.util.test.ts*,
🗎 *useAiStream.hook.promptSourceIdentity.test.ts*,
🗎 *Prompt.fallbackPersistAck.clearInput.test.tsx*,
🗎 *Prompt.fallbackPersistAck.clearAttachments.test.tsx*, 👉 *"Réservation finalisation"*,
🗎 *LiveSessionOpen.service.ts*,
🗎 *LiveSessionOpenResult.type.ts*,
🗎 *LiveSessionRun.type.ts*,
🗎 *SidebarStore.ts*, 🗎 *PromptLiveSessionBootstrap.adapter.ts*, 🗎 *PromptLiveSessionPanelQuery.util.ts*
🔧 ✅ *scanning git*: ❌ ᛘ *"Contrôle espaces diff"*, ᛘ *"Anciens symboles restants"*
🔧 ✅ *reading project*:
🗎 *PromptQueryDoneSound.manager.ts*,
🗎 *PromptThreadCleanup.manager.ts*,
🗎 *aiStopSuffix.util.ts*,
🗎 *OrchMessagesMemoryAssistantPatch.ts*,
🗎 *stream.endpoint.ts*,
🗎 *abortSource.endpoint.ts*, 🗎 *PromptLiveSessionDetachedStop.service.ts*, 🗎 *delete.endpoint.ts*, 🔎 e *"Tests synthèse source"*, 🔎 h *"Cas erreur source"*, 🔎 h *"Cas persistance source"*, 🔎 h *"Cas timers replay"*, 🔎 h *"Sources successives panel"*, 👉 *"Tests erreur source"*, 🗎 *useAiStream.hook.ts* [L620-L820], 🗎 *useAiStream.hook.ts* [L360-L620], 🗎 *usePromptLiveSession.hook.ts* [L1-L300], 🗎 *applyHydratedPromptSessionState.util.ts* [L1-

L240],
AiStreamStartArgs.type.ts,
preparePromptContinuationStreaming.util.ts,
preparePromptStreamingTurn.util.ts, usePromptRunCommands.hook.ts [L1-L180], usePromptRunCommands.hook.ts [L280-L380], usePromptRunCommands.hook.ts [L460-L690], 🔎 e "Garde runMerged", 🔎 e "Usages identité active", useAiStreamQueue.hook.ts, usePromptStreamDispatchGateQueue.hook.ts, 🔎 e "finalized stream id", 🔎 e "runMerged guard doc",
usePromptRunController.finalization.hook.ts,
usePromptCancelController.hook.ts, PromptLiveSessionBanner.tsx, SidebarQueryRunningBridge.ts [L1-L130], SidebarStore.handlers.manager.ts, openPanel.util.ts, 🔎 e "Ack fallback stream", 🔎 e "Meta fallback persisted", 👉 "Traitement ack fallback", useAiStream.hook.liveSessionObserver.test.ts, Prompt.tsx [L1-L210], 🔎 e "Auto-run live désactivé", usePromptSessionLocator.hook.ts [L1-L120], usePromptSessionLocator.hook.ts [L250-L310], liveSessionStreamHandlers.ts, SidebarStore.queryWindow.mutations.manager.ts [L1-L220], 🔎 h "Observer devient Idle", 🔎 e "Appels onObserverStop", 🔎 e "Tests activité Idle", 🔎 e "Usages handleReplyButton", usePromptThinkingLifecycle.hook.ts, 🔎 e "Usages pendingHrRef", 🔎 e "Reset séparateur", 🔎 e "Usage isHydrating", 🔎 e "Affichage hydratation", 👉 "Rendu hydratation", 🔎 e "Préparation Open registre", 👉 "Permis Open et lease", LiveSessionRegistry.service.ts [L420-L560], SidebarQueryRunningBridge.race.test.ts, SidebarStore.queryWindow.listeners.manager.ts [L180-L280], 🔎 e "État running input", Prompt.tsx [L280-L390], 🔎 e "Calcul running input", usePromptRunController.hook.ts [L420-L470], usePromptRunController.uiState.hook.ts, 🔎 e "Ownership EasyAgents", 🔎 e "Enregistrement thread Prompt", 🔎 e "Usages registerSessionThread", 🔎 e "Test observer cleanup", EasyAgentsBatchLiveSessionAdapter.service.ts [L1-L180], 🔎 e "Release viewer EasyAgents", 👉 "Politique release viewer", 🔎 h "Retry claim destructif", 🔎 h "Delete claim destructif", 🔎 h "Rejeu et handoff", 🔎 h "Tests stack protocole V1", aiStopSuffix.util.test.ts, 🔎 e "Bootstrap replayMode usages", 🔎 r "openCentralPanel", 🔎 e "RPC register thread", uiAiPromptHandlers.ts [L90-L145], 🔎 e "Appel cleanup thread", 👉 "Enregistrement cleanup", 🔎 e "Ownership cleanup Prompt", Prompt.tsx [L470-L545], 🔎 e "Usage cleanupOwned", 🔎 e "Cleanup son exact"

Vérification effectuée

Joel, votre spécification est **remarquablement précise** — elle pourrait faire passer un interrogatoire à un ordonnanceur asynchrone sans même hausser la voix 😄.

J'ai vérifié l'état courant au commit propre `a87dd0e4cbd1d468a687fbe5ee6b8a7772c6011f`, intitulé `feat(live-session): keep prompt runs alive across panel close and reopen`.

J'ai également lu les documents de maintenance pertinents :

- architecture générique liveSession
- architecture EasyAgents
- invariant d'isomorphisme streaming/hydratation
- invariants thinking/hourglass
- invariants Queue B
- barrière fallback persist

Le commit touche **169 fichiers** pour **10 153 ajouts** et **11 547 suppressions**. La spécification numérote **160 fichiers attendus** : ils sont présents, mais le commit ajoute également **9 fichiers hors récapitulatif**.

---

1) Toute la spécification est-elle implémentée ?
Réponse : ❌ Non

La conformité dépend de ce que l'on mesure :

Mesure	Estimation	Conclusion
Chemins attendus présents ou modifiés	**100 %**	✅ Les 160 entrées numérotées ont un correspondant dans le commit
Contrats backend fondamentaux	**≈ 88 %**	✅ Ensemble globalement solide
Coordination panneau / Activité / EasyAgents	**≈ 60 %**	❌ Plusieurs erreurs de cycle de vie
Frontend transactionnel et rejeu	**≈ 65 %**	⚠ Architecture présente, gardes essentielles incomplètes

Mesure	Estimation	Conclusion
Tests comportementaux demandés	**≈ 25–30 %**	❌ Une grande partie des tests critiques sont devenus des tests déclaratifs
Respect strict du périmètre	**≈ 70 %**	❌ Un fichier personnel et sept fichiers générés ont été modifiés
Conformité globale estimée	**≈ 68 %**	❌ La tranche ne peut pas être déclarée finalisée

Les principales raisons du verdict négatif sont :

1. la réouverture peut choisir une ancienne source retenue au lieu de la source active suivante ;
2. les sorties déjà extraites de Queue A/Queue B ne sont pas corrélées à `sourceId + streamInstanceId` ;
3. le cycle de vie des records `ownership='observer'` reste incomplet ;
4. le Retry Open n'est pas réellement single-flight et son nettoyage d'unmount n'est pas fiable ;
5. le compositeur normalPrompt devient interactif avant la baseline et l'armement du rejeu ;
6. plusieurs états live n'offrent pas le Stop du compositeur alors que la source continue ;
7. une query live avec locator dupliqué atteint le low-level opener ;
8. les tests les plus importants de la spécification n'exercent plus le code réel.

Conclusion de la question 1
La structure de la tranche est largement posée, mais la spécification complète n'est pas implémentée au niveau comportemental et probatoire.

---

2) L'implémentation actuelle fait-elle ce qu'elle doit faire selon la spécification, fonctionnalité technique par fonctionnalité ?
Les 160 entrées peuvent être regroupées en contrats techniques cohérents.

Fonctionnalité technique demandée	État	Complétude estimée	Cons
Identité exacte `{ locator, turnId, sourceId }`	✅	**100 %**	Le contrat, la validation et la run ke dans LiveSessionRun.type.ts.
Descripteurs stricts EasyAgents/normalPrompt	✅	**95 %**	L'union shared, `sourceId===turn` numéro de ligne normalPrompt sont dans LiveSessionPanel.type.ts.
Résultats Open stricts et enveloppe `session: unknown`	✅	**95 %**	Les branches et propriétés explicite correctement contrôlées dans LiveS
Réservations synchrones avant `await`	✅	**95 %**	Réservation, pointeur actif, époque dans LiveSessionRegistry.service.ts
Promotion réservation → record sans nouveau bump	✅	**95 %**	`registerRun(...)` conserve l'ép
Retraites, tombstones et résultats de finalisation	✅	**90 %**	Les cinq résultats existent ; missing terminal.
Claims destructifs et génération indépendante	✅	**90 %**	Acquisition, libération, handle exact implémentés.
Buffer raw-live complet, ordonné et non borné	✅	**100 %**	Le registre conserve un tableau com
Leases exactes et replay backlog + tail	✅	**90 %**	Acquisition, attach, réautorisation p sont présentes.
`replaceTurnAssistants` et `continueAssistant`	✅	**90 %**	Le service Open filtre les assistants messages via LiveSessionOpen.serv
Lifecycle lazy du `Readable`	✅	**90 %**	Le priming avant registration, le pre sont factorisés dans LiveSessionSou
Persistance `meta.error` avant append	⚠	**85 %**	L'ordre persistance → append → fo dans NormalPromptLiveSessionSou cette persistance peut remplacer le t un `INTERRUPTED` synthétique.
EOF sans terminal → SYSTEM_ERROR	✅	**100 %**	Le wrapper est persisté puis append
Throw sans terminal → INTERRUPTED	✅	**100 %**	Le terminal est persisté et forwardé
Terminal puis erreur transport → terminal unique	✅	**90 %**	Le backend n'ajoute pas de second t conserve `transportError`.

Fonctionnalité technique demandée	État	Complétude estimée	Cons
Protocole V1 du premier `meta.error`	✅	**95 %**	Encodage/décodage stricts présents
Enveloppe Slimbk non-200 dans le body	✅	**95 %**	Détection dédiée dans parseSlimbk logique métier dans les deux handle
Contexte webview d'origine non sérialisé	✅	**100 %**	Le contexte exact est attaché dans e dans extMessageBus.util.ts.
Guard webview exacte avant réservation	✅	**100 %**	Le handler normal compare la webv émettrice dans iaStreamHandlers.ts.
FIFO extension et finalisation post-forwarding	✅	**90 %**	L'ordre endpoint → FIFO → Activi est présent.
Coordinateur de panneau injecté et sans import du low-level opener	✅	**90 %**	L'ouverture est injectée par port et asynchrone.
Sélection de la source active lors d'une réouverture	❌	**45 %**	PromptLiveSessionPanelCoordinato compris un ancien `handoffReady`, nouvelle.
Rejet extension des locators dupliqués avant low-level	❌	**50 %**	PromptLiveSessionPanelQuery.util. pas les doublons `sessionId` / `sto`
Activité Idle/open et Running/source/open|closed	⚠	**80 %**	Les sources normales font bien oper observer n'effectue pas cette transiti
Stop détaché uniquement source/closed	✅	**95 %**	Le sous-composant et le backend ex dans SidebarActivityQuery.tsx et Pr `stopRequested` est latché avant l' aussi `disableAfterClick`.
Désactivation immédiate de Stop	✅	**90 %**	
Activité observer terminée avec panneau ouvert	❌	**40 %**	`onObserverStop(...)` appelle p projection au lieu de la convertir en
Son exact par runRef + attempt	⚠	**80 %**	La déduplication et les attempts son dans PromptQueryDoneSound.mana que par la finalisation des sources n observer.
Delete durable sous claim	✅	**90 %**	Acquisition, fermeture, suppression release `finally` sont présents dans
Retry EasyAgents destructif sous claim	✅	**90 %**	Le claim est acquis avant fermeture jusqu'au `finally` dans EasyAgent
Préservation du comportement EasyAgents	⚠	**70 %**	Identité, replayMode, lifecycle et R observer/cleanup/son restent incomp
Snapshot brut unique et frontière React externe	✅	**95 %**	La séparation `Prompt` / `PromptRun` correctement réalisées dans Prompt.
Désactivation hydration classique et auto-run pour live	✅	**95 %**	Le bootstrap live contrôle bien l'hyd retire les paramètres `autoRun*`.
`start()` synchrone accepté/refusé	✅	**90 %**	Le single-flight synchrone et la géne présents dans useAiStream.hook.ts.
Aucune mutation des commandes sur refus	⚠	**85 %**	Send, SPECIF, Continue, Retry et E retourne toutefois pas le résultat stri
Guard `sourceId + streamInstanceId` des callbacks réseau	✅	**85 %**	`onData`, `onComplete` et `onError`
Guard de `runMerged` déjà sorti de Queue A/B	❌	**20 %**	Les items de Queue A/B ne transpor tentative ; `onMergedChunk(...)` identité.
Publication outcome après Queue A+B+commit	✅	**95 %**	Les deux queues et la barrière React l'identité active.
Direct handoff normalPrompt sans second Open	✅	**90 %**	Le handoff direct attend terminal +

Fonctionnalité technique demandée	État	Complétude estimée	Cons
			+ `finalizedStreamInstanceId`
Retry Open normalPrompt single-flight	❌	**45 %**	Le Retry reste visuellement activé, a plusieurs clics peuvent démarrer plu
Annulation timers/Open à l'unmount	❌	**45 %**	Le cleanup est porté par un effet qui premier rerun.
`preserveLocal` sans lecture draft/attachments/nextTurnId	✅	**95 %**	applyHydratedPromptSessionState.t ces propriétés.
Compositeur normalPrompt + banner sans second Stop	⚠	**75 %**	La composition visuelle est correcte et Stop n'est pas disponible durant c
Barrière fallback `{turnId, streamInstanceId}`	✅	**90 %**	L'ack exact efface ; une finalisation
Continuation et thinking par nouvelle tentative	⚠	**75 %**	Le plan pur et les nouvelles UIDs ex invariants complets, et la sortie Que
Programming docs demandées	✅	**100 %**	Les six documents génériques dema
Tests des courses et isomorphisme	❌	**≈ 25 %**	Plusieurs fichiers portent les bons n composant, ni pipeline réel.
Aucun fichier hors périmètre / généré	❌	**0 %**	Le commit inclut un document pers

Conclusion de la question 2
Une grande partie des structures backend est opérationnelle, mais plusieurs contrats asynchrones et frontend essentiels ne sont pas complètement matérialisés dans le code actuel.

---

3) Y a-t-il des bugs ?
Réponse : ✅ Oui, plusieurs

🔴 **Bug 1 — Une réouverture peut sélectionner une ancienne source retenue**

Dans PromptLiveSessionPanelCoordinator.service.ts, l'injection normalPrompt cherche :

- une réservation quelconque ;
- ou un record avec `ownership === 'source'`.

Elle ne rejette pas les records `sourceSettled` ou `handoffReady`.

Or LiveSessionRegistry.service.ts conserve un ancien record handoff-ready pendant la rétention terminale.

Scénario précis :

1. source A termine et reste retenue ;
2. source B démarre sur le même locator ;
3. B est promue en record ;
4. `listActiveSnapshots()` retourne les anciens records dans leur ordre d'insertion ;
5. le coordinateur peut sélectionner A avant B ;
6. Open(A) voit B active et répond `not_found` ;
7. la réouverture échoue alors que B est bien active.

Sévérité : critique.

🔴 **Bug 2 — Les chunks déjà sortis de Queue A/Queue B ne sont pas gardés par tentative**

Dans useAiStream.hook.ts :

- Queue A reçoit seulement un `AiStreamedChunk` ;
- Queue B reçoit seulement un `AiStreamedChunk` ;
- `runMerged(...)` ne capture ni `sourceId` ni `streamInstanceId` ;
- `onMergedChunk(...)` applique directement les setters React.

Dans useAiStreamQueue.hook.ts, Queue A n'expose aucun reset de génération.

Dans usePromptStreamDispatchGateQueue.hook.ts, un reset efface les conteneurs, mais ne peut pas annuler une continuation déjà entrée dans un `await`.

Conséquences possibles :

- un token ancien peut être appendu au successeur ;
- un ancien `fallback_persisted` peut être associé à l'identité courante ;
- une continuation post-`await` de `tool_calls` peut muter le nouveau stream ;
- les sanitizers ou locks du successeur peuvent recevoir une sortie de l'ancienne tentative.

Sévérité : critique, course dépendante du timing.

🔴 **Bug 3 — Retry Open n'est pas single-flight et le cleanup d'unmount devient inactif**

Dans usePromptLiveSession.hook.ts :

- chaque appel à `open(...)` crée immédiatement une nouvelle requête ;
- aucun `openInFlightRef` ne bloque un second appel ;
- `retryHandoff()` ne passe pas le mode vers un état qui désactive Retry ;
- PromptLiveSessionBanner.tsx n'utilise pas `disableAfterClick` pour Retry.

Le cleanup d'unmount est également fragile :

1. l'effet bootstrap dépend de `open` ;
2. `open` dépend du grand objet `params` ;
3. après un changement d'état, l'effet précédent exécute son cleanup ;
4. le nouvel effet retourne immédiatement à cause de `bootstrappedRef.current` ;
5. ce nouvel effet ne fournit plus de cleanup ;
6. les futurs Retry/timers ne sont donc plus systématiquement annulés à l'unmount.

Sévérité : élevée.

🔴 **Bug 4 — Le compositeur devient interactif avant la baseline et peut perdre des attachments locaux**

Dans NormalPromptLiveSession.adapter.ts, `composerInteractive` vaut `true` pour tous les états live non normaux, y compris `liveSessionOpening`.

L'utilisateur peut donc modifier le compositeur ou préparer des attachments pendant que Open retourne encore `not_ready`.

Lorsque Open devient prêt, l'hydratation initiale `hydrateFromSnapshot` de applyHydratedPromptSessionState.util.ts :

- protège partiellement le texte via `hasLocalPromptOverrideRef` ;
- mais remplace les attachments sans protection équivalente ;
- remplace aussi le `nextTurnId` de composition.

Cela viole l'exigence « interactif seulement après baseline installée et replay armé ».

Sévérité : élevée, risque de perte d'état local.

🔴 **Bug 5 — Stop n'est pas disponible dans plusieurs états où la source continue**

Le Stop du compositeur dépend de `transportStopAvailable`, lui-même armé par `useAiStream.start()`.

Avant que l'observer soit accepté :

- durant `not_ready` ;
- après une erreur d'hydratation ;
- après certains échecs Open/replay ;

la source backend peut encore être active, mais :

- `transportStopAvailable === false` ;
- la banner normalPrompt interdit son propre Stop ;
- le compositeur affiche Send désactivé plutôt que Stop.

L'utilisateur doit fermer le panneau pour obtenir éventuellement le Stop détaché.

Sévérité : élevée.

🟠 **Bug 6 — Le coordinateur autorise un cleanup alors qu'un record observer est actif**

La méthode de recherche d'autorité dans PromptLiveSessionPanelCoordinator.service.ts ne considère que :

- les réservations ;
- les records `ownership='source'`.

Mais EasyAgents enregistre volontairement ses runs avec `ownership='observer'` dans EasyAgentsBatchLiveSessionAdapter.service.ts.

Le coordinateur peut alors appeler le cleanup Prompt alors que le record EasyAgents est encore actif. Selon l'existence d'un mapping de thread, cela peut déclencher prématurément :

- orphan cleanup ;
- trivial cleanup ;
- voire le chemin Abort → AwaitStop → ClearMemory dans un cas de panneau réutilisé ou préalablement enregistré.

Sévérité : élevée.

🟠 **Bug 7 — Un observer terminé disparaît d'Activité au lieu de devenir Idle**

Dans SidebarQueryRunningBridge.ts, `onObserverStop(...)` appelle `panelClosed(...)`.

Or `panelClosed(...)`, dans SidebarStore.queryWindow.mutations.manager.ts, supprime tout observer.
Si le panneau est encore ouvert après handoff EasyAgents, l'item devrait devenir Idle/open ; il disparaît à la place.
Sévérité : moyenne à élevée, régression UX visible.

🟠 **Bug 8 — L'état de son exact EasyAgents n'est jamais nettoyé**

`cleanupPromptQueryDoneSoundRun(...)` n'est appelé que par la finalisation de source normale dans le coordinateur.
Le chemin EasyAgents :

- appelle `onAiStreamStart(...)` ;
- observe le terminal ;
- appelle `onAiStreamStop(...)` ;
- mais ne supprime jamais l'entrée exacte de `stateByRunKey`.

Chaque run EasyAgents regardé peut donc laisser un état process-local retenu.
Sévérité : moyenne, fuite mémoire process-lifetime.

🟠 **Bug 9 — Les locators dupliqués ne sont pas rejetés avant le low-level opener**

Le parseur extension PromptLiveSessionPanelQuery.util.ts vérifie les doublons des paramètres live, mais pas ceux de `sessionId` ou `storageScopeRelPath`.
Le low-level opener sélectionne ensuite la première occurrence. Le frontend finit par rejeter la query, mais seulement après création du panneau.
Cela viole explicitement l'invariant « query live invalide rejetée avant le low-level opener ».
Sévérité : moyenne.

🟡 **Bug 10 — Un échec transitoire de persistance peut remplacer le terminal original**

Dans NormalPromptLiveSessionSource.service.ts :

1. un `meta.error` entrant tente sa persistance ;
2. si cette persistance échoue, le chunk n'est pas appendu ;
3. le catch voit `hasTerminalMeta === false` ;
4. il synthétise un `INTERRUPTED` ;
5. si la persistance de ce second terminal réussit, le terminal original est perdu.

La règle demandait qu'un wrapper synthétique ne remplace pas un terminal source déjà observé.
Sévérité : moyenne, cas d'incident de persistance transitoire.

🔴 **Défaut de vérification — Les tests critiques ne testent pas le comportement annoncé**

Exemples vérifiés :

- usePromptLiveSession.hydration.test.tsx compare uniquement deux littéraux de type ;
- usePromptLiveSession.handoff.test.tsx compare uniquement deux politiques ;
- usePromptLiveSession.normalPromptIsomorphism.test.tsx parse un DTO, sans comparaison streaming/replay/hydratation ;
- useAiStream.hook.promptSourceIdentity.test.ts construit deux objets sans monter le hook ;
- useAiStream.hook.liveSessionObserver.test.ts ne monte pas non plus le hook ;
- usePromptRunCommands.startAcceptance.test.tsx ne teste aucune des sept commandes ;
- Prompt.fallbackPersistAck.clearInput.test.tsx simule un objet local sans exécuter Prompt ;
- PromptLiveSessionPanelCoordinator.service.test.ts ne couvre ni dispose, ni settle, ni wake-up, ni claim, ni source successive.

Les noms des tests correspondent à la spécification, mais leur contenu ne prouve pas les comportements annoncés.
Sévérité qualité : critique.

⚠ **Défaut de périmètre**

Le commit inclut une modification hors périmètre dans document personnel hors périmètre. Son contenu n'a pas été inspecté.
Il modifie également sept fichiers générés, notamment :

- allEndpoints.generated.ts
- callLocalApi.generated.ts
- ProxyLocalApiMessages.generated.ts

Le diff ne permet pas de déterminer s'ils ont été produits automatiquement ou édités manuellement, mais **l'état versionné ne respecte pas la règle projet qui interdit de modifier les `*.generated.*`**.

Conclusion de la question 3
Oui : au moins neuf défauts techniques ou contractuels sont identifiables, dont plusieurs courses de haute sévérité.

---

4) Quel est mon avis sur le code source et la spécification ?
Avis sur la spécification : ✅ Excellent — 9,5/10
Points forts :

- distinction très nette entre identité logique, identité source et identité frontend ;
- invariants asynchrones explicités ;
- excellent traitement des ABA, tombstones, claims et retraites ;
- responsabilité terminale clairement attribuée ;
- attention exceptionnelle à l'isomorphisme streaming/hydratation ;
- scénarios de test très complets ;
- gestion réaliste des erreurs après header, des sources successives et des suppressions destructives.

Point faible principal :

- la tranche est extrêmement large et atomique : **160 fichiers** et plusieurs sous-systèmes critiques dans le même changement ;
- le risque annoncé de **92 %** était cohérent et même plutôt lucide.

Avis sur le code : ⚠ Architecture prometteuse, preuve insuffisante — 6/10
Les meilleurs éléments sont :

- contrats shared stricts ;
- modularisation source/lifecycle/coordinator ;
- séparation `sourceId`, `streamInstanceId`, `replayOpenAttemptId` ;
- réservations et claims bien conçus ;
- ordre durable-before-append correctement centralisé ;
- parser Slimbk post-header propre et isolé ;
- frontière React externe bien construite ;
- commentaires techniques nombreux et utiles.

Les faiblesses déterminantes sont :

- plusieurs invariants écrits dans les commentaires ne sont pas effectivement gardés dans le code ;
- la coordination `observer` n'est pas terminée ;
- la sélection active/retained est incorrecte ;
- la pipeline Queue A/B n'a pas reçu l'identité demandée ;
- le state machine Open/Retry manque d'un vrai single-flight ;
- le compositeur normalPrompt s'ouvre trop tôt ;
- les tests frontend critiques ont été fortement réduits et remplacés par des assertions de constantes ou de types.

Autrement dit : **la charpente est sérieuse, mais plusieurs boulons asynchrones sont encore posés sur la table — et les tests font parfois semblant de tenir la clé plate 🔧.**
Conclusion de la question 4
La spécification est de très haute qualité ; l'implémentation contient de bonnes décisions architecturales, mais elle n'a pas atteint le niveau de sûreté demandé.

---

5) Résumé du rapport

Axe audité	Score estimé	Verdict	Résumé
Inventaire des 160 fichiers spécifiés	**100 %**	✅	Tous les chemins demandés sont représentés
Contrats shared	**95 %**	✅	Identités, descripteurs et Open stricts
Registre liveSession	**90 %**	✅	Réservations, époques, claims et retraites solides
Source Prompt normale	**85 %**	⚠	Bon ordre durable/append, mais terminal original remplaçable sur incident transitoire
Protocole transport	**95 %**	✅	V1 et erreurs body post-header correctement séparés
Origine webview exacte	**100 %**	✅	Guard extension-only correctement placé
Coordination panneau	**50–60 %**	❌	Mauvaise source possible, observer ignoré pendant cleanup
Activité et Stop	**75 %**	⚠	Normal source correct, observer terminal incorrect
Son	**80 %**	⚠	Attempts exacts, cleanup EasyAgents manquant

Axe audité	Score estimé	Verdict	Résumé
Delete / Retry destructifs	**90 %**	✅	Claims et handles exacts présents
Bootstrap frontend	**90 %**	⚠	Bonne frontière React, validation extension locator incomplète
Start transactionnel	**80 %**	⚠	Accept/refuse correct, Reply et sorties Queue incomplets
Queue A/B et guards de tentative	**50 %**	❌	Les sorties déjà en vol ne portent pas l'identité
Open/replay/handoff frontend	**65 %**	⚠	Direct handoff présent, Retry/unmount/Stop incomplets
`preserveLocal`	**95 %**	✅	Les champs compositeur ignorés ne sont réellement pas lus
Compositeur et fallback ack	**75 %**	⚠	Barrière correcte, interaction live activée trop tôt
Thinking/Continue	**75 %**	⚠	Plans présents, sûreté inter-tentatives non démontrée
Programming docs	**100 %**	✅	Documentation ciblée mise à jour
Tests comportementaux	**25–30 %**	❌	Plusieurs tests critiques sont uniquement déclaratifs
Respect du périmètre	**60 %**	❌	9 fichiers supplémentaires, dont document personnel et fichiers générés
Conformité globale	**≈ 68 %**	❌	Non finalisé, corrections nécessaires
Aptitude à livraison	—	❌	Trop de courses et de comportements non testés

---

6) Conclusion en une phrase

Cette implémentation pose une fondation architecturale convaincante, mais elle ne peut pas être considérée comme finalisée tant que les courses de réouverture, les guards Queue A/B, le cycle observer, le state machine Open/Retry et les vrais tests comportementaux ne sont pas corrigés.

---

7) Ensembles de corrections proposés

Lot A — Coordination panneau, Activité et EasyAgents

Corrections proposées :

- sélectionner exclusivement la réservation ou le record source réellement actif ;
- exclure les records settled/handoff-ready de l'injection normalPrompt ;
- considérer les records observer actifs avant tout cleanup ;
- faire passer observer/open vers Idle au terminal ;
- nettoyer l'état son exact EasyAgents ;
- rejeter les locators dupliqués avant le low-level opener ;
- ajouter les tests de sources successives, observer close/settle, wake-ups et cleanup.

Priorité : critique.

Lot B — Identités frontend, Queue A/B et Open/Retry

Corrections proposées :

- associer chaque entrée Queue A/B à `sourceId + streamInstanceId` ;
- vérifier cette identité avant et après chaque `await` ;
- empêcher un ancien `fallback_persisted` de viser le successeur ;
- ajouter un vrai single-flight Open ;
- annuler proprement timers et requêtes à l'unmount ;
- désactiver Retry pendant la requête ;
- fournir Stop avant même l'acceptation de l'observer lorsque la source exacte est encore active ;
- rendre le compositeur interactif seulement après baseline + replay armé ;
- retourner le résultat strict depuis Reply.

Priorité : critique.

Lot C — Durabilité, tests et hygiène du commit

Corrections proposées :

- préserver le terminal `meta.error` original lorsqu'une persistance échoue ;
- reconstruire les véritables tests de hooks, composants, queues et courses ;
- restaurer les scénarios supprimés des tests fallback/hydratation/handoff ;

- ajouter les getters piégés et comparaisons d’isomorphisme réelles ;
- retirer la modification du document personnel ;
- retirer du commit les modifications des fichiers générés ;
- conserver uniquement les sources de génération et les tests métier.

Priorité : élevée.

Lot D — Toutes les corrections

Ce lot applique **A + B + C**, avec les tests de non-régression complets et la remise en conformité stricte du périmètre.

Priorité recommandée : maximale.

TL-DR

Le backend liveSession est bien avancé, mais la tranche reste non finalisée à cause de plusieurs courses frontend/panneau et d’une couverture de tests très inférieure à celle promise par la spécification.

🧩 Coder le lot panneau et Activité

🛡️ Coder le lot frontend et identités

🧪 Coder le lot tests et hygiène

🚀 Coder toutes les corrections

</ASSISTANT>

<USER>

[Coder toutes les corrections]

</USER>

<ASSISTANT>

I'm thinking...

</ASSISTANT>

<ASSISTANT>

🔧 ✅ *reading*

project:

🔎 r *"listActiveSnapshots"*,

🔎 r *"listReservedSnapshots"*,

🔎 r *"onObserverStop"*,

🔎 r *"cleanupPromptQueryDoneSoundRun"*,

🔎 r *"runMerged"*, 🔎 r *"enqueue"*, 🔎 r *"retryHandoff"*, 🔎 r *"transportStopAvailable"*, 🔎 h *"Harnais hooks streaming"*, 🔎 h *"Transitions activité"*, 📄 *LiveSessionService.ts* [L120-L220], 📄 *LiveSessionRegistry.service.ts* [L740-L830], 📄 *LiveSessionRecord.type.ts* [L1-L260],

📄 *PromptLiveSessionPanelCoordinator.service.ts*,

📄 *PromptLiveSessionPanelQuery.util.ts*,

📄 *SidebarQueryRunningBridge.ts*,

📄 *SidebarStore.queryWindow.listeners.manager.ts*,

📄 *liveSessionStreamHandlers.ts*,

📄 *usePromptStreamDispatchGateQueue.hook.ts*,

📄 *useAiStreamQueue.hook.ts*, 📄 *PromptQueryDoneSound.manager.ts* [L1-L180], 🔎 r *"useSerializedStreamQueue"*, 🔎 e *"Usages queue A"*, 📄 *AiStreamStartArgs.type.ts* [L1-L240], 📄 *useAiStream.hook.ts* [L130-L540], 📄 *useAiStream.hook.ts* [L540-L867], 📄 *usePromptLiveSession.hook.ts*, 📄 *NormalPromptLiveSession.adapter.ts*, 🔎 e *"Stop compositeur wiring"*, 📄 *Prompt.tsx* [L320-L460], 📄 *Prompt.tsx* [L720-L900], 📄 *Prompt.tsx* [L630-L725], 📄 *usePromptCancelController.hook.ts* [L1-L230], 🔎 e *"callLocalApi Prompt import"*, 📄 *abortSource.endpoint.ts* [L1-L160], 📄 *Prompt.tsx* [L1180-L1255], 📄 *usePromptRunCommands.hook.ts* [L580-L691], 🔎 e *"Définition Reply"*, 📄 *usePromptRunCommands.hook.ts* [L530-L590], 🔎 e *"Reply callback types"*, 📄 *Prompt.tsx* [L1335-L1365], 📁 f *"Tests Queue B"*, 🔎 e *"EasyAgents réservation"*, 📄 *SidebarStore.queryWindow.mutations.manager.ts* [L1-L120], 🔎 h *"Locator query constants"*, 📄 *ExtensionConfigConst.const.ts* [L75-L100], 📄 *openPanel.util.ts* [L600-L670], 🔎 h *"Harnais hook liveSession"*, 📄 *Prompt.liveSessionBootstrap.test.tsx*, 📄 *Prompt.tsx* [L140-L320], 📄 *NormalPromptLiveSessionSource.service.ts* [L140-L294], 📄 *NormalPromptLiveSessionSource.service.test.ts* [L430-L515],

PromptLiveSessionPanelCoordinator.service.test.ts,
PromptLiveSessionPanelQuery.util.test.ts,
SidebarQueryRunningBridge.race.test.ts,
AiStreamMergedQueue.test.ts,
PromptStreamDispatchGateQueue.test.ts,
NormalPromptLiveSession.adapter.test.ts,
usePromptRunCommands.startAcceptance.test.tsx, LiveSessionRegistry.service.test.ts [L1-L180],
useAiStream.hook.rawLiveSideband.test.ts,
usePromptRunCommands.continueResponse.test.tsx, usePromptRunCommands.hook.ts [L1-L180], useAiStream.hook.ts [L1-L150], e "Usages stop requested", e "Fixtures réservation snapshot", e "Normal Prompt viewer group", e "Mocks listeners manager", e "PromptLiveSessionApi refs", e "Mocks Queue A hook", e "Mocks module Queue A", e "Mocks Queue B hook", LiveSessionOpenResult.type.ts [L1-L180], e "Live Open session fixture", h "Chat session minimal fixture", LiveSessionOpen.service.test.ts [L1-L50], applyHydratedPromptSessionState.util.test.ts [L1-L220], SidebarStore.rehydrate.manager.ts [L100-L210], e "Réservation supposée source", e "Mocks usePromptLiveSession"

patching file: LiveSessionRecord.type.ts

AICode/src/extension/src/aiTools/orch/liveSession/registry/LiveSessionRecord.type.ts

Copy
Review
Accept
PatchFull file

```
-     readonly viewerGroupId: string;
-     readonly replayMode: LiveSessionReplayMode;
-
+     readonly viewerGroupId: string;
+     /** Reservation ownership is required by panel coordination and Activity rehydration. */
+     readonly ownership: LiveSessionOwnership;
+     readonly replayMode: LiveSessionReplayMode;
+
```

```
-
- /** Deeply immutable notification emitted after an observer group becomes inactive. */
-
+
+ /** Deeply immutable cleanup wake-up emitted when an observer group is inactive or later settles inactive. */
+
```

AICode/src/extension/src/aiTools/orch/liveSession/registry/LiveSessionRecord.type.ts

Copy
Review
Accept

patching file: LiveSessionRegistry.service.ts

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

Copy
Review
Accept
PatchFull file

```
-         }
-         record.sourceSettled = true;
-
+         }
+         const observerGroupInactive = !this.hasViewerForGroupUnsafe(
+             record.viewerGroupId,
+         );
```

```
+         record.sourceSettled = true;
+
```

```
-             this.retireRecord(record, 'retiredMissingBaseline');
-             return true;
-
+             this.retireRecord(record, 'retiredMissingBaseline');
+             if (observerGroupInactive) this.emitObserverInactive(record);
+             return true;
+
```

```
-             this.retireRecord(record, 'retiredMissingTerminal');
-             return true;
-
+             this.retireRecord(record, 'retiredMissingTerminal');
+             if (observerGroupInactive) this.emitObserverInactive(record);
+             return true;
+
```

```
-         this.signal(record);
-     }
-
+         this.signal(record);
+         if (!this.hasViewerForGroupUnsafe(record.viewerGroupId)) {
+             // A detached panel may already have consumed the lease-release wake-up
before settlement completed.
+             this.emitObserverInactive(record);
+         }
+     }
+
```

```
-                     viewerGroupId: reservation.viewerGroupId,
-
+                     viewerGroupId: reservation.viewerGroupId,
+                     ownership: reservation.ownership,
+
```

```
-
-     /** Emit observer inactivity after all registry mutations completed. */
-
+
+     /** Emit an idempotent observer-inactive cleanup wake-up after all registry
mutations completed. */
+
```

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

Copy

Review

Accept

patching file: *SidebarStore.rehydrate.manager.ts*

AICode/src/extension/src/extension/panels/sidebar/SidebarStore.rehydrate.manager.ts

Copy

Review

Accept

PatchFull file

```
-                 isSameQueryActivityLocator(snapshot.runRef.locator, locator) &&
-
+                 isSameQueryActivityLocator(snapshot.runRef.locator, locator) &&
+                 snapshot.ownership === 'source' &&
+
```

```
-                 (snapshot.kind === 'reservation' ||
-                     (snapshot.ownership === 'source' &&
```

```
-                         !snapshot.sourceSettled &&
-                         !snapshot.handoffReady)),
-
+                 (snapshot.kind === 'reservation' ||
+                     (!snapshot.sourceSettled && !snapshot.handoffReady)),
+
```

```
-         const open = openLocators.get(key);
-         const ownership =
-             snapshot.kind === 'record' ? snapshot.ownership : 'source';
-
+         const open = openLocators.get(key);
+         // Reservations retain their real ownership so observer-only sources can never
become detached source rows.
+         const ownership = snapshot.ownership;
+
```

AICode/src/extension/src/extension/panels/sidebar/SidebarStore.rehydrate.manager.ts

Copy

Review

Accept

🔧 ✅ *patching file*: *PromptLiveSessionPanelCoordinator.service.ts*

AICode/src/extension/src/extension/panels/prompt/liveSession/PromptLiveSessionPanelCoordinator.service.ts

Copy

Review

Accept

PatchFull file

```
-             if (explicit.kind === 'classic') {
-                 const source = this.deps.liveSession
-                     .listActiveSnapshots()
-                     .find(
-                         (snapshot) =>
-                             snapshot.runRef.locator.sessionId ===
-                                 args.locator.sessionId &&
-                             snapshot.runRef.locator.storageScopeRelPath ===
-                                 args.locator.storageScopeRelPath &&
-                             (snapshot.kind === 'reservation' ||
-                                 snapshot.ownership === 'source'),
-                     );
-
+             if (explicit.kind === 'classic') {
+                 // Only the current non-settled normal Prompt source may drive implicit
injection.
+                 // Retained handoff snapshots and EasyAgents observer reservations are
intentionally excluded.
+                 const source = this.findActiveNormalPromptSource(args.locator);
+
```

```
-                 }
-             }
-
+                 }
+             }
+             const locatorKey = this.locatorKey(args.locator);
+
```

```
-             if (descriptor) {
-                 this.descriptorByLocator.set(
-                     this.locatorKey(args.locator),
-                     descriptor,
-                 );
-
```

```
+             if (descriptor) {
+                 this.descriptorByLocator.set(locatorKey, descriptor);
+
──────────────────────────────────────────────────────────────────────────────
-             }
-             this.detachedLocatorKeys.delete(this.locatorKey(args.locator));
-
+             }
+             this.detachedLocatorKeys.delete(locatorKey);
+
──────────────────────────────────────────────────────────────────────────────
-             }
-             const active = this.findActiveSource(locator);
-
+             }
+             const active = this.findActiveAuthority(locator);
+
──────────────────────────────────────────────────────────────────────────────
-         if (
-             this.findActiveSource(locator) ||
-
+         if (
+             this.findActiveAuthority(locator) ||
+
──────────────────────────────────────────────────────────────────────────────
-         }
-         this.detachedLocatorKeys.delete(key);
-
+         }
+         const descriptor = this.descriptorByLocator.get(key);
+         this.detachedLocatorKeys.delete(key);
+
──────────────────────────────────────────────────────────────────────────────
-         await cleanupOnPromptDispose(locator);
-
+         await cleanupOnPromptDispose(locator);
+         if (descriptor) {
+             // Exact sound state survives replay attempts and is retired only after all
live authority disappeared.
+             cleanupPromptQueryDoneSoundRun({
+                 locator,
+                 turnId: descriptor.liveSessionTurnId,
+                 sourceId: descriptor.liveSessionSourceId,
+             });
+         }
+
──────────────────────────────────────────────────────────────────────────────
-
-     /** Find source authority independently from stale Sidebar projections. */
-     private findActiveSource(locator: ChatSessionLocator) {
-
+
+     /** Find the exact active normal-Prompt source eligible for implicit live-session
injection. */
+     private findActiveNormalPromptSource(locator: ChatSessionLocator) {
+         return this.deps.liveSession
+             .listActiveSnapshots()
+             .find(
+                 (snapshot) =>
+                     snapshot.runRef.locator.sessionId === locator.sessionId &&
+                     snapshot.runRef.locator.storageScopeRelPath ===
+                         locator.storageScopeRelPath &&
+                     snapshot.ownership === 'source' &&
+                     (snapshot.kind === 'reservation' ||
+                         (!snapshot.sourceSettled && !snapshot.handoffReady)),
+             );
```

```
+     }
+
+     /** Find any reservation or record that still owns destructive Prompt cleanup
authority. */
+     private findActiveAuthority(locator: ChatSessionLocator) {
+
```

```
-                     (snapshot.kind === 'reservation' ||
-                         (snapshot.ownership === 'source' &&
-                             !snapshot.sourceSettled &&
-                             !snapshot.handoffReady)),
-
+                     (snapshot.kind === 'reservation' ||
+                         (snapshot.ownership === 'observer'
+                             ? !snapshot.handoffReady
+                             : !snapshot.sourceSettled &&
+                               !snapshot.handoffReady)),
+
```

AICode/src/extension/src/extension/panels/prompt/liveSession/PromptLiveSessionPanelCoordinator.service.ts
Copy
Review
Accept

patching file: *PromptLiveSessionPanelQuery.util.ts*
AICode/src/extension/src/extension/panels/prompt/liveSession/PromptLiveSessionPanelQuery.util.ts
Copy
Review
Accept
PatchFull file

```
- import { randomUUID } from 'node:crypto';
-
-
+ import { randomUUID } from 'node:crypto';
+
+ import { ExtensionConfigConst } from
'../../../../sharedlib/lib/const/ExtensionConfigConst.const.js';
+
```

```
-     if (liveEntries.length === 0) return { kind: 'classic' };
-
+     if (liveEntries.length === 0) return { kind: 'classic' };
+
+     // A live query must never let the low-level opener silently select the first
locator occurrence.
+     for (const locatorParam of [
+         ExtensionConfigConst.DEEP_LINK_PARAM_SESSION_ID,
+         ExtensionConfigConst.DEEP_LINK_PARAM_STORAGE_SCOPE_REL_PATH,
+     ] as const) {
+         const occurrenceCount = (query ?? []).filter(
+             (entry) => entry.name === locatorParam,
+         ).length;
+         if (occurrenceCount > 1) {
+             return {
+                 kind: 'invalid',
+                 message: `Duplicate live-session locator parameter: ${locatorParam}`,
+             };
+         }
+     }
+
+
```

AICode/src/extension/src/extension/panels/prompt/liveSession/PromptLiveSessionPanelQuery.util.ts
Copy

Review
Accept

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

```
-         running: false,
-     });
- }
-
+         running: false,
+     });
+ }
+
+ /** Final non-stale observer transition: its still-open Prompt panel becomes Idle. */
+ export async function observerSettled(runRef: LiveSessionRunRef): Promise<void> {
+     await updateSidebarStore((store) => {
+         const items = store.sidebarContentState.activityState.queryState.items;
+         const index = items.findIndex(
+             (candidate) =>
+                 candidate.status === 'running' &&
+                 candidate.ownership === 'observer' &&
+                 candidate.runRef.turnId === runRef.turnId &&
+                 candidate.runRef.sourceId === runRef.sourceId &&
+                 isSameQueryActivityLocator(candidate, runRef.locator),
+         );
+         if (index < 0) return;
+         const current = items[index];
+         if (current.status !== 'running' || current.ownership !== 'observer')
+             return;
+
+         // Observer activities can exist only while their panel is open, so settlement
always projects Idle/open.
+         items[index] = {
+             status: 'idle',
+             panelState: 'open',
+             sessionId: current.sessionId,
+             storageScopeRelPath: current.storageScopeRelPath,
+             title: current.title,
+             lastQueryElapsedMs: current.liveQueryElapsedMs,
+             usage: current.usage,
+             updatedAt: Date.now(),
+             openedAt: current.openedAt,
+         };
+     });
+     emitSafely(runningListeners, {
+         locator: runRef.locator,
+         running: false,
+     });
+ }
+
```

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

patching file: *SidebarQueryRunningBridge.ts*

AICode/src/extension/src/messageBus/handlers/ia/SidebarQueryRunningBridge.ts
Copy
Review
Accept
PatchFull file

```
- import type { ChatSessionLocator } from
'../../../sharedlib/types/chatTypes/ChatSessionLocator.type.js';
- import {
-
+ import type { ChatSessionLocator } from
'../../../sharedlib/types/chatTypes/ChatSessionLocator.type.js';
+ import {
+     observerSettled,
+
```

```
- } from
'../../../extension/panels/sidebar/SidebarStore.queryWindow.listeners.manager.js';
- import {
-     panelClosed,
-     setLiveQueryElapsedForWindow,
- } from
'../../../extension/panels/sidebar/SidebarStore.queryWindow.mutations.manager.js';
-
+ } from
'../../../extension/panels/sidebar/SidebarStore.queryWindow.listeners.manager.js';
+ import { setLiveQueryElapsedForWindow } from
'../../../extension/panels/sidebar/SidebarStore.queryWindow.mutations.manager.js';
+
```

```
-
-     /** Observer cleanup removes its runtime and delegates panel-aware removal. */
-
+
+     /** Observer cleanup removes its runtime and projects the still-open Prompt panel
back to Idle. */
+
```

```
-         this.runtimeByLocator.delete(this.locatorKey(runRef.locator));
-         await panelClosed({ locator: runRef.locator });
-
+         this.runtimeByLocator.delete(this.locatorKey(runRef.locator));
+         await observerSettled(runRef);
+
```

AICode/src/extension/src/messageBus/handlers/ia/SidebarQueryRunningBridge.ts
Copy
Review
Accept

patching file: *NormalPromptLiveSessionSource.service.ts*
AICode/src/extension/src/aiTools/orch/liveSession/source/NormalPromptLiveSessionSource.service.ts
Copy
Review
Accept
PatchFull file

```
-         let hasTerminalMeta = false;
-
+         let hasTerminalMeta = false;
+         let terminalMetaObserved = false;
+
```

```
-                     const chunk = this.deps.parseChunk(raw);
-
```

```
+                     const chunk = this.deps.parseChunk(raw);
+
+                     // Observation precedes durability so a failed terminal patch can
never be replaced by a synthetic error.
+                     terminalMetaObserved ||=
+                         isLiveSessionTerminalMetaChunk(chunk);
+
```

```
-             // A synchronous iterator-priming failure leaves only a reservation, so no
registry append is legal.
-             if (sourceStarted && !hasTerminalMeta) {
-
+             // A synchronous iterator-priming failure leaves only a reservation, so no
registry append is legal.
+             // A terminal already observed from Orch remains authoritative even if its
durability barrier failed.
+             if (sourceStarted && !terminalMetaObserved) {
+
```

AICode/src/extension/src/aiTools/orch/liveSession/source/NormalPromptLiveSessionSource.service.ts
Copy
Review
Accept

patching file: *usePromptRunCommands.hook.ts*
AICode/src/frontend/src/app/prompt/Prompt/usePromptRunCommands.hook.ts
Copy
Review
Accept
PatchFull file

```
-     onBeforeSubmitUi: () => void;
-     handleReplyButton: (label: string) => void;
-
+     onBeforeSubmitUi: () => void;
+     handleReplyButton: (label: string) => AiStreamStartResult;
+
```

```
-     const handleReplyButton = useCallback(
-         (label: string): void => {
-
+     const handleReplyButton = useCallback(
+         (label: string): AiStreamStartResult => {
+
```

```
-             // send like a prompt message
-             handleSend(`[${labelWithoutEmoji}]`, aiQueryOptionsBase, []);
-
+             // send like a prompt message
+             return handleSend(
+                 `[${labelWithoutEmoji}]`,
+                 aiQueryOptionsBase,
+                 [],
+             );
+
```

AICode/src/frontend/src/app/prompt/Prompt/usePromptRunCommands.hook.ts
Copy
Review
Accept

patching file: *NormalPromptLiveSession.adapter.ts*
AICode/src/frontend/src/app/prompt/liveSession/adapters/NormalPromptLiveSession.adapter.ts

Copy
Review
Accept
PatchFull file

```
-
- /** Build the normal Prompt surface: composer is usable, Send stays locked, and Stop
remains unique in textarea. */
-
+
+ /** Build the normal Prompt surface: composer unlocks after replay arming, Send stays
locked, and Stop stays unique. */
+
```

```
-         args.mode === 'liveSessionReplayFailed';
-
+         args.mode === 'liveSessionReplayFailed';
+     const replayBaselineInstalled = args.mode === 'liveSessionStreaming';
+
```

```
-         showComposer: true,
-         composerInteractive: true,
-
+         showComposer: true,
+         // Local draft and attachment edits are allowed only after Open hydration and
observer arming completed.
+         composerInteractive: replayBaselineInstalled,
+
```

```
-             showRetryAction: failed,
-             retryEnabled: failed,
-
+             showRetryAction: failed,
+             retryEnabled: failed && args.stopEnabled,
+
```

AICode/src/frontend/src/app/prompt/liveSession/adapters/NormalPromptLiveSession.adapter.ts
Copy
Review
Accept

🔧 ✅ *patching file*: 📄 *AiStreamStartArgs.type.ts*
AICode/src/frontend/src/messageBus/AiStreamStartArgs.type.ts
Copy
Review
Accept
PatchFull file

```
-
- export type AiStreamStartResult = { accepted: false } | AiStreamAcceptedStart;
-
+
+ /** Frontend-only identity carried through Queue A and Queue B for every accepted
stream mutation. */
+ export type AiStreamAttemptIdentity = {
+     readonly sourceId: LiveSessionRunRef['sourceId'];
+     readonly streamInstanceId: AiStreamAcceptedStart['streamInstanceId'];
+ };
+
+ export type AiStreamStartResult = { accepted: false } | AiStreamAcceptedStart;
+
```

AICode/src/frontend/src/messageBus/AiStreamStartArgs.type.ts
Copy
Review

Accept

patching file: *useAiStreamQueue.hook.ts*
AICode/src/frontend/src/messageBus/useAiStreamQueue.hook.ts
Error
(what is it?)

reading project: *useAiStreamQueue.hook.ts* [L30-L105], *useAiStreamQueue.hook.ts* [L230-L345], *useAiStreamQueue.hook.ts* [L380-L445]
patching file: *useAiStreamQueue.hook.ts*
AICode/src/frontend/src/messageBus/useAiStreamQueue.hook.ts
Copy
Review
Accept
PatchFull file

```
-  * - The queue is single-consumer, in-order, and uses a tail Promise reference.
-  * - Items are *chunks* (`AiStreamedChunk`) only; no per-item promises, no external task closures.
-
+  * - The queue is single-consumer, in-order, and uses a tail Promise reference.
+  * - Items contain one `AiStreamedChunk` plus its exact frontend attempt identity; no task closures are queued.
+
```

```
-  * Integration notes:
-  * - The queue accepts raw chunks via `enqueueChunk(chunk)`.
-
+  * Integration notes:
+  * - Production callbacks use `enqueueChunkForAttempt(chunk, identity)` so retries cannot inherit stale output.
+  * - Direct callers may use `enqueueChunk(chunk)`, which resolves the current identity through the injected reader.
+
```

```
- import { requestAnimationFrameTimeout } from '../lib/requestAnimationFrameTimeout.util.js';
-
+ import { requestAnimationFrameTimeout } from '../lib/requestAnimationFrameTimeout.util.js';
+ import type { AiStreamAttemptIdentity } from './AiStreamStartArgs.type.js';
+
```

```
-     enqueueChunk: (chunk: AiStreamedChunk) => void;
-
+     enqueueChunk: (chunk: AiStreamedChunk) => void;
+     /** Enqueue with the exact callback-captured attempt identity instead of reading current state. */
+     enqueueChunkForAttempt: (
+         chunk: AiStreamedChunk,
+         attemptIdentity: AiStreamAttemptIdentity,
+     ) => void;
+
```

```
-
- /** Await a short, well-defined sequence of yields intended to let React commit and run effects. */
-
+
+ /** Raw Queue A entry that keeps every mutation attached to one accepted frontend stream attempt. */
+ interface QueuedStreamChunk {
```

```
+     readonly chunk: AiStreamedChunk;
+     readonly attemptIdentity: AiStreamAttemptIdentity;
+ }
+
+ /** Compare the exact source and stream identities that delimit Queue A coalescence. */
+ function isSameAttempt(
+     left: AiStreamAttemptIdentity,
+     right: AiStreamAttemptIdentity,
+ ): boolean {
+     return (
+         left.sourceId === right.sourceId &&
+         left.streamInstanceId === right.streamInstanceId
+     );
+ }
+
+ /** Await a short, well-defined sequence of yields intended to let React commit and run
effects. */
+
```

```
- export function useSerializedStreamQueue(opts: {
-     runMerged: (chunk: AiStreamedChunk) => void | Promise<void>;
-
+ export function useSerializedStreamQueue(opts: {
+     runMerged: (
+         chunk: AiStreamedChunk,
+         attemptIdentity: AiStreamAttemptIdentity,
+     ) => void | Promise<void>;
+     getAttemptIdentity: () => AiStreamAttemptIdentity | undefined;
+
```

```
-     // Raw items waiting to be applied.
-     const queueRef = useRef<AiStreamedChunk[]>([]);
-
+     // Raw items waiting to be applied.
+     const queueRef = useRef<QueuedStreamChunk[]>([]);
+
```

```
-
-                 // Snapshot the current queue content at iteration start.
-                 // When throttled, cap the snapshot to the initial drain budget.
-                 const snapshot: readonly AiStreamedChunk[] = throttleEnabled
-                     ? queueRef.current.slice(0, remainingFromStart)
-                     : queueRef.current.slice(0);
-
+
+                 const head = queueRef.current[0];
+                 if (!head) break;
+
+                 // Restrict coalescence to one exact accepted attempt even when a retry
reuses the same turnId.
+                 const maximumSnapshotLength = throttleEnabled
+                     ? Math.min(queueRef.current.length, remainingFromStart)
+                     : queueRef.current.length;
+                 let sameAttemptLength = 0;
+                 while (sameAttemptLength < maximumSnapshotLength) {
+                     const candidate = queueRef.current[sameAttemptLength];
+                     if (
+                         !candidate ||
+                         !isSameAttempt(
+                             candidate.attemptIdentity,
+                             head.attemptIdentity,
+                         )
+                     ) {
+                         break;
+                     }
```

```
+                     sameAttemptLength += 1;
+                 }
+                 const snapshot = queueRef.current
+                     .slice(0, sameAttemptLength)
+                     .map((entry) => entry.chunk);
+
```

```
-                     queueRef.current.splice(0, coalesced.count);
-                     // Apply merged item through the provided runner.
-                     await opts.runMerged(coalesced.merged);
-
+                     queueRef.current.splice(0, coalesced.count);
+                     // Apply the merged item with the identity shared by every consumed
entry.
+                     await opts.runMerged(
+                         coalesced.merged,
+                         head.attemptIdentity,
+                     );
+
```

```
-                     // Fallback: apply a single item as-is.
-                     const item = queueRef.current.shift() as AiStreamedChunk;
-                     await opts.runMerged(item);
-
+                     // Fallback: apply a single item as-is.
+                     const item = queueRef.current.shift();
+                     if (!item) break;
+                     await opts.runMerged(
+                         item.chunk,
+                         item.attemptIdentity,
+                     );
+
```

```
-
-     /** Enqueue a raw streamed chunk (fire-and-forget). */
-     const enqueueChunk = useCallback(
-         (chunk: AiStreamedChunk): void => {
-             queueRef.current.push(chunk);
-
+
+     /** Enqueue a raw streamed chunk with an already captured exact attempt identity.
*/
+     const enqueueChunkForAttempt = useCallback(
+         (
+             chunk: AiStreamedChunk,
+             attemptIdentity: AiStreamAttemptIdentity,
+         ): void => {
+             queueRef.current.push({ chunk, attemptIdentity });
+
```

```
-
-     /** Resolve when all queued items are fully applied and no drain is in progress. */
-
+
+     /** Enqueue from callers that intentionally bind Queue A to the currently active
attempt. */
+     const enqueueChunk = useCallback(
+         (chunk: AiStreamedChunk): void => {
+             const attemptIdentity = opts.getAttemptIdentity();
+             if (!attemptIdentity) {
+                 throw new Error(
+                     'Queue A cannot enqueue a chunk without an active stream attempt
identity.',
+                 );
+             }
```

```
+             enqueueChunkForAttempt(chunk, attemptIdentity);
+         },
+         [enqueueChunkForAttempt, opts],
+     );
+
+     /** Resolve when all queued items are fully applied and no drain is in progress. */
+
```

```
-             enqueueChunk,
-
+             enqueueChunk,
+             enqueueChunkForAttempt,
+
```

```
-         }),
-         [enqueueChunk, whenIdle],
-
+         }),
+         [enqueueChunk, enqueueChunkForAttempt, whenIdle],
+
```

AICode/src/frontend/src/messageBus/useAiStreamQueue.hook.ts
Copy
Review
Accept

🔧 ✅ *reading project*: *usePromptStreamDispatchGateQueue.hook.ts* [L1-L120], *usePromptStreamDispatchGateQueue.hook.ts* [L120-L260], *usePromptStreamDispatchGateQueue.hook.ts* [L260-L430], *usePromptStreamDispatchGateQueue.hook.ts* [L430-L680], *usePromptStreamDispatchGateQueue.hook.ts* [L680-L860], *usePromptStreamDispatchGateQueue.hook.ts* [L850-L940]
🔧 ❌ *patching file*: *usePromptStreamDispatchGateQueue.hook.ts*
AICode/src/frontend/src/messageBus/usePromptStreamDispatchGateQueue.hook.ts
Error
(what is it?)

🔧 ✅ *reading project*: *usePromptStreamDispatchGateQueue.hook.ts* [L1-L230], *usePromptStreamDispatchGateQueue.hook.ts* [L380-L930]
🔧 ✅ *patching file*: *usePromptStreamDispatchGateQueue.hook.ts*
AICode/src/frontend/src/messageBus/usePromptStreamDispatchGateQueue.hook.ts
Copy
Review
Accept
PatchFull file

```
- } from './PromptStreamLockStore.js';
-
+ } from './PromptStreamLockStore.js';
+ import type { AiStreamAttemptIdentity } from './AiStreamStartArgs.type.js';
+
```

```
-     enqueueMerged: (merged: AiStreamedChunk) => Promise<void>;
-
+     enqueueMerged: (merged: AiStreamedChunk) => Promise<void>;
+
+     /** Enqueue with the exact Queue A identity instead of reading current state at dispatch time. */
+     enqueueMergedForAttempt: (
+         merged: AiStreamedChunk,
+         attemptIdentity: AiStreamAttemptIdentity,
```

```
+     ) => Promise<void>;
+
```

```
-     chunk: AiStreamedChunk;
-
+     chunk: AiStreamedChunk;
+     attemptIdentity: AiStreamAttemptIdentity;
+     generation: number;
+
```

```
-
- /** Narrowed meta.tool_calls chunk for safe access inside the gate. */
-
+
+ /** Deferred Queue B entry whose attempt identity must survive lock release and replay.
*/
+ type DeferredGateChunk = Pick<
+     GateQueueItem,
+     'chunk' | 'attemptIdentity' | 'generation'
+ >;
+
+ /** Narrowed meta.tool_calls chunk for safe access inside the gate. */
+
```

```
- export function usePromptStreamDispatchGateQueue(opts: {
-     dispatchMerged: (merged: AiStreamedChunk) => Promise<void> | void;
-
+ export function usePromptStreamDispatchGateQueue(opts: {
+     dispatchMerged: (
+         merged: AiStreamedChunk,
+         attemptIdentity: AiStreamAttemptIdentity,
+     ) => Promise<void> | void;
+     getAttemptIdentity: () => AiStreamAttemptIdentity | undefined;
+
```

```
-     const drainAgainRequestedRef = useRef<boolean>(false);
-
+     const drainAgainRequestedRef = useRef<boolean>(false);
+
+     // Every reset advances this generation so post-await continuations cannot mutate a
successor attempt.
+     const generationRef = useRef<number>(0);
+
```

```
-         turnId: string;
-         suffix: AiStreamedToolCall[];
-
+         turnId: string;
+         suffix: AiStreamedToolCall[];
+         attemptIdentity: AiStreamAttemptIdentity;
+         generation: number;
+
```

```
-      */
-     const deferredWhileLockedRef = useRef<AiStreamedChunk[]>([]);
-
+      */
+     const deferredWhileLockedRef = useRef<DeferredGateChunk[]>([]);
+
```

```
-      * - This must be called after each turn to avoid memory leaks.
-      * - It is safe only when the upstream stream queue is idle or at the start of a
turn.
-
+      * - This must be called after each turn to avoid memory leaks.
```

```
+      * - The generation bump keeps resets safe while an old dispatch continuation is
still awaiting.
+
```
```
-     const resetInternalState = useCallback((): void => {
-
+     const resetInternalState = useCallback((): void => {
+         generationRef.current += 1;
+         for (const item of queueRef.current) {
+             // Reset cancels old Queue B work while deliberately releasing Queue A
backpressure.
+             item.resolve();
+         }
+
```
```
-         syntheticCloseStateRef.current = null;
-
+         syntheticCloseStateRef.current = null;
+         for (const resolve of idleResolversRef.current) {
+             resolve();
+         }
+
```
```
-         lockedCallIdRef.current = null;
-         drainingRef.current = false;
-         drainAgainRequestedRef.current = false;
-         tailRef.current = Promise.resolve();
-
+         lockedCallIdRef.current = null;
+         if (!drainingRef.current) {
+             drainAgainRequestedRef.current = false;
+             tailRef.current = Promise.resolve();
+         }
+
```
```
-     const maybeEnqueueSynthesizedClosingToolCalls = useCallback(
-         (args: { turnId: string; reason: string }): void => {
-
+     const maybeEnqueueSynthesizedClosingToolCalls = useCallback(
+         (args: {
+             turnId: string;
+             reason: string;
+             attemptIdentity: AiStreamAttemptIdentity;
+             generation: number;
+         }): void => {
+
```
```
-                 chunk: syntheticChunk,
-
+                 chunk: syntheticChunk,
+                 attemptIdentity: args.attemptIdentity,
+                 generation: args.generation,
+
```
```
-
-             deferredWhileLockedRef.current.push(item.chunk);
-
+
+             deferredWhileLockedRef.current.push({
+                 chunk: item.chunk,
+                 attemptIdentity: item.attemptIdentity,
+                 generation: item.generation,
+             });
+
```
```
-                 turnId: item.chunk.turnId,
```

```
-                 reason: args.reason,
-
+                 turnId: item.chunk.turnId,
+                 reason: args.reason,
+                 attemptIdentity: item.attemptIdentity,
+                 generation: item.generation,
+
```

```
-     const gateToolCallsChunk = useCallback(
-         (chunk: ToolCallsMetaChunk): { gatedChunk?: ToolCallsMetaChunk } => {
-
+     const gateToolCallsChunk = useCallback(
+         (
+             chunk: ToolCallsMetaChunk,
+             context: Pick<GateQueueItem, 'attemptIdentity' | 'generation'>,
+         ): { gatedChunk?: ToolCallsMetaChunk } => {
+
```

```
-                 // Store the suffix for later replay (after unlock) but never mark it
as "seen" now.
-                 if (suffix.length > 0) {
-                     pendingToolCallsSuffixRef.current = {
-                         turnId: chunk.turnId,
-                         suffix,
-
+                 // Store the suffix for later replay (after unlock) but never mark it
as "seen" now.
+                 if (suffix.length > 0) {
+                     pendingToolCallsSuffixRef.current = {
+                         turnId: chunk.turnId,
+                         suffix,
+                         attemptIdentity: context.attemptIdentity,
+                         generation: context.generation,
+
```

```
-                         suffix,
-                     };
-
+                         suffix,
+                         attemptIdentity: context.attemptIdentity,
+                         generation: context.generation,
+                     };
+
```

```
-     const tryDispatchOne = useCallback(async (): Promise<boolean> => {
-
+     const tryDispatchOne = useCallback(async (): Promise<boolean> => {
+         const currentGeneration = generationRef.current;
+
+
```

```
-             pendingToolCallsSuffixRef.current = null;
-
+             pendingToolCallsSuffixRef.current = null;
+             if (pending.generation !== currentGeneration) return true;
+
```

```
-
-             const gated = gateToolCallsChunk(synthetic).gatedChunk;
-
+
+             const gated = gateToolCallsChunk(synthetic, pending).gatedChunk;
+
```

```
-
-             await opts.dispatchMerged(gated);
```

```
-             return true;
-
+
+             await opts.dispatchMerged(gated, pending.attemptIdentity);
+             return generationRef.current === pending.generation;
+
-------------------------------------------------------------------------------
-             if (!deferred) return false;
-
+             if (!deferred) return false;
+             if (deferred.generation !== currentGeneration) {
+                 deferredWhileLockedRef.current.shift();
+                 return true;
+             }
+
-------------------------------------------------------------------------------
-
-             await opts.dispatchMerged(deferred);
-             deferredWhileLockedRef.current.shift();
-
+
+             await opts.dispatchMerged(
+                 deferred.chunk,
+                 deferred.attemptIdentity,
+             );
+             if (generationRef.current !== deferred.generation) return false;
+             if (deferredWhileLockedRef.current[0] === deferred) {
+                 deferredWhileLockedRef.current.shift();
+             }
+
-------------------------------------------------------------------------------
-         if (!head) return false;
-
+         if (!head) return false;
+         if (head.generation !== currentGeneration) {
+             queueRef.current.shift();
+             head.resolve();
+             return true;
+         }
+
-------------------------------------------------------------------------------
-
-                 const gated = gateToolCallsChunk(head.chunk).gatedChunk;
-
+
+                 const gated = gateToolCallsChunk(head.chunk, head).gatedChunk;
+
-------------------------------------------------------------------------------
-
-                 await opts.dispatchMerged(gated);
-
+
+                 await opts.dispatchMerged(gated, head.attemptIdentity);
+
-------------------------------------------------------------------------------
-                 }
-                 await opts.dispatchMerged(head.chunk);
-
+                 }
+                 await opts.dispatchMerged(
+                     head.chunk,
+                     head.attemptIdentity,
+                 );
+
-------------------------------------------------------------------------------
-
```

```
-             // The head item is now fully processed; remove it and resolve its
backpressure promise.
-
+
+             if (generationRef.current !== head.generation) {
+                 head.resolve();
+                 return false;
+             }
+             // The head item is now fully processed; remove it and resolve its
backpressure promise.
+
```

```
-             // The head item is now fully processed; remove it and resolve its
backpressure promise.
-             queueRef.current.shift();
-
+             // The head item is now fully processed; remove it and resolve its
backpressure promise.
+             if (queueRef.current[0] === head) queueRef.current.shift();
+
```

```
-         } catch (err) {
-
+         } catch (err) {
+             if (generationRef.current !== head.generation) {
+                 head.resolve();
+                 return false;
+             }
+
```

```
-             // Fail fast but never leave a pending promise in the queue.
-             queueRef.current.shift();
-
+             // Fail fast but never leave a pending promise in the queue.
+             if (queueRef.current[0] === head) queueRef.current.shift();
+
```

```
-
-     /** Enqueue a merged chunk and return a promise that resolves when dispatched. */
-     const enqueueMerged = useCallback(
-         (merged: AiStreamedChunk): Promise<void> => {
-
+
+     /** Enqueue with one exact attempt identity captured by Queue A or a direct caller.
*/
+     const enqueueMergedForAttempt = useCallback(
+         (
+             merged: AiStreamedChunk,
+             attemptIdentity: AiStreamAttemptIdentity,
+         ): Promise<void> => {
+
```

```
-             return new Promise<void>((resolve, reject) => {
-                 queueRef.current.push({ chunk: merged, resolve, reject });
-
+             return new Promise<void>((resolve, reject) => {
+                 queueRef.current.push({
+                     chunk: merged,
+                     attemptIdentity,
+                     generation: generationRef.current,
+                     resolve,
+                     reject,
+                 });
+
```

```
-
```

```
-     /** Resolve when all queued items are fully dispatched and no drain is in progress.
*/
-
+
+     /** Enqueue from direct callers that intentionally bind Queue B to the current
accepted attempt. */
+     const enqueueMerged = useCallback(
+         (merged: AiStreamedChunk): Promise<void> => {
+             const attemptIdentity = opts.getAttemptIdentity();
+             if (!attemptIdentity) {
+                 return Promise.reject(
+                     new Error(
+                         'Queue B cannot enqueue a chunk without an active stream
attempt identity.',
+                     ),
+                 );
+             }
+             return enqueueMergedForAttempt(merged, attemptIdentity);
+         },
+         [enqueueMergedForAttempt, opts],
+     );
+
+     /** Resolve when all queued items are fully dispatched and no drain is in progress.
*/
+
```

```
-             enqueueMerged,
-
+             enqueueMerged,
+             enqueueMergedForAttempt,
+
```

```
-         }),
-         [enqueueMerged, whenIdle, resetInternalState],
-
+         }),
+         [
+             enqueueMerged,
+             enqueueMergedForAttempt,
+             whenIdle,
+             resetInternalState,
+         ],
+
```

AICode/src/frontend/src/messageBus/usePromptStreamDispatchGateQueue.hook.ts
Copy
Review
Accept

🔧 ❌ *patching file*: *useAiStream.hook.ts*
AICode/src/frontend/src/messageBus/useAiStream.hook.ts
Error
(what is it?)

🔧 ✅ *reading project*: *useAiStream.hook.ts* [L35-L120], *useAiStream.hook.ts* [L175-L530], *useAiStream.hook.ts* [L540-L800], *useAiStream.hook.ts* [L790-L840]
🔧 ✅ *patching file*: *useAiStream.hook.ts*
AICode/src/frontend/src/messageBus/useAiStream.hook.ts
Copy
Review
Accept

PatchFull file

```
- import type { AiStreamStartArgs } from './AiStreamStartArgs.type.js';
- import type {
-
+ import type { AiStreamStartArgs } from './AiStreamStartArgs.type.js';
+ import type {
+     AiStreamAttemptIdentity,
+
```

```
-     | Extract<AiStreamStartArgs, { kind: 'liveSessionObserver' }>;
-
+     | Extract<AiStreamStartArgs, { kind: 'liveSessionObserver' }>;
+
+ /** Exact accepted identity retained across callbacks, Queue A, Queue B, and React
commit barriers. */
+ type ActiveAiStreamIdentity = {
+     readonly runRef: LiveSessionRunRef;
+     readonly streamInstanceId: string;
+     readonly attemptIdentity: AiStreamAttemptIdentity;
+ };
+
+ /** Compare frontend-only attempt identity without trusting a logical turnId reused by
retries. */
+ function isSameStreamAttempt(
+     active: ActiveAiStreamIdentity | undefined,
+     attemptIdentity: AiStreamAttemptIdentity,
+ ): boolean {
+     return (
+         active?.attemptIdentity.sourceId === attemptIdentity.sourceId &&
+         active.attemptIdentity.streamInstanceId ===
+             attemptIdentity.streamInstanceId
+     );
+ }
+
```

```
-     const [transportStopAvailable, setTransportStopAvailable] = useState(false);
-     const activeIdentityRef = useRef<
-         { runRef: LiveSessionRunRef; streamInstanceId: string } | undefined
-     >(undefined);
-
+     const [transportStopAvailable, setTransportStopAvailable] = useState(false);
+     const activeIdentityRef = useRef<ActiveAiStreamIdentity | undefined>(
+         undefined,
+     );
+
```

```
-     const streamQueueRef = useRef<StreamSerializedQueueApi | null>(null);
-
+     const streamQueueRef = useRef<StreamSerializedQueueApi | null>(null);
+
+     /** Read the active identity for direct queue callers that intentionally bind to
current state. */
+     const getActiveAttemptIdentity = useCallback(
+         (): AiStreamAttemptIdentity | undefined =>
+             activeIdentityRef.current?.attemptIdentity,
+         [],
+     );
+
```

```
-     const onMergedChunk = useCallback(
-         async (merged: AiStreamedChunk): Promise<void> => {
-
+     const onMergedChunk = useCallback(
+         async (
+             merged: AiStreamedChunk,
```

```
+             attemptIdentity: AiStreamAttemptIdentity,
+         ): Promise<void> => {
+             // Queue output cannot mutate Prompt after its exact source/stream attempt
lost ownership.
+             if (!isSameStreamAttempt(activeIdentityRef.current, attemptIdentity))
+                 return;
+
+
```

```
-                         // Ack meta: the fallback snapshot is now on disk and safe to
clear the composer input.
-                         const identity = activeIdentityRef.current;
-                         if (!identity) return;
-
+                         // Ack meta: the fallback snapshot is now on disk and safe to
clear the composer input.
+
```

```
-                             turnId: merged.turnId,
-                             streamInstanceId: identity.streamInstanceId,
-
+                             turnId: merged.turnId,
+                             streamInstanceId:
+                                 attemptIdentity.streamInstanceId,
+
```

```
-                         await q.waitForCommit();
-                         setLastToolCalls(merged.value.value);
-
+                         await q.waitForCommit();
+                         if (
+                             !isSameStreamAttempt(
+                                 activeIdentityRef.current,
+                                 attemptIdentity,
+                             )
+                         )
+                             return;
+                         setLastToolCalls(merged.value.value);
+
```

```
-                         setLastToolCalls(merged.value.value);
-                         await q.waitForCommit();
-
+                         setLastToolCalls(merged.value.value);
+                         await q.waitForCommit();
+                         if (
+                             !isSameStreamAttempt(
+                                 activeIdentityRef.current,
+                                 attemptIdentity,
+                             )
+                         )
+                             return;
+
```

```
-                             }
-                             await q.waitForCommit();
-
+                             }
+                             await q.waitForCommit();
+                             if (
+                                 !isSameStreamAttempt(
+                                     activeIdentityRef.current,
+                                     attemptIdentity,
+                                 )
+                             )
```

```
+                                return;
+
```

```
-         dispatchMerged: onMergedChunk,
-
+         dispatchMerged: onMergedChunk,
+         getAttemptIdentity: getActiveAttemptIdentity,
+
```

```
-     const opts: {
-         runMerged: (merged: AiStreamedChunk) => Promise<void>;
-
+     const opts: {
+         runMerged: (
+             merged: AiStreamedChunk,
+             attemptIdentity: AiStreamAttemptIdentity,
+         ) => Promise<void>;
+         getAttemptIdentity: () => AiStreamAttemptIdentity | undefined;
+
```

```
-         () => ({
-             runMerged: async (merged: AiStreamedChunk) => {
-                 await dispatchGateQueue.enqueueMerged(merged);
-
+         () => ({
+             runMerged: async (
+                 merged: AiStreamedChunk,
+                 attemptIdentity: AiStreamAttemptIdentity,
+             ) => {
+                 await dispatchGateQueue.enqueueMergedForAttempt(
+                     merged,
+                     attemptIdentity,
+                 );
+
```

```
-                 );
-             },
-
+                 );
+             },
+             getAttemptIdentity: getActiveAttemptIdentity,
+
```

```
-         }),
-         [dispatchGateQueue],
-
+         }),
+         [dispatchGateQueue, getActiveAttemptIdentity],
+
```

```
-                         };
-
+                         };
+             const attemptIdentity: AiStreamAttemptIdentity = {
+                 sourceId: runRef.sourceId,
+                 streamInstanceId: acceptedStreamInstanceId,
+             };
+
```

```
-             activeIdentityRef.current = {
-                 runRef,
-                 streamInstanceId: acceptedStreamInstanceId,
-
+             activeIdentityRef.current = {
+                 runRef,
+                 streamInstanceId: acceptedStreamInstanceId,
```

```
+                 attemptIdentity,
+
```

```
-                     if (
-                         activeIdentityRef.current?.streamInstanceId !==
-                             identityAtEntry.streamInstanceId ||
-                         activeIdentityRef.current.runRef.sourceId !==
-                             identityAtEntry.runRef.sourceId
-
+                     if (
+                         !isSameStreamAttempt(
+                             activeIdentityRef.current,
+                             identityAtEntry.attemptIdentity,
+                         )
+
```

```
-                     // JIT merging happens at pull; sanitization for visible tokens
occurs inside the runner.
-                     streamQueue.enqueueChunk(aiStreamedChunk);
-
+                     // JIT merging happens at pull; sanitization for visible tokens
occurs inside the runner.
+                     streamQueue.enqueueChunkForAttempt(
+                         aiStreamedChunk,
+                         identityAtEntry.attemptIdentity,
+                     );
+
```

```
-                     console.error('[useAiStream.onData] handler failed:', err);
-                     setIsStreaming(false);
-
+                     console.error('[useAiStream.onData] handler failed:', err);
+                     if (
+                         isSameStreamAttempt(
+                             activeIdentityRef.current,
+                             identityAtEntry.attemptIdentity,
+                         )
+                     ) {
+                         setIsStreaming(false);
+                     }
+
```

```
-                     } finally {
-                         // Reset the lock store and the gate queue at the end of the
turn even if a
-                         // queued UI transformation failed while the transport itself
completed.
-                         PromptStreamLockStore.reset();
-                         dispatchGateQueue.resetInternalState();
-
+                     } finally {
+
```

```
-                         if (
-                             activeIdentityRef.current?.streamInstanceId ===
-                             identityAtEntry.streamInstanceId
-                         ) {
-                             setOutcome('completed');
-                             setTransportStopAvailable(false);
-                             setIsStreaming(false);
-                             activeIdentityRef.current = undefined;
-                         }
-
+                         if (
+                             !isSameStreamAttempt(
```

```
+                                activeIdentityRef.current,
+                                identityAtEntry.attemptIdentity,
+                            )
+                        )
+                            return;
+
+                        // Only the still-owning attempt may reset global lock/gate
state and publish completion.
+                        PromptStreamLockStore.reset();
+                        dispatchGateQueue.resetInternalState();
+                        setOutcome('completed');
+                        setTransportStopAvailable(false);
+                        setIsStreaming(false);
+                        activeIdentityRef.current = undefined;
+
-------------------------------------------------------------------------------
-                // Publish a meta:error via the queue so the runner can append the
canonical suffix exactly once.
-                const currentTurnId = currentTurnIdRef.current;
-                if (!currentTurnId) {
-                    // A transport error without an active run cannot be correlated
safely to Prompt state.
-                    PromptStreamLockStore.reset();
-                    dispatchGateQueue.resetInternalState();
-                    setIsStreaming(false);
-                    return;
-                }
-
+                // Publish a meta:error via the queue so the runner can append the
canonical suffix exactly once.
+                const currentTurnId = identityAtEntry.runRef.turnId;
+
-------------------------------------------------------------------------------
-                        await dispatchGateQueue.whenIdle();
-                        if (!terminalDataSeenRef.current) {
-
+                        await dispatchGateQueue.whenIdle();
+                        if (
+                            !isSameStreamAttempt(
+                                activeIdentityRef.current,
+                                identityAtEntry.attemptIdentity,
+                            )
+                        )
+                            return;
+                        if (!terminalDataSeenRef.current) {
+
-------------------------------------------------------------------------------
-                        if (!terminalDataSeenRef.current) {
-                            streamQueue.enqueueChunk({
-                                turnId: currentTurnId,
-                                type: 'meta',
-                                value: {
-                                    metaType: 'error',
-                                    value: {
-                                        reason: AiStopErrorReason.SYSTEM_ERROR,
-                                        message: SYSTEM_ERROR_CANONICAL_TEXT,
-                                        details: errMsg,
-
+                        if (!terminalDataSeenRef.current) {
+                            streamQueue.enqueueChunkForAttempt(
+                                {
+                                    turnId: currentTurnId,
+                                    type: 'meta',
+                                    value: {
+                                        metaType: 'error',
```

```
+                                        value: {
+                                            reason: AiStopErrorReason.SYSTEM_ERROR,
+                                            message:
+                                                SYSTEM_ERROR_CANONICAL_TEXT,
+                                            details: errMsg,
+                                        },
+
```

```
-                                },
-                            });
-
+                                },
+                                identityAtEntry.attemptIdentity,
+                            );
+
```

```
-                    } finally {
-                        // Reset the lock store and the gate queue even when error-meta
dispatch itself fails.
-                        PromptStreamLockStore.reset();
-                        dispatchGateQueue.resetInternalState();
-
+                    } finally {
+
```

```
-                        if (
-                            activeIdentityRef.current?.streamInstanceId ===
-                            identityAtEntry.streamInstanceId
-                        ) {
-                            setOutcome('transportError');
-                            setTransportStopAvailable(false);
-                            setIsStreaming(false);
-                            activeIdentityRef.current = undefined;
-                        }
-
+                        if (
+                            !isSameStreamAttempt(
+                                activeIdentityRef.current,
+                                identityAtEntry.attemptIdentity,
+                            )
+                        )
+                            return;
+
+                        // Stale errors cannot reset successor queues or append a
second terminal.
+                        PromptStreamLockStore.reset();
+                        dispatchGateQueue.resetInternalState();
+                        setOutcome('transportError');
+                        setTransportStopAvailable(false);
+                        setIsStreaming(false);
+                        activeIdentityRef.current = undefined;
+
```

```
-    }, [activeStream, threadId, transportStopAvailable]);
-
+    }, [activeStream, threadId, transportStopAvailable]);
+
+    /**
+     * Identity frozen into the listener callbacks mounted for this accepted stream
instance.
+     * An old listener therefore keeps identity A even after refs and state advance to
successor B.
+     */
+    const listenerAttemptIdentity = useMemo<
+        AiStreamAttemptIdentity | undefined
```

```
+     >(
+         () =>
+             activeRunRef && streamInstanceId
+                 ? {
+                       sourceId: activeRunRef.sourceId,
+                       streamInstanceId,
+                   }
+                 : undefined,
+         [activeRunRef, streamInstanceId],
+     );
+
```

```
-             onData: (aiStreamedChunk: AiStreamedChunk) => {
-
+             onData: (aiStreamedChunk: AiStreamedChunk) => {
+                 if (!listenerAttemptIdentity) return;
+
```

```
-                 if (!identityAtEntry) return;
-                 try {
-
+                 if (
+                     !identityAtEntry ||
+                     !isSameStreamAttempt(
+                         identityAtEntry,
+                         listenerAttemptIdentity,
+                     )
+                 )
+                     return;
+                 try {
+
```

```
-             onComplete: () => {
-
+             onComplete: () => {
+                 if (!listenerAttemptIdentity) return;
+
```

```
-                 if (!identityAtEntry) return;
-                 // Finalize only after the queue drained and one commit turn has
passed.
-
+                 if (
+                     !identityAtEntry ||
+                     !isSameStreamAttempt(
+                         identityAtEntry,
+                         listenerAttemptIdentity,
+                     )
+                 )
+                     return;
+                 // Finalize only after the queue drained and one commit turn has
passed.
+
```

```
-             onError: (err: unknown) => {
-
+             onError: (err: unknown) => {
+                 if (!listenerAttemptIdentity) return;
+
```

```
-                 const identityAtEntry = activeIdentityRef.current;
-                 if (!identityAtEntry) return;
-
+                 const identityAtEntry = activeIdentityRef.current;
+                 if (
+                     !identityAtEntry ||
```

```
+                     !isSameStreamAttempt(
+                         identityAtEntry,
+                         listenerAttemptIdentity,
+                     )
+                 )
+                     return;
+
```

```
-         }),
-         [dispatchGateQueue, onRawLiveReceived, streamQueue],
-
+         }),
+         [
+             dispatchGateQueue,
+             listenerAttemptIdentity,
+             onRawLiveReceived,
+             streamQueue,
+         ],
+
```

```
-      */
-     const dispatchGateQueue = usePromptStreamDispatchGateQueue({
-         dispatchMerged: onMergedChunk,
-         getAttemptIdentity: getActiveAttemptIdentity,
-     });
-
+      */
+     const dispatchGateQueueOptions = useMemo(
+         () => ({
+             dispatchMerged: onMergedChunk,
+             getAttemptIdentity: getActiveAttemptIdentity,
+         }),
+         [getActiveAttemptIdentity, onMergedChunk],
+     );
+     const dispatchGateQueue = usePromptStreamDispatchGateQueue(
+         dispatchGateQueueOptions,
+     );
+
```

AICode/src/frontend/src/messageBus/useAiStream.hook.ts
Copy
Review
Accept

🔧 ✅ *patching file*: *usePromptLiveSession.hook.ts*
AICode/src/frontend/src/app/prompt/liveSession/core/usePromptLiveSession.hook.ts
Copy
Review
Accept
PatchFull file

```
- import { useCallback, useEffect, useRef } from 'react';
- import type { Dispatch, MutableRefObject, SetStateAction } from 'react';
-
+ import { useCallback, useEffect, useMemo, useRef, useState } from 'react';
+ import type { Dispatch, MutableRefObject, SetStateAction } from 'react';
+
```

```
- export type PromptLiveSessionApi = {
-
+ export type PromptLiveSessionApi = {
+     markSourceStopRequested: () => boolean;
+
```

```
-     retryHandoff: () => void;
```

```
-
+     retryHandoff: () => void;
+     sourceStopAvailable: boolean;
+     sourceStopPending: boolean;
+     stopSource: () => Promise<void>;
+
```

```
- ): PromptLiveSessionApi {
-
+ ): PromptLiveSessionApi {
+     // Async continuations always read the latest React bindings without making Open
callback identity unstable.
+     const paramsRef = useRef(params);
+     paramsRef.current = params;
+
```

```
-     const openAttemptRef = useRef<string | undefined>(undefined);
-
+     const openAttemptRef = useRef<string | undefined>(undefined);
+     const openInFlightRef = useRef(false);
+     const mountedRef = useRef(false);
+
```

```
-     const openRef = useRef<
-         ((mode: 'hydrateFromSnapshot' | 'preserveLocal') => void) | undefined
-
+     const openRef = useRef<
+         ((mode: 'hydrateFromSnapshot' | 'preserveLocal') => boolean) | undefined
+
```

```
-     const handoffStartedRef = useRef(false);
-
+     const handoffStartedRef = useRef(false);
+     const sourceStopPendingRef = useRef(false);
+     const [sourceStopPending, setSourceStopPending] = useState(false);
+
```

```
-     const [sourceStopPending, setSourceStopPending] = useState(false);
-
+     const [sourceStopPending, setSourceStopPending] = useState(false);
+
+     /** Cancel the exact scheduled not-ready retry without affecting an in-flight Open
request. */
+     const clearRetryTimer = useCallback((): void => {
+         if (retryTimerRef.current === undefined) return;
+         window.clearTimeout(retryTimerRef.current);
+         retryTimerRef.current = undefined;
+     }, []);
+
+     /** Return whether one async Open continuation still owns this mounted hook
instance. */
+     const ownsOpenAttempt = useCallback((attemptId: string): boolean => {
+         return (
+             mountedRef.current && openAttemptRef.current === attemptId
+         );
+     }, []);
+
```

```
-         (response: unknown, mode: 'hydrateFromSnapshot' | 'preserveLocal') => {
-
+         (response: unknown, mode: 'hydrateFromSnapshot' | 'preserveLocal') => {
+             const current = paramsRef.current;
+
```

```
-             if (
-                 !params.bootstrap ||
```

```
-
+            if (
+                !current.bootstrap ||
+
--------------------------------------------------------------------------------
-                    envelope.threadId !==
-                        params.bootstrap.runRef.locator.sessionId ||
-                    envelope.turnId !== params.bootstrap.runRef.turnId ||
-                    envelope.sourceId !== params.bootstrap.runRef.sourceId
-
+                    envelope.threadId !==
+                        current.bootstrap.runRef.locator.sessionId ||
+                    envelope.turnId !== current.bootstrap.runRef.turnId ||
+                    envelope.sourceId !== current.bootstrap.runRef.sourceId
+
--------------------------------------------------------------------------------
-                            : 'classicHydration',
-                    replaceAllMessages: params.replaceAllMessages,
-                    setPromptValue: params.setPromptValue,
-                    setAttachments: params.setAttachments,
-                    setNextTurnId: params.setNextTurnId,
-                    setOpenThreadId: params.setOpenThreadId,
-                    setHydrateScrollSeq: params.setHydrateScrollSeq,
-                    hasLocalPromptOverrideRef: params.hasLocalPromptOverrideRef,
-                    resetPromptRunState: params.resetPromptRunState,
-
+                            : 'classicHydration',
+                    replaceAllMessages: current.replaceAllMessages,
+                    setPromptValue: current.setPromptValue,
+                    setAttachments: current.setAttachments,
+                    setNextTurnId: current.setNextTurnId,
+                    setOpenThreadId: current.setOpenThreadId,
+                    setHydrateScrollSeq: current.setHydrateScrollSeq,
+                    hasLocalPromptOverrideRef: current.hasLocalPromptOverrideRef,
+                    resetPromptRunState: current.resetPromptRunState,
+
--------------------------------------------------------------------------------
-                return parsed;
-            },
-            [params],
-
+                return parsed;
+            },
+            [],
+
--------------------------------------------------------------------------------
-            async (error: unknown, unavailable = false): Promise<void> => {
-                if (!params.bootstrap) return;
-
+            async (error: unknown, unavailable = false): Promise<void> => {
+                const current = paramsRef.current;
+                if (!current.bootstrap || !mountedRef.current) return;
+

--------------------------------------------------------------------------------
-                if (
-                    params.bootstrap.presentationAdapter.handoffPolicy ===
-
+                if (
+                    current.bootstrap.presentationAdapter.handoffPolicy ===
+

--------------------------------------------------------------------------------
-                ) {
-                    params.setHandoffFailureMessage(message);
-                    params.setPromptRuntimeMode('liveSessionReplayFailed');
-
+                ) {
```

```
+                 current.setHandoffFailureMessage(message);
+                 current.setPromptRuntimeMode('liveSessionReplayFailed');
+
```

```
-                 title: unavailable
-                     ? params.bootstrap.presentationAdapter
-
+                 title: unavailable
+                     ? current.bootstrap.presentationAdapter
+
```

```
-                           .unavailableDialogTitle
-                     : params.bootstrap.presentationAdapter.failureDialogTitle,
-
+                           .unavailableDialogTitle
+                     : current.bootstrap.presentationAdapter.failureDialogTitle,
+
```

```
-         },
-         [params],
-
+         },
+         [],
+
```

```
-     const open = useCallback(
-         (snapshotMode: 'hydrateFromSnapshot' | 'preserveLocal'): void => {
-             const bootstrap = params.bootstrap;
-             if (!bootstrap) return;
-
+     const open = useCallback(
+         (
+             snapshotMode: 'hydrateFromSnapshot' | 'preserveLocal',
+         ): boolean => {
+             const current = paramsRef.current;
+             const bootstrap = current.bootstrap;
+             if (!mountedRef.current || !bootstrap || openInFlightRef.current)
+                 return false;
+             clearRetryTimer();
+
```

```
-             openAttemptRef.current = attemptId;
-             params.setIsHydrating(true);
-
+             openAttemptRef.current = attemptId;
+             openInFlightRef.current = true;
+             current.setIsHydrating(true);
+
```

```
-                     );
-                     if (openAttemptRef.current !== attemptId) return;
-
+                     );
+                     if (!ownsOpenAttempt(attemptId)) return;
+
```

```
-                         retryTimerRef.current = window.setTimeout(
-                             () => openRef.current?.(snapshotMode),
-
+                         retryTimerRef.current = window.setTimeout(
+                             () => {
+                                 retryTimerRef.current = undefined;
+                                 openRef.current?.(snapshotMode);
+                             },
+
```

```
-                     if (parsed.kind === 'terminal') {
-                         params.setPromptRuntimeMode('normal');
-                         params.setHandoffFailureMessage(undefined);
-
+                     if (parsed.kind === 'terminal') {
+                         const latest = paramsRef.current;
+                         latest.setPromptRuntimeMode('normal');
+                         latest.setHandoffFailureMessage(undefined);
+
--------------------------------------------------------------------------------
-                     }
-                     const startResult = params.startLiveObserver({
-
+                     }
+                     const latest = paramsRef.current;
+                     const startResult = latest.startLiveObserver({
+

--------------------------------------------------------------------------------
-                         startResult.streamInstanceId;
-                     params.setPromptRuntimeMode('liveSessionStreaming');
-                     params.setHandoffFailureMessage(undefined);
-
+                         startResult.streamInstanceId;
+                     latest.setPromptRuntimeMode('liveSessionStreaming');
+                     latest.setHandoffFailureMessage(undefined);
+

--------------------------------------------------------------------------------
-                 } catch (error) {
-                     if (openAttemptRef.current === attemptId) await fail(error);
-
+                 } catch (error) {
+                     if (ownsOpenAttempt(attemptId)) await fail(error);
+

--------------------------------------------------------------------------------
-                 } finally {
-                     if (openAttemptRef.current === attemptId) {
-                         params.setIsHydrating(false);
-
+                 } finally {
+                     if (ownsOpenAttempt(attemptId)) {
+                         openInFlightRef.current = false;
+                         paramsRef.current.setIsHydrating(false);
+

--------------------------------------------------------------------------------
-             })();
-
+             })();
+             return true;
+

--------------------------------------------------------------------------------
-         },
-         [fail, hydrate, params],
-
+         },
+         [clearRetryTimer, fail, hydrate, ownsOpenAttempt],
+

--------------------------------------------------------------------------------
-     useEffect(() => {
-         if (!params.enabled || !params.bootstrap || bootstrappedRef.current)
-
+     useEffect(() => {
+         mountedRef.current = true;
+         return () => {
+             // StrictMode and real unmount both invalidate Open/timer continuations
before they can mutate state.
+             mountedRef.current = false;
```

```
+             bootstrappedRef.current = false;
+             openAttemptRef.current = undefined;
+             openInFlightRef.current = false;
+             clearRetryTimer();
+         };
+     }, [clearRetryTimer]);
+
+     useEffect(() => {
+         if (!params.enabled || !params.bootstrap || bootstrappedRef.current)
+
-----------------------------------------------------------------------------------------------------
-         open(params.bootstrap.presentationAdapter.snapshotPolicy);
-         return () => {
-             openAttemptRef.current = undefined;
-             if (retryTimerRef.current !== undefined) {
-                 window.clearTimeout(retryTimerRef.current);
-             }
-         };
-
+         open(params.bootstrap.presentationAdapter.snapshotPolicy);
+
-----------------------------------------------------------------------------------------------------
-                 'directAfterReplay' ||
-
+                 'directAfterReplay' ||
+             params.promptRuntimeMode !== 'liveSessionStreaming' ||
+
-----------------------------------------------------------------------------------------------------
-             !params.bootstrap ||
-             params.streamOutcome !== 'transportError'
-
+             !params.bootstrap ||
+             params.streamOutcome !== 'transportError' ||
+             (params.promptRuntimeMode !== 'liveSessionStreaming' &&
+                 params.promptRuntimeMode !== 'liveSessionHandoffPending')
+
-----------------------------------------------------------------------------------------------------
-     const retryHandoff = useCallback((): void => {
-         if (!params.bootstrap) return;
-
+     const retryHandoff = useCallback((): void => {
+         const current = paramsRef.current;
+         if (
+             !current.bootstrap ||
+             openInFlightRef.current ||
+             sourceStopPendingRef.current
+         )
+             return;
+
-----------------------------------------------------------------------------------------------------
-         handoffStartedRef.current = false;
-         open(params.bootstrap.presentationAdapter.retrySnapshotPolicy);
-     }, [open, params.bootstrap]);
-
+         handoffStartedRef.current = false;
+         observerStreamInstanceRef.current = undefined;
+         current.setPromptRuntimeMode('liveSessionOpening');
+         current.setHandoffFailureMessage(undefined);
+         open(current.bootstrap.presentationAdapter.retrySnapshotPolicy);
+     }, [open]);
+
+     /** Synchronously latch one exact normal-Prompt source Stop before either abort
transport begins. */
+     const markSourceStopRequested = useCallback((): boolean => {
+         const current = paramsRef.current;
```

```
+         const bootstrap = current.bootstrap;
+         if (
+             !current.enabled ||
+             !bootstrap ||
+             bootstrap.presentationAdapter.handoffPolicy !==
+                 'directAfterReplay' ||
+             current.promptRuntimeMode === 'normal' ||
+             sourceStopPendingRef.current
+         )
+             return false;
+
+         sourceStopPendingRef.current = true;
+         setSourceStopPending(true);
+         return true;
+     }, []);
+
+     /** Stop the exact normal-Prompt source even before a live observer transport has
been accepted. */
+     const stopSource = useCallback(async (): Promise<void> => {
+         const bootstrap = paramsRef.current.bootstrap;
+         if (!bootstrap || !markSourceStopRequested()) return;
+         try {
+             const result = await callLocalApi(
+                 '/api/ai/orch/liveSession/abortSource',
+                 {
+                     sessionId: bootstrap.runRef.locator.sessionId,
+                     storageScopeRelPath:
+                         bootstrap.runRef.locator.storageScopeRelPath,
+                     turnId: bootstrap.runRef.turnId,
+                     sourceId: bootstrap.runRef.sourceId,
+                 },
+             );
+             if (!result.accepted && mountedRef.current) {
+                 sourceStopPendingRef.current = false;
+                 setSourceStopPending(false);
+             }
+         } catch (error) {
+             sourceStopPendingRef.current = false;
+             if (mountedRef.current) setSourceStopPending(false);
+             console.error('[usePromptLiveSession] source Stop failed:', error);
+         }
+     }, [markSourceStopRequested]);
+
+     /** Keep the unique composer Stop visible for every non-normal normalPrompt replay
state. */
+     const sourceStopAvailable =
+         params.enabled &&
+         params.bootstrap?.presentationAdapter.handoffPolicy ===
+             'directAfterReplay' &&
+         params.promptRuntimeMode !== 'normal';
+
+     useEffect(() => {
+         if (params.promptRuntimeMode !== 'normal') return;
+         sourceStopPendingRef.current = false;
+         setSourceStopPending(false);
+     }, [params.promptRuntimeMode]);
+
-
-     return { retryHandoff };
-
+
+     return useMemo(
+         () => ({
```

```
+             markSourceStopRequested,
+             retryHandoff,
+             sourceStopAvailable,
+             sourceStopPending,
+             stopSource,
+         }),
+         [
+             markSourceStopRequested,
+             retryHandoff,
+             sourceStopAvailable,
+             sourceStopPending,
+             stopSource,
+         ],
+     );
+
```

AICode/src/frontend/src/app/prompt/liveSession/core/usePromptLiveSession.hook.ts

Copy

Review

Accept

reading project: Prompt.tsx [L1235-L1325], Prompt.tsx [L390-L470], Prompt.tsx [L650-L730], Prompt.tsx [L1195-L1230]

patching file: Prompt.tsx

AICode/src/frontend/src/app/prompt/Prompt/Prompt.tsx

Copy

Review

Accept

PatchFull file

```
-     /**
-      * Cancel controller (Stop button + global Escape + confirm policy).
-      *
-      * IMPORTANT invariants:
-      * - Escape binding must be active while a run is pending/streaming OR while an index-gate notice is visible,
-      *   so Escape can cancel a gated/buffered action.
-      * - The confirmation timer must NOT start while the index-gate notice is visible.
-      * - When cancel is executed, we close the index-gate notice (closeNotice) for both Stop and Escape.
-      */
-     const cancelControllerParams = useMemo(
-         () => ({
-             runningForEscape:
-                 uiPending ||
-                 isStreaming ||
-                 transportStopAvailable ||
-                 isIndexPending,
-             runningForConfirm:
-                 (uiPending || isStreaming || transportStopAvailable) &&
-                 !isIndexPending,
-             abortAckPending,
-             handleStopCore: handleStopFromComposer,
-             onAfterCancelCleanup: closeNotice,
-         }),
-         [
-             abortAckPending,
-             closeNotice,
-             handleStopFromComposer,
-             isIndexPending,
-             isStreaming,
-             transportStopAvailable,
-             uiPending,
```

```
-         ],
-     );
-     const { handleStopFromComposerRequested } = usePromptCancelController(
-         cancelControllerParams,
-     );
-
-     /**
-
+     /**
+
```

```
-     const finalIsLocked = isPromptLocked || isIndexPending;
-
+     const finalIsLocked = isPromptLocked || isIndexPending;
+
+     // Live replay may allow draft editing after baseline installation, but every Send
hotkey remains locked.
+     const runSubmissionLocked =
+         finalIsLocked || promptRuntimeMode !== 'normal';
+
```

```
-         onCloseNotice: closeNotice,
-         isLocked: finalIsLocked,
-
+         onCloseNotice: closeNotice,
+         isLocked: runSubmissionLocked,
+
```

```
-     );
-     const { retryHandoff } = usePromptLiveSession(promptLiveSessionParams);
-
+     );
+     const {
+         markSourceStopRequested,
+         retryHandoff,
+         sourceStopAvailable,
+         sourceStopPending,
+         stopSource,
+     } = usePromptLiveSession(promptLiveSessionParams);
+
+     // The normal-Prompt source is stoppable before Open becomes replay-ready or the
observer transport starts.
+     const effectiveTransportStopAvailable =
+         transportStopAvailable || sourceStopAvailable;
+     const effectiveAbortAckPending = abortAckPending || sourceStopPending;
+
+     /** Route the unique composer Stop to the accepted transport or directly to the
exact source reservation. */
+     const handleEffectiveStopCore = useCallback(async (): Promise<void> => {
+         if (transportStopAvailable) {
+             if (
+                 sourceStopAvailable &&
+                 !markSourceStopRequested()
+             ) {
+                 return;
+             }
+             await handleStopFromComposer();
+             return;
+         }
+         if (!sourceStopAvailable) {
+             await handleStopFromComposer();
+             return;
+         }
+         await stopSource();
+     }, [
```

```
+         handleStopFromComposer,
+         markSourceStopRequested,
+         sourceStopAvailable,
+         stopSource,
+         transportStopAvailable,
+     ]);
+
+     /**
+      * Cancel controller (Stop button + global Escape + confirm policy).
+      *
+      * IMPORTANT invariants:
+      * - Escape remains active for pre-observer normalPrompt sources and index-gated
actions.
+      * - The confirmation timer starts only for an actual source/transport execution.
+      * - Stop is disabled synchronously for both transport and direct-source abort
handshakes.
+      */
+     const cancelControllerParams = useMemo(
+         () => ({
+             runningForEscape:
+                 uiPending ||
+                 isStreaming ||
+                 effectiveTransportStopAvailable ||
+                 isIndexPending,
+             runningForConfirm:
+                 (uiPending ||
+                     isStreaming ||
+                     effectiveTransportStopAvailable) &&
+                 !isIndexPending,
+             abortAckPending: effectiveAbortAckPending,
+             handleStopCore: handleEffectiveStopCore,
+             onAfterCancelCleanup: closeNotice,
+         }),
+         [
+             closeNotice,
+             effectiveAbortAckPending,
+             effectiveTransportStopAvailable,
+             handleEffectiveStopCore,
+             isIndexPending,
+             isStreaming,
+             uiPending,
+         ],
+     );
+     const { handleStopFromComposerRequested } = usePromptCancelController(
+         cancelControllerParams,
+     );
+
+
--------------------------------------------------------------------------------
-             failureDetails: liveSessionHandoffFailureMessage,
-             transportStopAvailable,
-             stopEnabled: !abortAckPending,
-
+             failureDetails: liveSessionHandoffFailureMessage,
+             transportStopAvailable: effectiveTransportStopAvailable,
+             stopEnabled: !effectiveAbortAckPending,
+
--------------------------------------------------------------------------------
-     }, [
-         abortAckPending,
-
+     }, [
+         effectiveAbortAckPending,
+         effectiveTransportStopAvailable,
```

```
+
```

```
-          promptRuntimeMode,
-          transportStopAvailable,
-
+          promptRuntimeMode,
+
```

```
-                  /* Controlled run-state for the Send/Stop button:
-                     - Show Stop while streaming
-
+                  /* Controlled run-state for the Send/Stop button:
+                     - Show Stop while streaming or while an exact pre-observer live
source remains stoppable
+
```

```
-                      isRunningForInput ||
-                      transportStopAvailable ||
-
+                      isRunningForInput ||
+                      effectiveTransportStopAvailable ||
+
```

```
-                  /* Disable the Stop button strictly during abort handshake */
-                  stopEnabled={!abortAckPending}
-
+                  /* Disable the Stop button strictly during abort handshake */
+                  stopEnabled={!effectiveAbortAckPending}
+
```

```
-          isRunningForInput,
-          transportStopAvailable,
-
+          isRunningForInput,
+          effectiveTransportStopAvailable,
+
```

```
-          isIndexPending,
-          abortAckPending,
-
+          isIndexPending,
+          effectiveAbortAckPending,
+
```

AICode/src/frontend/src/app/prompt/Prompt/Prompt.tsx
Copy
Review
Accept

🔧 ✅ *reading project*: 🔎 e *"Queue A sans identité"*, 🔎 e *"Queue B sans identité"*, 🔎 e *"Fixtures réservation restantes"*, 🔎 e *"Ancien queue exact manquant"*, 🔎 e *"Ancien Gate exact manquant"*, 🔎 e *"Queue A tests appels"*
🔧 ❌ *patching file*: *AiStreamMergedQueue.test.ts*
AICode/src/frontend/src/messageBus/__test__/AiStreamMergedQueue.test.ts
Error
(what is it?)

🔧 ✅ *reading project*: *AiStreamMergedQueue.test.ts* [L1-L70], *AiStreamMergedQueue.test.ts* [L250-L450], *AiStreamMergedQueue.test.ts* [L450-L570], *AiStreamMergedQueue.test.ts* [L190-L260]
🔧 ❌ *patching file*: *AiStreamMergedQueue.test.ts*
AICode/src/frontend/src/messageBus/__test__/AiStreamMergedQueue.test.ts
Error
(what is it?)

🔧 ❌ *patching file*: *AiStreamMergedQueue.test.ts*
AICode/src/frontend/src/messageBus/__test__/AiStreamMergedQueue.test.ts
Error
(what is it?)

🔧 ❌ *patching file*: *AiStreamMergedQueue.test.ts*
AICode/src/frontend/src/messageBus/__test__/AiStreamMergedQueue.test.ts
Error
(what is it?)

🔧 ✅ *patching file*: *AiStreamMergedQueue.test.ts*
AICode/src/frontend/src/messageBus/__test__/AiStreamMergedQueue.test.ts
Copy
Review
Accept
PatchFull file

```
- import { renderToStaticMarkup } from 'react-dom/server';
-
+ import { renderToStaticMarkup } from 'react-dom/server';
+ import type { AiStreamAttemptIdentity } from
'../../messageBus/AiStreamStartArgs.type.js';
+
```

```
- const MOCK_THROTTLE_MS: number = 200;
-
+ const MOCK_THROTTLE_MS: number = 200;
+
+ const ATTEMPT_A: AiStreamAttemptIdentity = {
+     sourceId: 'source-a',
+     streamInstanceId: 'stream-a',
+ };
+ const ATTEMPT_B: AiStreamAttemptIdentity = {
+     sourceId: 'source-a',
+     streamInstanceId: 'stream-b',
+ };
+ const ATTEMPT_C: AiStreamAttemptIdentity = {
+     sourceId: 'source-b',
+     streamInstanceId: 'stream-c',
+ };
+
```

```
- let useSerializedStreamQueue!: (opts: {
-     runMerged: (chunk: AiStreamedChunk) => void | Promise<void>;
-
+ let useSerializedStreamQueue!: (opts: {
+     runMerged: (
+         chunk: AiStreamedChunk,
+         attemptIdentity: AiStreamAttemptIdentity,
+     ) => void | Promise<void>;
+
```

```
-     const mod = await import('../../messageBus/useAiStreamQueue.hook.js');
-     useSerializedStreamQueue = mod.useSerializedStreamQueue;
-
+     const mod = await import('../../messageBus/useAiStreamQueue.hook.js');
+     useSerializedStreamQueue = (opts) =>
+         mod.useSerializedStreamQueue({
+             ...opts,
+             getAttemptIdentity: () => ATTEMPT_A,
```

```
+         });
+
```

```
-             expect(sequence).toContain('error');
-         });
-     });
-
+             expect(sequence).toContain('error');
+         });
+     });
+
+     it('never coalesces contiguous chunks across sourceId or streamInstanceId
boundaries', async () => {
+         const apiRef: { current: StreamSerializedQueueApi | null } = {
+             current: null,
+         };
+         const calls: Array<{
+             chunk: AiStreamedChunk;
+             attemptIdentity: AiStreamAttemptIdentity;
+         }> = [];
+
+         function Harness(): JSX.Element {
+             const runMerged = React.useCallback(
+                 async (
+                     chunk: AiStreamedChunk,
+                     attemptIdentity: AiStreamAttemptIdentity,
+                 ): Promise<void> => {
+                     calls.push({ chunk, attemptIdentity });
+                 },
+                 [],
+             );
+             const queueOptions = React.useMemo(
+                 () => ({ runMerged }),
+                 [runMerged],
+             );
+             const q = useSerializedStreamQueue(queueOptions);
+             apiRef.current = q;
+             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('A'), ATTEMPT_A);
+         api.enqueueChunkForAttempt(token('B'), ATTEMPT_B);
+         api.enqueueChunkForAttempt(token('C'), ATTEMPT_C);
+         await api.whenIdle();
+
+         expect(calls).toEqual([
+             { chunk: token('A'), attemptIdentity: ATTEMPT_A },
+             { chunk: token('B'), attemptIdentity: ATTEMPT_B },
+             { chunk: token('C'), attemptIdentity: ATTEMPT_C },
+         ]);
+     });
+
```

AICode/src/frontend/src/messageBus/__test__/AiStreamMergedQueue.test.ts
Copy
Review
Accept

patching file: *PromptStreamDispatchGateQueue.test.ts*

AICode/src/frontend/src/messageBus/__test__/PromptStreamDispatchGateQueue.test.ts
Copy
Review
Accept
PatchFull file

```
- import { describe, it, jest } from '@jest/globals';
- import React, { type JSX, useEffect, useMemo } from 'react';
-
+ import { describe, it, jest } from '@jest/globals';
+ import React, {
+     type JSX,
+     useCallback,
+     useEffect,
+     useMemo,
+ } from 'react';
+
```

```
- import { act } from 'react';
-
+ import { act } from 'react';
+ import type { AiStreamAttemptIdentity } from '../AiStreamStartArgs.type.js';
+
```

```
- } from '../usePromptStreamDispatchGateQueue.hook.js';
-
+ } from '../usePromptStreamDispatchGateQueue.hook.js';
+
+ const ATTEMPT_A: AiStreamAttemptIdentity = {
+     sourceId: 'source-a',
+     streamInstanceId: 'stream-a',
+ };
+ const ATTEMPT_B: AiStreamAttemptIdentity = {
+     sourceId: 'source-a',
+     streamInstanceId: 'stream-b',
+ };
+ const getAttemptA = (): AiStreamAttemptIdentity => ATTEMPT_A;
+
```

```
-             }
-         },
-
+             }
+         },
+         getAttemptIdentity: getAttemptA,
+
```

```
-     });
- });
-
+     });
+
+     it('preserves exact attempt identity and cannot let a stale post-reset dispatch
consume successor work', async () => {
+         PromptStreamLockStore.reset();
+         const container = document.createElement('div');
+         document.body.appendChild(container);
+         const root = createRoot(container);
+         const dispatches: Array<{
+             value: string;
+             attemptIdentity: AiStreamAttemptIdentity;
+         }> = [];
+         let gate: PromptStreamDispatchGateQueueApi | undefined;
+         let releaseFirstDispatch: (() => void) | undefined;
+         const firstDispatchBarrier = new Promise<void>((resolve) => {
```

```
+            releaseFirstDispatch = resolve;
+        });
+
+        function ResetHarness(): JSX.Element | null {
+            const dispatchMerged = useCallback(
+                async (
+                    chunk: AiStreamedChunk,
+                    attemptIdentity: AiStreamAttemptIdentity,
+                ): Promise<void> => {
+                    if (chunk.type !== 'token') return;
+                    dispatches.push({
+                        value: chunk.value,
+                        attemptIdentity,
+                    });
+                    if (
+                        attemptIdentity.streamInstanceId ===
+                        ATTEMPT_A.streamInstanceId
+                    ) {
+                        await firstDispatchBarrier;
+                    }
+                },
+                [],
+            );
+            const queueOptions = useMemo(
+                () => ({
+                    dispatchMerged,
+                    getAttemptIdentity: getAttemptA,
+                }),
+                [dispatchMerged],
+            );
+            const api = usePromptStreamDispatchGateQueue(queueOptions);
+            useEffect(() => {
+                gate = api;
+            }, [api]);
+            return null;
+        }
+
+        await act(async () => {
+            root.render(React.createElement(ResetHarness));
+        });
+        const queue = gate;
+        if (!queue) throw new Error('Gate queue API not initialized');
+
+        const first = queue.enqueueMergedForAttempt(
+            tokenChunk('A'),
+            ATTEMPT_A,
+        );
+        await act(async () => {
+            await Promise.resolve();
+        });
+
+        queue.resetInternalState();
+        const second = queue.enqueueMergedForAttempt(
+            tokenChunk('B'),
+            ATTEMPT_B,
+        );
+        const release = releaseFirstDispatch;
+        if (!release) throw new Error('First dispatch barrier is unavailable');
+        release();
+        await act(async () => {
+            await first;
+            await second;
+            await queue.whenIdle();
```

```
+         });
+
+         expect(dispatches).toEqual([
+             { value: 'A', attemptIdentity: ATTEMPT_A },
+             { value: 'B', attemptIdentity: ATTEMPT_B },
+         ]);
+
+         await act(async () => {
+             root.unmount();
+         });
+         document.body.removeChild(container);
+     });
+ });
+
```

AICode/src/frontend/src/messageBus/__test__/PromptStreamDispatchGateQueue.test.ts

Copy

Review

Accept

🔧 ✅ *patching file*: *useAiStream.hook.rawLiveSideband.test.ts*

AICode/src/frontend/src/messageBus/__test__/useAiStream.hook.rawLiveSideband.test.ts

Copy

Review

Accept

PatchFull file

```
- import { act } from 'react';
-
+ import { act } from 'react';
+ import type { AiStreamAttemptIdentity } from '../AiStreamStartArgs.type.js';
+
```

```
-
- const enqueueChunkMock = jest.fn<(chunk: AiStreamedChunk) => void>();
-
+
+ const enqueueChunkMock = jest.fn<
+     (
+         chunk: AiStreamedChunk,
+         attemptIdentity: AiStreamAttemptIdentity,
+     ) => void
+ >();
+
```

```
-     useSerializedStreamQueue: () => ({
-         enqueueChunk: enqueueChunkMock,
-
+     useSerializedStreamQueue: () => ({
+         enqueueChunk: (chunk: AiStreamedChunk) =>
+             enqueueChunkMock(chunk, {
+                 sourceId: 'default-source',
+                 streamInstanceId: 'default-stream',
+             }),
+         enqueueChunkForAttempt: enqueueChunkMock,
+
```

AICode/src/frontend/src/messageBus/__test__/useAiStream.hook.rawLiveSideband.test.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?)

reading project: *useAiStream.hook.promptSourceIdentity.test.ts*
patching file: *useAiStream.hook.promptSourceIdentity.test.ts*
AICode/src/frontend/src/messageBus/__test__/useAiStream.hook.promptSourceIdentity.test.ts
Copy
Review
Accept
PatchFull file

```
- import { describe, it } from '@jest/globals';
- import type {
-
+ /** @jest-environment jsdom */
+
+ import { afterEach, beforeEach, describe, it, jest } from '@jest/globals';
+ import React from 'react';
+ import { act } from 'react';
+ import { createRoot } from 'react-dom/client';
+
+ import type {
+
```

```
- import type {
-     AiStreamAcceptedStart,
-
+ import type {
+     AiStreamAttemptIdentity,
+
```

```
- } from '../AiStreamStartArgs.type.js';
-
+ } from '../AiStreamStartArgs.type.js';
+ import type { AiStreamedChunk } from
'../../sharedlib/types/aiTypes/AiStreamedChunk.type.js';
+
```

```
- import type { AiStreamedChunk } from
'../../sharedlib/types/aiTypes/AiStreamedChunk.type.js';
-
+ import type { AiStreamedChunk } from
'../../sharedlib/types/aiTypes/AiStreamedChunk.type.js';
+
+ type StreamListenerHandlers = {
+     onData: (chunk: AiStreamedChunk) => void;
+     onComplete: () => void;
+     onError: (error: unknown) => void;
+ };
+
+ type QueueRunMerged = (
+     chunk: AiStreamedChunk,
+     attemptIdentity: AiStreamAttemptIdentity,
+ ) => Promise<void> | void;
+
+ const queuedChunks: Array<{
+     chunk: AiStreamedChunk;
+     attemptIdentity: AiStreamAttemptIdentity;
+ }> = [];
+ const handlersByChannel = new Map<string, StreamListenerHandlers>();
+ let queueRunMerged: QueueRunMerged | undefined;
+
```

```
+ jest.unstable_mockModule('../useAiStreamQueue.hook.js', () => ({
+     useSerializedStreamQueue: (opts: { runMerged: QueueRunMerged }) => {
+         queueRunMerged = opts.runMerged;
+         return {
+             enqueueChunk: () => {
+                 throw new Error('The production hook must use exact enqueue.');
+             },
+             enqueueChunkForAttempt: (
+                 chunk: AiStreamedChunk,
+                 attemptIdentity: AiStreamAttemptIdentity,
+             ) => {
+                 queuedChunks.push({ chunk, attemptIdentity });
+             },
+             whenIdle: async () => {},
+             waitForCommit: async () => {},
+         };
+     },
+ }));
+
+ jest.unstable_mockModule(
+     '../usePromptStreamDispatchGateQueue.hook.js',
+     () => ({
+         usePromptStreamDispatchGateQueue: (opts: {
+             dispatchMerged: QueueRunMerged;
+         }) => ({
+             enqueueMerged: async () => {
+                 throw new Error('The production hook must use exact enqueue.');
+             },
+             enqueueMergedForAttempt: async (
+                 chunk: AiStreamedChunk,
+                 attemptIdentity: AiStreamAttemptIdentity,
+             ) => {
+                 await opts.dispatchMerged(chunk, attemptIdentity);
+             },
+             whenIdle: async () => {},
+             resetInternalState: () => {},
+         }),
+     }),
+ );
+
+ jest.unstable_mockModule('../useStreamListener.hook.js', () => ({
+     useStreamListener: (
+         channel: string,
+         args: unknown,
+         handlers: StreamListenerHandlers,
+     ) => {
+         if (args !== undefined) handlersByChannel.set(channel, handlers);
+     },
+ }));
+
+ jest.unstable_mockModule('../useAiStreamRawLiveLogs.hook.js', () => ({
+     useAiStreamRawLiveLogs: () => ({
+         rawLiveLogs: [],
+         onRawLiveReceived: () => {},
+         reset: () => {},
+     }),
+ }));
+
+ jest.unstable_mockModule(
+     '../useAiStreamSyntaxErrorTagOpenerSanitizer.hook.js',
+     () => ({
+         useAiStreamSyntaxErrorTagOpenerSanitizer: () => ({
+             sanitize: (value: string) => value,
```

```
+             beginLogicalLine: () => {},
+             reset: () => {},
+         }),
+     }),
+ );
+
+ jest.unstable_mockModule(
+     '../useAiStreamSyntaxErrorTagNoAttributesSanitizer.hook.js',
+     () => ({
+         useAiStreamSyntaxErrorTagNoAttributesSanitizer: () => ({
+             sanitize: (value: string) => value,
+             reset: () => {},
+         }),
+     }),
+ );
+
+ const { useAiStream } = await import('../useAiStream.hook.js');
+
+ type HarnessApi = ReturnType<typeof useAiStream>;
+ const harnessApiRef: { current: HarnessApi | null } = { current: null };
+
+ function Harness(): React.ReactElement {
+     const api = useAiStream('a'.repeat(32));
+     harnessApiRef.current = api;
+     return React.createElement('div');
+ }
+
-------------------------------------------------------------------------------
- describe('useAiStream prompt source identity contract', () => {
-     it('allocates source and stream identity only for accepted starts', () => {
-         const refused: AiStreamStartResult = { accepted: false };
-         const accepted: AiStreamAcceptedStart = {
-             accepted: true,
-             runRef: {
-                 locator: {
-                     sessionId: 'a'.repeat(32),
-                     storageScopeRelPath: '',
-
+ describe('useAiStream prompt source identity contract', () => {
+     let container: HTMLDivElement;
+     let root: ReturnType<typeof createRoot>;
+
+     beforeEach(() => {
+         queuedChunks.length = 0;
+         handlersByChannel.clear();
+         queueRunMerged = undefined;
+         harnessApiRef.current = null;
+         container = document.createElement('div');
+         document.body.appendChild(container);
+         root = createRoot(container);
+     });
+
+     afterEach(() => {
+         act(() => root.unmount());
+         container.remove();
+     });
+
+     it('rejects a same-stack second start and drops Queue A output from a superseded
attempt', async () => {
+         await act(async () => {
+             root.render(React.createElement(Harness));
+         });
+         const api = harnessApiRef.current;
+         if (!api) throw new Error('HarnessApi not initialized');
```

```
+
+         let currentStart: AiStreamStartResult | undefined;
+         let refusedStart: AiStreamStartResult | undefined;
+         act(() => {
+             currentStart = api.start({
+                 kind: 'promptQuery',
+                 turnId: 'turn-shared',
+                 sourceId: 'source-shared',
+                 storageScopeRelPath: '',
+                 aiQuery: { type: 'prompt', promptValue: 'A' },
+                 aiQueryOptions: {},
+             });
+             refusedStart = api.start({
+                 kind: 'promptQuery',
+                 turnId: 'turn-shared',
+                 sourceId: 'source-shared',
+                 storageScopeRelPath: '',
+                 aiQuery: { type: 'prompt', promptValue: 'B' },
+                 aiQueryOptions: {},
+             });
+         });
+         if (!currentStart || !refusedStart)
+             throw new Error('Start results were not captured.');
+         expect(currentStart.accepted).toBe(true);
+         expect(refusedStart).toEqual({ accepted: false });
+         if (!currentStart.accepted) throw new Error('Expected accepted start.');
+
+         const firstHandlers = handlersByChannel.get(
+             'proxylocalapi:aiStream',
+         );
+         if (!firstHandlers) throw new Error('Prompt stream handlers missing.');
+         act(() => {
+             firstHandlers.onData({
+                 turnId: 'turn-shared',
+                 type: 'token',
+                 value: 'stale',
+             });
+         });
+         expect(queuedChunks[0]?.attemptIdentity).toEqual({
+             sourceId: currentStart.runRef.sourceId,
+             streamInstanceId: currentStart.streamInstanceId,
+         });
+
+         act(() => {
+             api.reset();
+             currentStart = api.start({
+                 kind: 'promptQuery',
+                 turnId: 'turn-shared',
+                 sourceId: 'source-shared',
+                 storageScopeRelPath: '',
+                 aiQuery: { type: 'prompt', promptValue: 'B' },
+                 aiQueryOptions: {},
+             });
+         });
+         expect(currentStart.accepted).toBe(true);
+         if (!currentStart.accepted)
+             throw new Error('Expected the successor start to be accepted.');
+
+         await act(async () => {
+             firstHandlers.onComplete();
+             await Promise.resolve();
+             await Promise.resolve();
+         });
```

```
+          expect(harnessApiRef.current?.isStreaming).toBe(true);
+          expect(harnessApiRef.current?.activeRunRef?.sourceId).toBe(
+              currentStart.runRef.sourceId,
+          );
+          act(() => {
+              firstHandlers.onData({
+                  turnId: 'turn-shared',
+                  type: 'token',
+                  value: 'ghost-listener-frame',
+              });
+          });
+          expect(queuedChunks).toHaveLength(1);
+
+          const runner = queueRunMerged;
+          const stale = queuedChunks[0];
+          if (!runner || !stale) throw new Error('Queue harness incomplete.');
+
+          await act(async () => {
+              await runner(stale.chunk, stale.attemptIdentity);
+          });
+          expect(harnessApiRef.current?.tokens).toEqual([]);
+          await act(async () => {
+              await runner(
+                  {
+                      turnId: 'turn-shared',
+                      type: 'meta',
+                      value: { metaType: 'fallback_persisted', value: true },
+
-----------------------------------------------------------------------------------------------------------------------
-                  },
-                  turnId: 'turn',
-                  sourceId: 'source',
-              },
-              streamInstanceId: 'stream',
-          };
-          expect(Object.keys(refused)).toEqual(['accepted']);
-          expect(accepted.runRef.sourceId).toBe('source');
-          expect(accepted.streamInstanceId).toBe('stream');
-
+                  },
+                  stale.attemptIdentity,
+              );
+          });
+          expect(harnessApiRef.current?.fallbackPersistedAck).toBeUndefined();
+
+          const secondHandlers = handlersByChannel.get(
+              'proxylocalapi:aiStream',
+          );
+          if (!secondHandlers || !currentStart.accepted)
+              throw new Error('Second stream handlers missing.');
+          act(() => {
+              secondHandlers.onData({
+                  turnId: 'turn-shared',
+                  type: 'token',
+                  value: 'current',
+              });
+          });
+          const current = queuedChunks[1];
+          if (!current) throw new Error('Current queue entry missing.');
+          await act(async () => {
+              await runner(current.chunk, current.attemptIdentity);
+          });
+          expect(harnessApiRef.current?.tokens).toEqual(['current']);
+
```

AICode/src/frontend/src/messageBus/__test__/useAiStream.hook.promptSourceIdentity.test.ts
Copy
Review
Accept

🔧 ✅ *reading project*: *usePromptLiveSession.normalPrompt.test.tsx*
🔧 ✅ *patching file*: *usePromptLiveSession.normalPrompt.test.tsx*
AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.normalPrompt.test.tsx
Copy
Review
Accept
PatchFull file

```
-
- import { describe, it } from '@jest/globals';
-
+
+ import { afterEach, beforeEach, describe, it, jest } from '@jest/globals';
+ import { act, renderHook } from '@testing-library/react';
+ import { useMemo, useRef, useState } from 'react';
+
+ import type { AiStreamStartResult } from
'../../../../../messageBus/AiStreamStartArgs.type.js';
+ import type { PromptRuntimeMode } from '../PromptLiveSession.types.js';
+
+ const callLocalApiMock = jest.fn<
+     (path: string, args: unknown) => Promise<unknown>
+ >();
+
+ jest.unstable_mockModule(
+     '../../../../../messageBus/callLocalApi.util.js',
+     () => ({ callLocalApi: callLocalApiMock }),
+ );
+ jest.unstable_mockModule(
+     '../../../Prompt/applyHydratedPromptSessionState.util.js',
+     () => ({ applyHydratedPromptSessionState: () => {} }),
+ );
+ jest.unstable_mockModule('../../../../../messageBus/closeSelf.util.js', () => ({
+     closeSelf: async () => {},
+ }));
+ jest.unstable_mockModule(
+     '../../../../../vsc-design-system/popups/VscAlertDialog.js',
+     () => ({ showVscAlertDialog: async () => {} }),
+ );
+
+ const { bindNormalPromptLiveSession } =
+     await import('../../adapters/NormalPromptLiveSession.adapter.js');
+ const { usePromptLiveSession } = await import('../usePromptLiveSession.hook.js');
+
+ const locator = {
+     sessionId: 'a'.repeat(32),
+     storageScopeRelPath: '',
+ } as const;
+ const bootstrap = bindNormalPromptLiveSession(
+     {
+         openMode: 'liveSession',
+         liveSessionTurnId: 'turn',
+         liveSessionViewerLeaseId: 'lease',
+         liveSessionClosePolicy: 'continueOnPanelClose',
+         liveSessionSourceId: 'source',
+         liveSessionAdapter: 'normalPrompt',
+     },
```

```
+     locator,
+ );
+
+ const noop = (): void => {};
+ const refuseLiveObserver = (): AiStreamStartResult => ({ accepted: false });
+
```

```
-
- import { bindNormalPromptLiveSession } from
'../../adapters/NormalPromptLiveSession.adapter.js';
-
+
+ /** Build the real hook with local state so Open, Retry, and source Stop transitions
are observable. */
+ function useNormalPromptLiveSessionHarness() {
+     const [mode, setMode] = useState<PromptRuntimeMode>(
+         'liveSessionOpening',
+     );
+     const [, setFailure] = useState<string | undefined>(undefined);
+     const [, setHydrating] = useState(false);
+     const localOverrideRef = useRef(false);
+     const api = usePromptLiveSession({
+         enabled: true,
+         bootstrap,
+         openThreadId: undefined,
+         promptRuntimeMode: mode,
+         setPromptRuntimeMode: setMode,
+         setHandoffFailureMessage: setFailure,
+         replaceAllMessages: noop,
+         setPromptValue: noop,
+         setAttachments: noop,
+         setNextTurnId: noop,
+         setOpenThreadId: noop,
+         setHydrateScrollSeq: noop,
+         setIsHydrating: setHydrating,
+         hasLocalPromptOverrideRef: localOverrideRef,
+         resetPromptRunState: noop,
+         startLiveObserver: refuseLiveObserver,
+         isStreaming: false,
+         lastMetaChunk: undefined,
+         streamInstanceId: undefined,
+         finalizedStreamInstanceId: undefined,
+         streamOutcome: undefined,
+         terminalDataSeen: false,
+         transportStopAvailable: false,
+     });
+     return useMemo(() => ({ api, mode }), [api, mode]);
+ }
+
```

```
- describe('normal Prompt live-session policies', () => {
-     it('keeps composer interactive, Send locked, Stop unique, and Retry banner stop-
free', () => {
-         const ready = bindNormalPromptLiveSession(
-             {
-                 openMode: 'liveSession',
-                 liveSessionTurnId: 'turn',
-                 liveSessionViewerLeaseId: 'lease',
-                 liveSessionClosePolicy: 'continueOnPanelClose',
-                 liveSessionSourceId: 'source',
-                 liveSessionAdapter: 'normalPrompt',
-             },
-             { sessionId: 'a'.repeat(32), storageScopeRelPath: '' },
-
```

```
+ describe('normal Prompt live-session policies', () => {
+     beforeEach(() => {
+         jest.useFakeTimers();
+         callLocalApiMock.mockReset();
+     });
+
+     afterEach(() => {
+         jest.useRealTimers();
+     });
+
+     it('keeps one Open request in flight and cancels the scheduled not-ready retry on
unmount', async () => {
+         let resolveOpen: ((value: unknown) => void) | undefined;
+         callLocalApiMock.mockImplementation(
+             () =>
+                 new Promise((resolve) => {
+                     resolveOpen = resolve;
+                 }),
+         );
+         const { result, unmount } = renderHook(() =>
+             useNormalPromptLiveSessionHarness(),
+
──────────────────────────────────────────────────────────────────────────────────────────
-         );
-         const failed = ready.presentationAdapter.buildViewModel({
-             mode: 'liveSessionReplayFailed',
-             failureDetails: 'transport',
-             transportStopAvailable: false,
-             stopEnabled: false,
-
+         );
+
+         await act(async () => {
+             await Promise.resolve();
+             await Promise.resolve();
+         });
+         expect(callLocalApiMock).toHaveBeenCalledTimes(1);
+
+         act(() => result.current.api.retryHandoff());
+         expect(callLocalApiMock).toHaveBeenCalledTimes(1);
+
+         await act(async () => {
+             resolveOpen?.({ kind: 'not_ready' });
+             await Promise.resolve();
+             await Promise.resolve();
+         });
+
+         unmount();
+         await act(async () => {
+             jest.advanceTimersByTime(250);
+             await Promise.resolve();
+         });
+         expect(callLocalApiMock).toHaveBeenCalledTimes(1);
+     });
+
+     it('latches direct source Stop synchronously before the abort RPC resolves', async
() => {
+         let resolveOpen: ((value: unknown) => void) | undefined;
+         let resolveAbort: ((value: unknown) => void) | undefined;
+         callLocalApiMock.mockImplementation((path) => {
+             if (path.endsWith('/open')) {
+                 return new Promise((resolve) => {
+                     resolveOpen = resolve;
+                 });
```

```
+             }
+             return new Promise((resolve) => {
+                 resolveAbort = resolve;
+             });
+
```

```
-         });
-         expect(failed).toEqual(
-             expect.objectContaining({
-                 showComposer: true,
-                 composerInteractive: true,
-                 sendEnabled: false,
-                 banner: expect.objectContaining({
-                     showRetryAction: true,
-                     showStopAction: false,
-                 }),
-             }),
-
+         });
+         const { result, unmount } = renderHook(() =>
+             useNormalPromptLiveSessionHarness(),
+
```

```
-         );
-     });
-
+         );
+
+         expect(result.current.api.sourceStopAvailable).toBe(true);
+         let firstStop: Promise<void> | undefined;
+         await act(async () => {
+             firstStop = result.current.api.stopSource();
+             await result.current.api.stopSource();
+         });
+         expect(
+             callLocalApiMock.mock.calls.filter(([path]) =>
+                 path.endsWith('/abortSource'),
+             ),
+         ).toHaveLength(1);
+         expect(result.current.api.sourceStopPending).toBe(true);
+         expect(result.current.api.markSourceStopRequested()).toBe(false);
+         if (!firstStop) throw new Error('First Stop promise was not captured.');
+
+         await act(async () => {
+             resolveAbort?.({ accepted: false });
+             await firstStop;
+         });
+         expect(result.current.api.sourceStopPending).toBe(false);
+
+         unmount();
+         resolveOpen?.({ kind: 'not_found' });
+     });
+
```

AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.normalPrompt.test.tsx

Copy

Review

Accept

🔧 ✅ *reading project*: usePromptLiveSession.handoff.test.tsx

🔧 ✅ *patching file*: usePromptLiveSession.handoff.test.tsx

AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.handoff.test.tsx

Copy

Review
Accept
PatchFull file

```
-
- import { describe, it } from '@jest/globals';
-
+
+ import { beforeEach, describe, it, jest } from '@jest/globals';
+ import { act, renderHook, waitFor } from '@testing-library/react';
+ import { useMemo, useRef, useState } from 'react';
+
+ import type {
+     AiStreamOutcome,
+     AiStreamStartResult,
+ } from '../../../../../messageBus/AiStreamStartArgs.type.js';
+ import type { PromptRuntimeMode } from '../PromptLiveSession.types.js';
+
+ const callLocalApiMock = jest.fn<
+     (path: string, args: unknown) => Promise<unknown>
+ >();
+ const startLiveObserverMock = jest.fn<() => AiStreamStartResult>(() => ({
+     accepted: true,
+     runRef: {
+         locator: {
+             sessionId: 'a'.repeat(32),
+             storageScopeRelPath: '',
+         },
+         turnId: 'turn',
+         sourceId: 'source',
+     },
+     streamInstanceId: 'stream-observer',
+ }));
+
+ jest.unstable_mockModule(
+     '../../../../../messageBus/callLocalApi.util.js',
+     () => ({ callLocalApi: callLocalApiMock }),
+ );
+ jest.unstable_mockModule(
+     '../../../Prompt/applyHydratedPromptSessionState.util.js',
+     () => ({ applyHydratedPromptSessionState: () => {} }),
+ );
+ jest.unstable_mockModule('../../../../../messageBus/closeSelf.util.js', () => ({
+     closeSelf: async () => {},
+ }));
+ jest.unstable_mockModule(
+     '../../../../../vsc-design-system/popups/VscAlertDialog.js',
+     () => ({ showVscAlertDialog: async () => {} }),
+ );
+
+ const { bindNormalPromptLiveSession } =
+     await import('../../adapters/NormalPromptLiveSession.adapter.js');
+ const { usePromptLiveSession } = await import('../usePromptLiveSession.hook.js');
+
+ const locator = {
+     sessionId: 'a'.repeat(32),
+     storageScopeRelPath: '',
+ } as const;
+ const bootstrap = bindNormalPromptLiveSession(
+     {
+         openMode: 'liveSession',
+         liveSessionTurnId: 'turn',
+         liveSessionViewerLeaseId: 'lease',
+         liveSessionClosePolicy: 'continueOnPanelClose',
```

```
+         liveSessionSourceId: 'source',
+         liveSessionAdapter: 'normalPrompt',
+     },
+     locator,
+ );
+
+ const noop = (): void => {};
+
+ type HandoffSignals = {
+     terminalDataSeen: boolean;
+     streamOutcome: AiStreamOutcome | undefined;
+     finalizedStreamInstanceId: string | undefined;
+ };
+
-
- import type { PromptLiveSessionHandoffPolicy } from '../PromptLiveSession.types.js';
-
+
+ /** Mount the real handoff state machine while controlling only the
terminal/finalization signals. */
+ function useHandoffHarness(signals: HandoffSignals) {
+     const [mode, setMode] = useState<PromptRuntimeMode>(
+         'liveSessionOpening',
+     );
+     const [, setFailure] = useState<string | undefined>(undefined);
+     const [, setHydrating] = useState(false);
+     const localOverrideRef = useRef(false);
+     const api = usePromptLiveSession({
+         enabled: true,
+         bootstrap,
+         openThreadId: undefined,
+         promptRuntimeMode: mode,
+         setPromptRuntimeMode: setMode,
+         setHandoffFailureMessage: setFailure,
+         replaceAllMessages: noop,
+         setPromptValue: noop,
+         setAttachments: noop,
+         setNextTurnId: noop,
+         setOpenThreadId: noop,
+         setHydrateScrollSeq: noop,
+         setIsHydrating: setHydrating,
+         hasLocalPromptOverrideRef: localOverrideRef,
+         resetPromptRunState: noop,
+         startLiveObserver: startLiveObserverMock,
+         isStreaming: false,
+         lastMetaChunk: undefined,
+         streamInstanceId: 'stream-observer',
+         finalizedStreamInstanceId: signals.finalizedStreamInstanceId,
+         streamOutcome: signals.streamOutcome,
+         terminalDataSeen: signals.terminalDataSeen,
+         transportStopAvailable: false,
+     });
+     return useMemo(() => ({ api, mode }), [api, mode]);
+ }
+
- describe('usePromptLiveSession handoff policies', () => {
-     it('keeps direct normalPrompt handoff distinct from EasyAgents terminal snapshots',
() => {
-         const normal: PromptLiveSessionHandoffPolicy = 'directAfterReplay';
-         const easyAgents: PromptLiveSessionHandoffPolicy = 'terminalSnapshot';
-         expect(normal).toBe('directAfterReplay');
-         expect(easyAgents).toBe('terminalSnapshot');
```

```
-
+ describe('usePromptLiveSession handoff policies', () => {
+     beforeEach(() => {
+         jest.clearAllMocks();
+         callLocalApiMock.mockResolvedValue({
+             kind: 'live',
+             session: {
+                 meta: {
+                     formatVersionNumber: 1,
+                     sessionId: locator.sessionId,
+                     title: 'Chat',
+                     createdAt: 1,
+                     updatedAt: 2,
+                     firstSendState: 'postSend',
+                 },
+                 messages: [],
+             },
+             threadId: locator.sessionId,
+             turnId: 'turn',
+             sourceId: 'source',
+             replayMode: 'replaceTurnAssistants',
+         });
+     });
+
+     it('hands normalPrompt directly to normal mode only after exact frontend
finalization and without a second Open', async () => {
+         const { result, rerender } = renderHook(
+             (signals: HandoffSignals) => useHandoffHarness(signals),
+             {
+                 initialProps: {
+                     terminalDataSeen: false,
+                     streamOutcome: undefined,
+                     finalizedStreamInstanceId: undefined,
+                 },
+             },
+         );
+
+         await waitFor(() =>
+             expect(result.current.mode).toBe('liveSessionStreaming'),
+         );
+         expect(callLocalApiMock).toHaveBeenCalledTimes(1);
+
+         await act(async () => {
+             rerender({
+                 terminalDataSeen: true,
+                 streamOutcome: 'completed',
+                 finalizedStreamInstanceId: 'stream-observer',
+             });
+         });
+         await waitFor(() => expect(result.current.mode).toBe('normal'));
+         expect(callLocalApiMock).toHaveBeenCalledTimes(1);
+     });
+
+     it('keeps Retry in opening mode while stale transportError signals from the
previous observer remain rendered', async () => {
+         let resolveRetryOpen: ((value: unknown) => void) | undefined;
+         callLocalApiMock
+             .mockResolvedValueOnce({
+                 kind: 'live',
+                 session: {
+                     meta: {
+                         formatVersionNumber: 1,
+                         sessionId: locator.sessionId,
```

```
+                         title: 'Chat',
+                         createdAt: 1,
+                         updatedAt: 2,
+                         firstSendState: 'postSend',
+                     },
+                     messages: [],
+                 },
+                 threadId: locator.sessionId,
+                 turnId: 'turn',
+                 sourceId: 'source',
+                 replayMode: 'replaceTurnAssistants',
+             })
+             .mockImplementationOnce(
+                 () =>
+                     new Promise((resolve) => {
+                         resolveRetryOpen = resolve;
+                     }),
+             );
+         const { result, rerender, unmount } = renderHook(
+             (signals: HandoffSignals) => useHandoffHarness(signals),
+             {
+                 initialProps: {
+                     terminalDataSeen: false,
+                     streamOutcome: undefined,
+                     finalizedStreamInstanceId: undefined,
+                 },
+             },
+         );
+         await waitFor(() =>
+             expect(result.current.mode).toBe('liveSessionStreaming'),
+         );
+
+         await act(async () => {
+             rerender({
+                 terminalDataSeen: false,
+                 streamOutcome: 'transportError',
+                 finalizedStreamInstanceId: undefined,
+             });
+         });
+         await waitFor(() =>
+             expect(result.current.mode).toBe('liveSessionReplayFailed'),
+         );
+
+         act(() => result.current.api.retryHandoff());
+         await act(async () => {
+             await Promise.resolve();
+             await Promise.resolve();
+         });
+         expect(result.current.mode).toBe('liveSessionOpening');
+         expect(callLocalApiMock).toHaveBeenCalledTimes(2);
+
+         unmount();
+         resolveRetryOpen?.({ kind: 'not_found' });
+
```

AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.handoff.test.tsx

Copy

Review

Accept

🔧 ✅ *patching file*: *NormalPromptLiveSession.adapter.test.ts*

AICode/src/frontend/src/app/prompt/liveSession/adapters/__test__/NormalPromptLiveSession.adapter.test.ts

Copy

Review
Accept
PatchFull file

```
-          );
-          expect(
-              ready.presentationAdapter.buildViewModel({
-                  mode: 'liveSessionStreaming',
-                  transportStopAvailable: true,
-                  stopEnabled: true,
-              }),
-          ).toEqual(
-
+          );
+          const opening = ready.presentationAdapter.buildViewModel({
+              mode: 'liveSessionOpening',
+              transportStopAvailable: true,
+              stopEnabled: true,
+          });
+          const streaming = ready.presentationAdapter.buildViewModel({
+              mode: 'liveSessionStreaming',
+              transportStopAvailable: true,
+              stopEnabled: true,
+          });
+          const failedWhileStopping = ready.presentationAdapter.buildViewModel({
+              mode: 'liveSessionReplayFailed',
+              transportStopAvailable: true,
+              stopEnabled: false,
+          });
+
+          expect(opening).toEqual(
+              expect.objectContaining({ composerInteractive: false }),
+          );
+          expect(streaming).toEqual(
+
```

```
-              }),
-          );
-
+              }),
+          );
+          expect(failedWhileStopping.banner).toEqual(
+              expect.objectContaining({
+                  showRetryAction: true,
+                  retryEnabled: false,
+              }),
+          );
+
```

AICode/src/frontend/src/app/prompt/liveSession/adapters/__test__/NormalPromptLiveSession.adapter.test.ts
Copy
Review
Accept

🔧 ✅ *reading project*: *SidebarStore.rehydrate.manager.test.ts* [L35-L90], *delete.endpoint.liveSession.test.ts* [L95-L135], *LiveSessionRegistry.service.test.ts* [L70-L100]
🔧 ✅ *patching file*: *SidebarStore.rehydrate.manager.test.ts*
AICode/src/extension/src/extension/panels/sidebar/__test__/SidebarStore.rehydrate.manager.test.ts
Copy
Review
Accept
PatchFull file

```
-              storageScopeRelPath: 'chat',
```

```
-         };
-
+             storageScopeRelPath: 'chat',
+         };
+         const closedObserverLocator = {
+             sessionId: 'c'.repeat(32),
+             storageScopeRelPath: 'chat',
+         };
+
```

```
-                     sourceId: 'source-b',
-                 },
-
+                     sourceId: 'source-b',
+                 },
+                 ownership: 'source',
+
```

```
-                 sourceStartedAtMs: 2,
-                 abortRequested: true,
-             },
-
+                 sourceStartedAtMs: 2,
+                 abortRequested: true,
+             },
+             {
+                 kind: 'reservation',
+                 runRef: {
+                     locator: closedObserverLocator,
+                     turnId: 'turn-c',
+                     sourceId: 'source-c',
+                 },
+                 ownership: 'observer',
+                 sourceStartedAtMs: 3,
+                 abortRequested: false,
+             },
+
```

```
-             ]),
-         );
-
+             ]),
+         );
+         expect(
+             sidebarStore.sidebarContentState.activityState.queryState.items.some(
+                 (item) => item.sessionId === closedObserverLocator.sessionId,
+             ),
+         ).toBe(false);
+
```

```
-                 },
-                 sourceStartedAtMs: 10,
-
+                 },
+                 ownership: 'source',
+                 sourceStartedAtMs: 10,
+
```

```
-
-     it('removes a closed non-source projection when no live authority remains', async
() => {
-
+
+     it('removes a closed non-source projection when only observer reservation authority
remains', async () => {
```

```
+
```

```
-                 openedAt: 1,
-             },
-         ];
-
+                 openedAt: 1,
+             },
+         ];
+         listActiveSnapshotsMock.mockReturnValue([
+             {
+                 kind: 'reservation',
+                 runRef: {
+                     locator,
+                     turnId: 'turn-d',
+                     sourceId: 'source-d',
+                 },
+                 ownership: 'observer',
+                 sourceStartedAtMs: 1,
+                 abortRequested: false,
+             },
+         ]);
+
```

AICode/src/extension/src/extension/panels/sidebar/__test__/SidebarStore.rehydrate.manager.test.ts
Copy
Review
Accept

patching file: *delete.endpoint.liveSession.test.ts*
AICode/src/extension/src/slimbk-local/endpoints/ai/orch/chat/__test__/delete.endpoint.liveSession.test.ts
Copy
Review
Accept
PatchFull file

```
-                 viewerGroupId: 'normalPrompt',
-
+                 viewerGroupId: 'normalPrompt',
+                 ownership: 'source',
+
```

AICode/src/extension/src/slimbk-local/endpoints/ai/orch/chat/__test__/delete.endpoint.liveSession.test.ts
Copy
Review
Accept

patching file: *LiveSessionRegistry.service.test.ts*
AICode/src/extension/src/aiTools/orch/liveSession/registry/__test__/LiveSessionRegistry.service.test.ts
Copy
Review
Accept
PatchFull file

```
-                 kind: 'reservation',
-                 runRef: RUN_A,
-
+                 kind: 'reservation',
+                 runRef: RUN_A,
+                 ownership: 'source',
+
```

```
-
```

```
-     it('emits an immutable observer-inactive wake-up only after the last exact lease is removed', () => {
-
+
+     it('emits immutable observer-inactive wake-ups after lease removal and later source settlement', () => {
+
```

```
-         expect(Object.isFrozen(events[0])).toBe(true);
-
+         expect(Object.isFrozen(events[0])).toBe(true);
+
+         // The handoff wake-up lets a panel coordinator retry cleanup if the first lease-release event ran too early.
+         registry.appendChunk({ runRef: RUN_A, chunk: terminal(RUN_A) });
+         expect(registry.markSourceSettled(RUN_A)).toBe(true);
+         expect(events).toHaveLength(1);
+         registry.markHandoffReady(RUN_A);
+         expect(events).toHaveLength(2);
+         expect(events[1]).toEqual(
+             expect.objectContaining({ runRef: RUN_A }),
+         );
+
```

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

Copy

Review

Accept

patching file: *PromptLiveSessionPanelCoordinator.service.test.ts*

AICode/src/extension/src/extension/panels/prompt/liveSession/__test__/PromptLiveSessionPanelCoordinator.service.test.ts

Copy

Review

Accept

PatchFull file

```
-                         viewerGroupId: 'group',
-
+                         viewerGroupId: 'group',
+                         ownership: 'source',
+
```

```
-         );
-     });
-
+         );
+     });
+
+     it('ignores retained handoff records and injects the current active source', async () => {
+         const open = jest.fn();
+         const activeRun = {
+             locator: LOCATOR,
+             turnId: 'turn',
+             sourceId: 'source-current',
+         } as const;
+         const coordinator = new PromptLiveSessionPanelCoordinatorService({
+             liveSession: {
+                 listActiveSnapshots: () => [
+                     {
+                         kind: 'record',
+                         runRef: RUN,
+                         viewerGroupId: 'group-old',
```

```
+                         ownership: 'source',
+                         replayMode: 'replaceTurnAssistants',
+                         sourceStartedAtMs: 1,
+                         baselineReady: true,
+                         sourceSettled: true,
+                         handoffReady: true,
+                         hasTerminalMeta: true,
+                         abortRequested: false,
+                         locatorEpoch: 1,
+                     },
+                     {
+                         kind: 'record',
+                         runRef: activeRun,
+                         viewerGroupId: 'group-current',
+                         ownership: 'source',
+                         replayMode: 'replaceTurnAssistants',
+                         sourceStartedAtMs: 2,
+                         baselineReady: true,
+                         sourceSettled: false,
+                         handoffReady: false,
+                         hasTerminalMeta: false,
+                         abortRequested: false,
+                         locatorEpoch: 2,
+                     },
+                 ],
+                 releaseViewerLease: () => undefined,
+                 isLocatorDestructionClaimed: () => false,
+                 onObserverInactive: () => () => {},
+                 onDestructionClaimReleased: () => () => {},
+             },
+             releaseEasyAgentsViewer: async () => {},
+         });
+
+         await coordinator.openPromptPanel({
+             locator: LOCATOR,
+             query: [{ name: 'sessionId', value: LOCATOR.sessionId }],
+             port: { open },
+         });
+
+         expect(open).toHaveBeenCalledWith(
+             expect.arrayContaining([
+                 {
+                     name: 'liveSessionSourceId',
+                     value: activeRun.sourceId,
+                 },
+             ]),
+         );
+     });
+
+     it('does not reinterpret an EasyAgents observer reservation as a normal Prompt
source', async () => {
+         const open = jest.fn();
+         const query = [{ name: 'sessionId', value: LOCATOR.sessionId }];
+         const coordinator = new PromptLiveSessionPanelCoordinatorService({
+             liveSession: {
+                 listActiveSnapshots: () => [
+                     {
+                         kind: 'reservation',
+                         runRef: RUN,
+                         viewerGroupId: 'easy-agents',
+                         ownership: 'observer',
+                         replayMode: 'replaceTurnAssistants',
+                         sourceStartedAtMs: 1,
```

```
+                         abortRequested: false,
+                         locatorEpoch: 1,
+                     },
+                 ],
+                 releaseViewerLease: () => undefined,
+                 isLocatorDestructionClaimed: () => false,
+                 onObserverInactive: () => () => {},
+                 onDestructionClaimReleased: () => () => {},
+             },
+             releaseEasyAgentsViewer: async () => {},
+         });
+
+         await coordinator.openPromptPanel({
+             locator: LOCATOR,
+             query,
+             port: { open },
+         });
+
+         expect(open).toHaveBeenCalledWith(query);
+     });
+
```

AICode/src/extension/src/extension/panels/prompt/liveSession/__test__/PromptLiveSessionPanelCoordinator.service.test.ts

Copy

Review

Accept

🔧 ✅ *patching file*: *PromptLiveSessionPanelQuery.util.test.ts*

AICode/src/extension/src/extension/panels/prompt/liveSession/__test__/PromptLiveSessionPanelQuery.util.test.ts

Copy

Review

Accept

PatchFull file

```
-         ).toBe('invalid');
-     });
-
+         ).toBe('invalid');
+         expect(
+             parsePromptLiveSessionExplicitQuery([
+                 { name: 'sessionId', value: 'a'.repeat(32) },
+                 { name: 'sessionId', value: 'b'.repeat(32) },
+                 { name: 'openMode', value: 'liveSession' },
+             ]).kind,
+         ).toBe('invalid');
+         expect(
+             parsePromptLiveSessionExplicitQuery([
+                 { name: 'storageScopeRelPath', value: '' },
+                 { name: 'storageScopeRelPath', value: 'other' },
+                 { name: 'openMode', value: 'liveSession' },
+             ]).kind,
+         ).toBe('invalid');
+     });
+
```

AICode/src/extension/src/extension/panels/prompt/liveSession/__test__/PromptLiveSessionPanelQuery.util.test.ts

Copy

Review

Accept

🔧 ✅ *patching file*: *SidebarQueryRunningBridge.race.test.ts*

AICode/src/extension/src/messageBus/handlers/ia/__test__/SidebarQueryRunningBridge.race.test.ts
Copy
Review
Accept
PatchFull file

```
-     async () => {},
- );
-
+     async () => {},
+ );
+ const observerSettledMock = jest.fn<
+     (runRef: LiveSessionRunRef) => Promise<void>
+ >(async () => {});
+
```

```
-         sourceSettled: sourceSettledMock,
-
+         sourceSettled: sourceSettledMock,
+         observerSettled: observerSettledMock,
+
```

```
-         sourceSettledMock.mockResolvedValue(undefined);
-
+         sourceSettledMock.mockResolvedValue(undefined);
+         observerSettledMock.mockResolvedValue(undefined);
+
```

```
-         );
-     });
-
+         );
+     });
+
+     it('returns a completed observer with an open panel to Idle instead of removing its
Activity row', async () => {
+         const service = new SidebarQueryActivityRuntimeService();
+         const observer = service.allocateStart({
+             runRef: RUN_A,
+             ownership: 'observer',
+             sourceStartedAtMs: Date.now(),
+             panelState: 'open',
+         });
+
+         await service.onObserverStop(RUN_A, observer.runId);
+
+         expect(observerSettledMock).toHaveBeenCalledWith(RUN_A);
+         expect(panelClosedMock).not.toHaveBeenCalled();
+         expect(service.getSnapshot(LOCATOR)).toBeUndefined();
+     });
+
```

AICode/src/extension/src/messageBus/handlers/ia/__test__/SidebarQueryRunningBridge.race.test.ts
Copy
Review
Accept

🔧 ✅ *reading project*: *NormalPromptLiveSessionSource.service.test.ts* [L1-L35]
🔧 ❌ *patching file*: *NormalPromptLiveSessionSource.service.test.ts*
AICode/src/extension/src/aiTools/orch/liveSession/source/__test__/NormalPromptLiveSessionSource.service.test.ts
Error
(what is it?)

🔧 ✅ *reading project*: NormalPromptLiveSessionSource.service.test.ts [L450-L510]
🔧 ❌ *patching file*: NormalPromptLiveSessionSource.service.test.ts
AICode/src/extension/src/aiTools/orch/liveSession/source/__test__/NormalPromptLiveSessionSource.service.test.ts
Error
(what is it?)

🔧 ✅ *patching file*: NormalPromptLiveSessionSource.service.test.ts
AICode/src/extension/src/aiTools/orch/liveSession/source/__test__/NormalPromptLiveSessionSource.service.test.ts
Copy
Review
Accept
PatchFull file

```
-         >(() => ({ becameReplayReady: false }));
-         const errorChunk: AiStreamedChunk = {
-
+         >(() => ({ becameReplayReady: false }));
+         const patchLastAssistantMeta = jest.fn(async () => {
+             throw new Error('durable patch failed');
+         });
+         const errorChunk: AiStreamedChunk = {
+
```

```
-                     getThreadMessagesSnapshot: () => [],
-                     patchLastAssistantMeta: async () => {
-                         throw new Error('durable patch failed');
-                     },
-
+                     getThreadMessagesSnapshot: () => [],
+                     patchLastAssistantMeta,
+
```

```
-         ).rejects.toThrow('durable patch failed');
-         expect(appendChunk).not.toHaveBeenCalled();
-
+         ).rejects.toThrow('durable patch failed');
+         expect(appendChunk).not.toHaveBeenCalled();
+         expect(patchLastAssistantMeta).toHaveBeenCalledTimes(1);
+
```

AICode/src/extension/src/aiTools/orch/liveSession/source/__test__/NormalPromptLiveSessionSource.service.test.ts
Copy
Review
Accept

🔧 ✅ *reading project*: 🔎 e "Specif action fixture", 🔎 e "AiQuery specifAction fixture", specifActionAutoRunQuery.util.test.ts [L1-L35], usePromptLiveSession.hook.ts, useAiStreamQueue.hook.ts [L25-L110], useAiStreamQueue.hook.ts [L245-L330], useAiStreamQueue.hook.ts [L405-L470], usePromptStreamDispatchGateQueue.hook.ts [L1-L250], usePromptStreamDispatchGateQueue.hook.ts [L400-L760], usePromptStreamDispatchGateQueue.hook.ts [L760-L980], useAiStream.hook.ts [L35-L150], useAiStream.hook.ts [L175-L540], useAiStream.hook.ts [L560-L850], useAiStream.hook.ts [L840-L930], Prompt.tsx [L380-L455], Prompt.tsx [L650-L790], Prompt.tsx [L1220-L1325], 🔎 e "Marqueur patch accidentel", useAiStream.hook.promptSourceIdentity.test.ts [L1-L230]
🔧 ✅ *patching file*: useAiStream.hook.promptSourceIdentity.test.ts
AICode/src/frontend/src/messageBus/__test__/useAiStream.hook.promptSourceIdentity.test.ts
Grouped
(what is it?)

reading project: e *"Réservations sans ownership"*, *EasyAgentsBatchRunManager.service.ts* [L405-L455], *EasyAgentsBatchLastViewerStop.service.ts* [L1-L220], e *"Composer interactive expectations"*, *PromptInput.batchLiveViewerMode.test.tsx* [L1-L70], *usePromptRunCommands.startAcceptance.test.tsx*

patching file: *usePromptRunCommands.startAcceptance.test.tsx*

AICode/src/frontend/src/app/prompt/Prompt/__test__/usePromptRunCommands.startAcceptance.test.tsx

Copy

Review

Accept

PatchFull file

```
-
- import { describe, it } from '@jest/globals';
- import type { AiStreamStartResult } from
'../../../../messageBus/AiStreamStartArgs.type.js';
-
+
+ import { beforeEach, describe, it, jest } from '@jest/globals';
+ import { act, renderHook } from '@testing-library/react';
+ import { useMemo, useRef, useState } from 'react';
+
+ import type {
+     AiStreamStartArgs,
+     AiStreamStartResult,
+ } from '../../../../messageBus/AiStreamStartArgs.type.js';
+ import type { AiStreamedChunk } from
'../../../../sharedlib/types/aiTypes/AiStreamedChunk.type.js';
+ import type { PromptMessage } from '../../PromptHistory/promptHistoryTypes.type.js';
+ import type { ToolCallNoticeApi } from '../usePromptToolCallNotice.hook.js';
+
+ const startMock = jest.fn<
+     (
+         args: Extract<AiStreamStartArgs, { kind: 'promptQuery' }>,
+     ) => AiStreamStartResult
+ >();
+ const toolCallNoticeMock = {
+     ingestToolCalls: jest.fn(),
+     ensureGapAfterNotice: jest.fn(),
+     finalizeActiveNoticeOnFirstMessageToken: jest.fn(),
+     finalizeActiveNoticeOnFirstThinkingToken: jest.fn(),
+     finalizeActiveNoticeAtTurnEnd: jest.fn(),
+     reset: jest.fn(),
+     webSearchOpenIfNeeded: jest.fn(),
+     webSearchAddItem: jest.fn(),
+     webSearchSetResults: jest.fn(),
+     webSearchMarkError: jest.fn(),
+     markWriteOrPatchErrorByPath: jest.fn(),
+     ciOpenItem: jest.fn(),
+     ciAppendCode: jest.fn(),
+     ciAppendOutputs: jest.fn(),
+     ciAppendFile: jest.fn(),
+     ciFinalize: jest.fn(),
+     applyToolCallsUpdatePayload: jest.fn(),
+ } satisfies ToolCallNoticeApi;
+
+ jest.unstable_mockModule('../usePromptAiQueryOptionsBase.hook.js', () => ({
+     usePromptAiQueryOptionsBase: () => ({
+         reasoning: 'medium',
+         verbosity: 'low',
+     }),
+ }));
```

```
+
+ const { usePromptRunCommands } =
+     await import('../usePromptRunCommands.hook.js');
+
+ const sessionLocator = {
+     sessionId: 'a'.repeat(32),
+     storageScopeRelPath: '',
+ } as const;
+
+ const noop = (): void => {};
+
+ /** Expose all seven command entry points with observable shared Prompt mutations. */
+ function useCommandHarness() {
+     const [messages, setMessages] = useState<PromptMessage[]>([
+         { turnId: 'turn', uid: 'user', role: 'user', text: 'Question' },
+         {
+             turnId: 'turn',
+             uid: 'assistant',
+             role: 'assistant',
+             text: 'Partial answer',
+         },
+     ]);
+     const [uiPending, setUiPending] = useState(false);
+     const [, setStreamingThinkingTokens] = useState<string[]>([]);
+     const [, setStreamingTokens] = useState<string[]>([]);
+     const commands = usePromptRunCommands({
+         sessionLocator,
+         dbSettingsExtUiPrompt: {
+             autoScroll: true,
+             paginationEnabled: false,
+         },
+         start: startMock,
+         messages,
+         setMessages,
+         setUiPending,
+         setStreamingThinkingTokens,
+         setStreamingTokens,
+         streamingMessageUidRef: useRef<string | undefined>(undefined),
+         streamingThinkingUidRef: useRef<string | undefined>(undefined),
+         currentTurnIdRef: useRef<string | undefined>(undefined),
+         baseAssistantTextRef: useRef(''),
+         lastConsumedRawIndexRef: useRef(0),
+         toolCallNotice: toolCallNoticeMock,
+         metaOnlyFinalizedTurnIdRef: useRef<string | undefined>(undefined),
+         lastMetaChunkFinalizedRef: useRef<AiStreamedChunk | undefined>(
+             undefined,
+         ),
+         resetLastReceivedStreamKind: noop,
+     });
+     return useMemo(
+         () => ({ ...commands, messages, uiPending }),
+         [commands, messages, uiPending],
+     );
+ }
+
-------------------------------------------------------------------------------
- describe('Prompt command acceptance contract', () => {
-     it('represents refusal without fabricated identities and acceptance with exact
identities', () => {
-         const refused: AiStreamStartResult = { accepted: false };
-         const accepted: AiStreamStartResult = {
-
+ describe('Prompt command acceptance contract', () => {
+     beforeEach(() => {
```

```
+         jest.clearAllMocks();
+     });
+
+     it('keeps every command mutation-free when synchronous start is refused', () => {
+         startMock.mockReturnValue({ accepted: false });
+         const { result } = renderHook(() => useCommandHarness());
+         const initialMessages = result.current.messages;
+         const results: AiStreamStartResult[] = [];
+
+         act(() => {
+             results.push(result.current.handleSend('Send', {}, []));
+             results.push(
+                 result.current.handleSendSpecifAction({
+                     type: 'specifAction',
+                     action: 'code',
+                     specifName: 'SPECIF_TEST',
+                     specifMarkdown: 'Specification',
+                     strictMode: true,
+                     programmingDocEnabled: true,
+                     additionalInstructions: '',
+                 }),
+             );
+             results.push(result.current.handleContinueResponse());
+             results.push(result.current.handleRetryQuery());
+             results.push(
+                 result.current.handleSend(
+                     'Compact',
+                     {},
+                     [],
+                     undefined,
+                     'compactContextNow',
+                 ),
+             );
+             results.push(result.current.handleReplyButton('✅ Continue'));
+             results.push(
+                 result.current.handleSubmitEdit('turn', 'Edited', {}, []),
+             );
+         });
+
+         expect(results).toEqual(Array.from({ length: 7 }, () => ({
+             accepted: false,
+         })));
+         expect(result.current.messages).toBe(initialMessages);
+         expect(result.current.uiPending).toBe(false);
+         expect(startMock).toHaveBeenCalledTimes(7);
+         expect(toolCallNoticeMock.reset).not.toHaveBeenCalled();
+     });
+
+     it('returns the exact accepted identity from Reply and commits one scaffold', () =>
{
+         startMock.mockImplementation((args) => ({
+
-------------------------------------------------------------------------------
-             runRef: {
-                 locator: {
-                     sessionId: 'a'.repeat(32),
-                     storageScopeRelPath: '',
-                 },
-                 turnId: 'turn',
-
+             runRef: {
+                 locator: sessionLocator,
+                 turnId: args.turnId,
+
```

```
-             streamInstanceId: 'stream',
-         };
-         expect(refused).toEqual({ accepted: false });
-         expect(accepted).toEqual(
-
+             streamInstanceId: 'stream',
+         }));
+         const { result } = renderHook(() => useCommandHarness());
+         let replyResult: AiStreamStartResult | undefined;
+
+         act(() => {
+             replyResult = result.current.handleReplyButton('✅ Continue');
+         });
+
+         expect(replyResult).toEqual(
+
```

```
-             }),
-         );
-
+             }),
+         );
+         expect(startMock).toHaveBeenCalledWith(
+             expect.objectContaining({
+                 kind: 'promptQuery',
+                 aiQuery: { type: 'prompt', promptValue: '[Continue]' },
+             }),
+         );
+         expect(result.current.uiPending).toBe(true);
+         expect(result.current.messages).toHaveLength(5);
+         expect(
+             result.current.messages.filter(
+                 (message) => message.role === 'user',
+             ),
+         ).toHaveLength(2);
+         expect(toolCallNoticeMock.reset).toHaveBeenCalledTimes(1);
+
```

AICode/src/frontend/src/app/prompt/Prompt/__test__/usePromptRunCommands.startAcceptance.test.tsx
Copy
Review
Accept

🔧 ✅ *reading project*: 🔎 e *"Queue A API fixtures"*, 🔎 e *"Queue B API fixtures"*, 🔎 e *"RunMerged signatures"*, 🔎 e *"DispatchMerged signatures"*, 📄 *useAiStream.hook.rawLiveSideband.test.ts* [L1-L45], 📄 *usePromptLiveSession.hydration.test.tsx*
🔧 ✅ *patching file*: 📄 *usePromptLiveSession.hydration.test.tsx*
AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.hydration.test.tsx
Copy
Review
Accept
PatchFull file

```
-
- import { describe, it } from '@jest/globals';
-
+
+ import { beforeEach, describe, it, jest } from '@jest/globals';
+ import { act, renderHook, waitFor } from '@testing-library/react';
+ import { useMemo, useRef, useState } from 'react';
+ import type { SetStateAction } from 'react';
+
```

```
+ import type { AiStreamStartResult } from
'../../../../../messageBus/AiStreamStartArgs.type.js';
+ import type { PromptAttachment } from '../../../lib/PromptAttachment.type.js';
+ import type { PromptRuntimeMode } from '../PromptLiveSession.types.js';
+
+ const callLocalApiMock = jest.fn<
+     (path: string, args: unknown) => Promise<unknown>
+ >();
+ const setPromptValueMock = jest.fn<
+     (value: SetStateAction<string>) => void
+ >();
+ const setAttachmentsMock = jest.fn<
+     (value: SetStateAction<readonly PromptAttachment[]>) => void
+ >();
+ const setNextTurnIdMock = jest.fn<(value: string | undefined) => void>();
+
+ jest.unstable_mockModule(
+     '../../../../../messageBus/callLocalApi.util.js',
+     () => ({ callLocalApi: callLocalApiMock }),
+ );
+ jest.unstable_mockModule('../../../../../messageBus/closeSelf.util.js', () => ({
+     closeSelf: async () => {},
+ }));
+ jest.unstable_mockModule(
+     '../../../../../vsc-design-system/popups/VscAlertDialog.js',
+     () => ({ showVscAlertDialog: async () => {} }),
+ );
+
+ const { bindNormalPromptLiveSession } =
+     await import('../../adapters/NormalPromptLiveSession.adapter.js');
+ const { usePromptLiveSession } = await import('../usePromptLiveSession.hook.js');
+
+ const locator = {
+     sessionId: 'a'.repeat(32),
+     storageScopeRelPath: '',
+ } as const;
+ const bootstrap = bindNormalPromptLiveSession(
+     {
+         openMode: 'liveSession',
+         liveSessionTurnId: 'turn',
+         liveSessionViewerLeaseId: 'lease',
+         liveSessionClosePolicy: 'continueOnPanelClose',
+         liveSessionSourceId: 'source',
+         liveSessionAdapter: 'normalPrompt',
+     },
+     locator,
+ );
+
+ const noop = (): void => {};
+ const refuseLiveObserver = (): AiStreamStartResult => ({ accepted: false });
+
--------------------------------------------------------------------------------
-
- import type { PromptLiveSessionSnapshotPolicy } from '../PromptLiveSession.types.js';
-
+
+ /** Run the real Open/retry policy while exposing its mode transition. */
+ function useHydrationHarness() {
+     const [mode, setMode] = useState<PromptRuntimeMode>(
+         'liveSessionOpening',
+     );
+     const [, setFailure] = useState<string | undefined>(undefined);
+     const [, setHydrating] = useState(false);
+     const localOverrideRef = useRef(false);
```

```
+     const api = usePromptLiveSession({
+         enabled: true,
+         bootstrap,
+         openThreadId: undefined,
+         promptRuntimeMode: mode,
+         setPromptRuntimeMode: setMode,
+         setHandoffFailureMessage: setFailure,
+         replaceAllMessages: noop,
+         setPromptValue: setPromptValueMock,
+         setAttachments: setAttachmentsMock,
+         setNextTurnId: setNextTurnIdMock,
+         setOpenThreadId: noop,
+         setHydrateScrollSeq: noop,
+         setIsHydrating: setHydrating,
+         hasLocalPromptOverrideRef: localOverrideRef,
+         resetPromptRunState: noop,
+         startLiveObserver: refuseLiveObserver,
+         isStreaming: false,
+         lastMetaChunk: undefined,
+         streamInstanceId: undefined,
+         finalizedStreamInstanceId: undefined,
+         streamOutcome: undefined,
+         terminalDataSeen: false,
+         transportStopAavailable: false,
+     });
+     return useMemo(() => ({ api, mode }), [api, mode]);
+ }
+
--------------------------------------------------------------------------------
- describe('usePromptLiveSession hydration policies', () => {
-     it('distinguishes initial full hydration from preserve-local retry', () => {
-         const initial: PromptLiveSessionSnapshotPolicy = 'hydrateFromSnapshot';
-         const retry: PromptLiveSessionSnapshotPolicy = 'preserveLocal';
-         expect(initial).not.toBe(retry);
-
+ describe('usePromptLiveSession hydration policies', () => {
+     beforeEach(() => {
+         jest.clearAllMocks();
+     });
+
+     it('uses preserveLocal on Retry without reading draft, attachment, or nextTurnId
getters', async () => {
+         const session = {
+             meta: {
+                 formatVersionNumber: 1,
+                 sessionId: locator.sessionId,
+                 title: 'Chat',
+                 createdAt: 1,
+                 updatedAt: 2,
+                 firstSendState: 'postSend',
+             },
+             messages: [],
+         };
+         Object.defineProperties(session, {
+             promptDraft: {
+                 get: () => {
+                     throw new Error('promptDraft must stay unread');
+                 },
+             },
+             promptDraftAttachments: {
+                 get: () => {
+                     throw new Error('attachments must stay unread');
+                 },
+             },
```

```
+         });
+         Object.defineProperty(session.meta, 'nextTurnId', {
+             get: () => {
+                 throw new Error('nextTurnId must stay unread');
+             },
+         });
+         callLocalApiMock
+             .mockResolvedValueOnce({ kind: 'not_found' })
+             .mockResolvedValueOnce({
+                 kind: 'terminal',
+                 session,
+                 threadId: locator.sessionId,
+                 turnId: 'turn',
+                 sourceId: 'source',
+             });
+
+         const { result } = renderHook(() => useHydrationHarness());
+         await waitFor(() =>
+             expect(result.current.mode).toBe('liveSessionReplayFailed'),
+         );
+
+         act(() => result.current.api.retryHandoff());
+         await waitFor(() => expect(result.current.mode).toBe('normal'));
+
+         expect(setPromptValueMock).not.toHaveBeenCalled();
+         expect(setAttachmentsMock).not.toHaveBeenCalled();
+         expect(setNextTurnIdMock).not.toHaveBeenCalled();
+
```

AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.hydration.test.tsx
Copy
Review
Accept

🔧 ✅ *reading project*: *usePromptLiveSession.normalPromptIsomorphism.test.tsx*, 🔎h *"Hydration assistant fixture"*, *frontSessionToPromptMessages.summary.nextTurn.hydration.test.ts* [L1-L55]
🔧 ✅ *patching file*: *usePromptLiveSession.normalPromptIsomorphism.test.tsx*
AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.normalPromptIsomorphism.test.tsx
Copy
Review
Accept
PatchFull file

```
- import { describe, it } from '@jest/globals';
-
-
+ import { describe, it } from '@jest/globals';
+
+ import { AiStopErrorReason } from
'../../../../../sharedlib/types/aiTypes/AiStopErrorReason.type.js';
+ import { buildStopSuffix } from
'../../../../../sharedlib/types/aiTypes/aiStopSuffix.util.js';
+
```

```
- import { ZLiveSessionOpenResult } from
'../../../../../sharedlib/types/liveSessionTypes/LiveSessionOpenResult.type.js';
-
+ import { ZLiveSessionOpenResult } from
'../../../../../sharedlib/types/liveSessionTypes/LiveSessionOpenResult.type.js';
+ import type { PromptMessage } from '../../../PromptHistory/promptHistoryTypes.type.js';
+ import { applyHydratedPromptSessionState } from
'../../../Prompt/applyHydratedPromptSessionState.util.js';
+
```

```
-     it.each(['replaceTurnAssistants', 'continueAssistant'] as const)(
-         'preserves strict source identity for %s replay',
-
+     it.each(['replaceTurnAssistants', 'continueAssistant'] as const)(
+         'hydrates the same canonical terminal assistant text produced by streaming for
%s replay',
+
-         (replayMode) => {
-
+         (replayMode) => {
+             const streamedAssistantText = `Answer${buildStopSuffix(
+                 AiStopErrorReason.SYSTEM_ERROR,
+             )}`;
+
-                     },
-                     messages: [],
-
+                     },
+                     messages: [
+                         {
+                             role: 'assistant',
+                             turnId: 'turn',
+                             content: [{ text: streamedAssistantText }],
+                         },
+                     ],
+
-             });
-             expect(result).toEqual(
-                 expect.objectContaining({ sourceId: 'source', replayMode }),
-             );
-
+             });
+             const hydratedMessages: PromptMessage[][] = [];
+
+             applyHydratedPromptSessionState({
+                 mode: 'hydrateFromSnapshot',
+                 rawSession: result.session,
+                 rawThreadId: result.threadId,
+                 options: { allowOrphanToolResults: true },
+                 scrollStrategy: 'classicHydration',
+                 replaceAllMessages: (messages) => {
+                     hydratedMessages.push(messages);
+                 },
+                 setPromptValue: () => {},
+                 setAttachments: () => {},
+                 setNextTurnId: () => {},
+                 setOpenThreadId: () => {},
+                 setHydrateScrollSeq: () => {},
+                 hasLocalPromptOverrideRef: { current: false },
+                 resetPromptRunState: () => {},
+             });
+
+             const assistant = hydratedMessages[0]?.find(
+                 (message) =>
+                     message.role === 'assistant' &&
+                     message.turnId === 'turn' &&
+                     message.assistantBubbleKind === 'msg',
+             );
+             expect(assistant?.text).toBe(streamedAssistantText);
+             expect(result).toEqual(
+                 expect.objectContaining({ sourceId: 'source', replayMode }),
```

```
+             );
+
```

AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.normalPromptIsomorphism.test.tsx
Copy
Review
Accept

🔧 ✅ *reading project*: 🔎 e *"Sound cleanup tests"*, *SidebarStore.rehydrate.manager.test.ts* [L1-L45]
🔧 ✅ *patching file*: *SidebarStore.rehydrate.manager.test.ts*
AICode/src/extension/src/extension/panels/sidebar/__test__/SidebarStore.rehydrate.manager.test.ts
Grouped
(what is it?)

🔧 ✅ *reading project*: *SidebarStore.manager.queryRunningUpsert.test.ts*
🔧 ✅ *patching file*: *SidebarStore.manager.queryRunningUpsert.test.ts*
AICode/src/extension/src/extension/panels/sidebar/__test__/SidebarStore.manager.queryRunningUpsert.test.ts
Copy
Review
Accept
PatchFull file

```
- } from '../SidebarStore.core.manager.js';
- import {
-
+ } from '../SidebarStore.core.manager.js';
+ import {
+     observerSettled,
+
```

```
-     });
- });
-
+     });
+
+     it('maps a settled observer back to Idle while its Prompt panel remains open',
async () => {
+         await sourceStarted({
+             runRef: RUN,
+             ownership: 'observer',
+             panelState: 'open',
+             sourceStartedAtMs: Date.now(),
+             stopRequested: false,
+         });
+         await observerSettled(RUN);
+         expect(
+             sidebarStore.sidebarContentState.activityState.queryState.items[0],
+         ).toEqual(
+             expect.objectContaining({ status: 'idle', panelState: 'open' }),
+         );
+     });
+ });
+
```

AICode/src/extension/src/extension/panels/sidebar/__test__/SidebarStore.manager.queryRunningUpsert.test.ts
Copy
Review
Accept

🔧 ✅ *patching file*: *usePromptLiveSession.hydration.test.tsx*

AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.hydration.test.tsx
Grouped
(what is it?)

🔧 ✅ *patching file*: *usePromptLiveSession.normalPrompt.test.tsx*
AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.normalPrompt.test.tsx
Grouped
(what is it?)

🔧 ✅ *reading project*: *PromptStreamDispatchGateQueue.test.ts* [L730-L850], *PromptStreamDispatchGateQueue.test.ts* [L1010-L1135]
🔧 ✅ *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?)

🔧 ✅ *scanning git*: ♈ *"Vérifier le diff courant"*, ♈ *"Résumé des fichiers modif"*, ♈ *"Chemins modifiés"*
🔧 ✅ *patching file*: *useAiStream.hook.promptSourceIdentity.test.ts*
AICode/src/frontend/src/messageBus/__test__/useAiStream.hook.promptSourceIdentity.test.ts
Grouped
(what is it?)

🔧 ✅ *reading project*: 🔎 e *"Fallback persisted shape"*, *usePromptLiveSession.normalPrompt.test.tsx* [L95-L175]
🔧 ✅ *patching file*: *usePromptLiveSession.normalPrompt.test.tsx*
AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.normalPrompt.test.tsx
Grouped
(what is it?)

🔧 ✅ *reading project*: *applyHydratedPromptSessionState.util.ts* [L230-L346], 🔎 e *"usePromptLiveSession usages"*, *useAiStream.hook.liveSessionObserver.test.ts*, 🔎 e *"Any dans changements"*, 🔎 e *"TODO dans changements"*, 🔎 e *"eslint disable changements"*, *PromptLiveSessionPanelQuery.util.ts* [L1-L90], *PromptLiveSessionPanelCoordinator.service.ts* [L45-L255]
🔧 ✅ *patching file*: *usePromptStreamDispatchGateQueue.hook.ts*
AICode/src/frontend/src/messageBus/usePromptStreamDispatchGateQueue.hook.ts
Grouped
(what is it?)

🔧 ✅ *reading project*:
usePromptRunCommands.startAcceptance.test.tsx,
usePromptLiveSession.normalPrompt.test.tsx,
usePromptLiveSession.hydration.test.tsx,
usePromptLiveSession.handoff.test.tsx,
usePromptLiveSession.normalPromptIsomorphism.test.tsx, *PromptStreamDispatchGateQueue.test.ts* [L1040-

L1135], AiStreamMergedQueue.test.ts [L1-L60], AiStreamMergedQueue.test.ts [L525-L600], promptHistoryTypes.type.ts [L1-L130]
🔧 ✅ *patching file*: usePromptLiveSession.hydration.test.tsx
AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.hydration.test.tsx
Grouped
(what is it?)

🔧 ✅ *patching file*: usePromptLiveSession.handoff.test.tsx
AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.handoff.test.tsx
Grouped
(what is it?)

🔧 ✅ *reading project*: 🔎 e "Assertions type ajoutées", NormalPromptLiveSession.adapter.ts [L40-L90]
🔧 ✅ *patching file*: NormalPromptLiveSession.adapter.ts
AICode/src/frontend/src/app/prompt/liveSession/adapters/NormalPromptLiveSession.adapter.ts
Grouped
(what is it?)

🔧 ✅ *reading project*: NormalPromptLiveSessionSource.service.test.ts [L450-L510], SidebarQueryRunningBridge.usage.test.ts [L1-L60], 🔎 e "Composer interactive usage", 🔎 e "Snapshot equality tests", 🔎 e "Définition kill EasyAgents", 👉 "Kill EasyAgents", 🔎 e "Observer inactive émission", LiveSessionRegistry.service.ts [L590-L650], LiveSessionRegistry.service.ts [L250-L330], LiveSessionRegistry.service.ts [L330-L380], 🔎 e "Retire record impl", LiveSessionRegistry.service.ts [L1000-L1055]
🔧 ✅ *patching file*: LiveSessionRegistry.service.ts
AICode/src/extension/src/aiTools/orch/liveSession/registry/LiveSessionRegistry.service.ts
Grouped
(what is it?)

🔧 ✅ *reading project*: LiveSessionRegistry.service.ts [L310-L380], LiveSessionRegistry.service.ts [L380-L395], 🔎 e "Observer event test", LiveSessionRegistry.service.test.ts [L235-L280]
🔧 ✅ *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*: 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?)

🔧 ✅ *patching file*: LiveSessionService.ts
AICode/src/extension/src/aiTools/orch/liveSession/core/LiveSessionService.ts
Copy
Review

Accept
PatchFull file

```
-
-     /** Subscribe to exact observer-inactive wake-ups. */
-
+
+     /** Subscribe to exact observer-inactive cleanup wake-ups, including post-release
settlement retries. */
+
```

AICode/src/extension/src/aiTools/orch/liveSession/core/LiveSessionService.ts
Copy
Review
Accept

🔧 ✅ *reading project*: 🔎 e *"buildStopSuffix signature"*, 👉 *"Signature suffixe"*
🔧 ✅ *scanning recent files*
🔧 ✅ *patching file*: 📄 *useAiStream.hook.ts*
AICode/src/frontend/src/messageBus/useAiStream.hook.ts
Grouped
(what is it?)

🔧 ✅ *patching file*: 📄 *AiStreamStartArgs.type.ts*
AICode/src/frontend/src/messageBus/AiStreamStartArgs.type.ts
Grouped
(what is it?)

🔧 ✅ *reading project*: 🔎 e *"Gate API objets"*, 🔎 e *"Queue API objets"*, 🔎 e *"Snapshot mocks directs"*
🔧 ✅ *patching file*: 📄 *PromptLiveSessionPanelCoordinator.service.ts*
AICode/src/extension/src/extension/panels/prompt/liveSession/PromptLiveSessionPanelCoordinator.service.ts
Grouped
(what is it?)

🔧 ✅ *reading project*: 🔎 e *"Transport Stop usages Prompt"*, 🔎 e *"Final lock définition"*, 🔎 e *"Appels markSourceSettled"*, 📄 *EasyAgentsBatchRunExecution.service.ts* [L365-L425]
🔧 ✅ *patching file*: 📄 *LiveSessionRegistry.service.ts*
AICode/src/extension/src/aiTools/orch/liveSession/registry/LiveSessionRegistry.service.ts
Grouped
(what is it?)

🔧 ✅ *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*: 📄 *LiveSessionRegistry.service.test.ts*
AICode/src/extension/src/aiTools/orch/liveSession/registry/__test__/LiveSessionRegistry.service.test.ts
Grouped
(what is it?)

🔧 ✅ *patching file*: 📄 *PromptLiveSessionPanelCoordinator.service.ts*

AICode/src/extension/src/extension/panels/prompt/liveSession/PromptLiveSessionPanelCoordinator.service.ts
Grouped
(what is it?)

🔧 ✅ *patching file*: 📄 *usePromptLiveSession.normalPrompt.test.tsx*
AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.normalPrompt.test.tsx
Grouped
(what is it?)

🔧 ✅ *patching file*: 📄 *usePromptLiveSession.hydration.test.tsx*
AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.hydration.test.tsx
Grouped
(what is it?)

🔧 ✅ *patching file*: 📄 *usePromptLiveSession.handoff.test.tsx*
AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.handoff.test.tsx
Grouped
(what is it?)

🔧 ✅ *patching file*: 📄 *usePromptRunCommands.startAcceptance.test.tsx*
AICode/src/frontend/src/app/prompt/Prompt/__test__/usePromptRunCommands.startAcceptance.test.tsx
Grouped
(what is it?)

🔧 ✅ *patching file*: 📄 *PromptStreamDispatchGateQueue.test.ts*
AICode/src/frontend/src/messageBus/__test__/PromptStreamDispatchGateQueue.test.ts
Grouped
(what is it?)

🔧 ✅ *patching file*: 📄 *usePromptLiveSession.normalPrompt.test.tsx*
AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.normalPrompt.test.tsx
Grouped
(what is it?)

🔧 ✅ *patching file*: 📄 *usePromptLiveSession.hydration.test.tsx*
AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.hydration.test.tsx
Grouped
(what is it?)

🔧 ✅ *patching file*: 📄 *usePromptRunCommands.startAcceptance.test.tsx*
AICode/src/frontend/src/app/prompt/Prompt/__test__/usePromptRunCommands.startAcceptance.test.tsx
Grouped
(what is it?)

🔧 ✅ *reading project*: 🔎 e *"Assistant bubble kind hydration"*
🔧 ✅ *patching file*: 📄 *Prompt.tsx*
AICode/src/frontend/src/app/prompt/Prompt/Prompt.tsx
Grouped
(what is it?)

reading project: e *"Queue identity doc"*, *prompt-stream-dispatch-lock.programming-doc-readme.md* [L1-L220]

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

```
1. **Queue A** (existing): `useAiStreamQueue.hook.ts`
     - Receives raw `AiStreamedChunk` events.
     - Performs front-prefix coalescing (tokens, thinking tokens, tool_calls).
1. **Queue A** (existing): `useAiStreamQueue.hook.ts`
     - Receives raw `AiStreamedChunk` events with the exact `{ sourceId,
streamInstanceId }` attempt identity.
     - Performs front-prefix coalescing only inside one contiguous exact attempt (tokens,
thinking tokens, tool_calls).
```

```
2. **Queue B** (new): `usePromptStreamDispatchGateQueue.hook.ts`
     - Receives already-merged chunks from Queue A.
2. **Queue B** (new): `usePromptStreamDispatchGateQueue.hook.ts`
     - Receives already-merged chunks and their unchanged exact attempt identity from
Queue A.
```

```
     - Provides backpressure: Queue A awaits Queue B, so blocking has zero polling and
minimal CPU.
     - Provides backpressure: Queue A awaits Queue B, so blocking has zero polling and
minimal CPU.
     - Advances a reset generation so an older post-`await` dispatch cannot remove or
mutate successor work.
```

```
The reset/finalization boundary is guarded by `sourceId + streamInstanceId`. A stale
callback, continuation after `await`, or Queue A `runMerged` item from an older accepted
stream cannot 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 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.
```

AICode/src/frontend/src/messageBus/prompt-stream-dispatch-lock.programming-doc-readme.md

Copy

Review

Accept

reading project: e *"Wake-up doc liveSession"*, *liveSession.programming-doc-readme.md* [L55-L78]

patching file: *liveSession.programming-doc-readme.md*

AICode/src/extension/src/aiTools/orch/liveSession/liveSession.programming-doc-readme.md

Copy

Review

Accept

PatchFull file

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`. The registry emits deeply immutable advisory wake-ups only after complete observer-inactive and destruction-claim-release mutations, and isolates listener failures.

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`. 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.

The normal Prompt source persists every Orch `meta.error` before registry append and frontend forwarding. Synthetic SYSTEM_ERROR or INTERRUPTED terminals follow the same durable-before-append rule. The source service never performs post-forwarding finalization.

The normal Prompt source persists every Orch `meta.error` before registry append and frontend forwarding. Once a terminal was observed from Orch, a failure of that durability barrier fails the transport without replacing the original terminal with a synthetic one. Synthetic SYSTEM_ERROR or INTERRUPTED terminals are created only when no source terminal was observed and follow the same durable-before-append rule. The source service never performs post-forwarding finalization.
AICode/src/extension/src/aiTools/orch/liveSession/liveSession.programming-doc-readme.md
Copy
Review
Accept

🔧 ✅ *reading project*: *PromptLiveSessionPanelCoordinator.service.test.ts* [L1-L220]
🔧 ✅ *patching file*: *usePromptLiveSession.hook.ts*
AICode/src/frontend/src/app/prompt/liveSession/core/usePromptLiveSession.hook.ts
Grouped
(what is it?)

🔧 ✅ *reading project*: 🔎 e *"Queue A appels exacts"*, 🔎 e *"Queue B appels exacts"*, 🔎 e *"API stop live source"*, 🔎 e *"Observer settled refs"*, 🔎 e *"Ownership réservations"*, *useAiStreamQueue.hook.ts* [L450-L515], *usePromptStreamDispatchGateQueue.hook.ts* [L220-L305], *usePromptStreamDispatchGateQueue.hook.ts* [L950-L1045], *useAiStream.hook.ts* [L520-L585], *useAiStream.hook.ts* [L745-L935], *useStreamListener.hook.ts* [L1-L180]
🔧 ✅ *patching file*: *prompt-stream-dispatch-lock.programming-doc-readme.md*
AICode/src/frontend/src/messageBus/prompt-stream-dispatch-lock.programming-doc-readme.md
Grouped
(what is it?)

🔧 ✅ *patching file*: *useAiStream.hook.promptSourceIdentity.test.ts*
AICode/src/frontend/src/messageBus/__test__/useAiStream.hook.promptSourceIdentity.test.ts
Grouped
(what is it?)

🔧 ✅ *patching file*: *usePromptLiveSession.normalPrompt.test.tsx*

AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.normalPrompt.test.tsx
Grouped
(what is it?)

🔧 ✅ *patching file*: *useAiStream.hook.ts*
AICode/src/frontend/src/messageBus/useAiStream.hook.ts
Grouped
(what is it?)

🔧 ✅ *patching file*: *usePromptLiveSession.hook.ts*
AICode/src/frontend/src/app/prompt/liveSession/core/usePromptLiveSession.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*: *AiStreamMergedQueue.test.ts*
AICode/src/frontend/src/messageBus/__test__/AiStreamMergedQueue.test.ts
Grouped
(what is it?)

🔧 ✅ *reading project*: *Prompt.tsx* [L500-L650], *Prompt.tsx* [L455-L510], 🔎 e *"finalIsLocked usages"*, 🔎 e *"Prompt input lock prop"*
🔧 ✅ *patching file*: *Prompt.tsx*
AICode/src/frontend/src/app/prompt/Prompt/Prompt.tsx
Grouped
(what is it?)

🔧 ✅ *patching file*: *PromptLiveSessionPanelCoordinator.service.ts*
AICode/src/extension/src/extension/panels/prompt/liveSession/PromptLiveSessionPanelCoordinator.service.ts
Grouped
(what is it?)

🔧 ✅ *patching file*: *PromptLiveSessionPanelCoordinator.service.ts*
AICode/src/extension/src/extension/panels/prompt/liveSession/PromptLiveSessionPanelCoordinator.service.ts
Grouped
(what is it?)

🔧 ✅ *patching file*: *PromptInput.batchLiveViewerMode.test.tsx*
AICode/src/frontend/src/app/prompt/PromptInput/__test__/PromptInput.batchLiveViewerMode.test.tsx
Copy
Review
Accept
PatchFull file

```
- describe('PromptInput live-session surfaces', () => {
-     it('keeps EasyAgents read-only and normalPrompt composer interactive with Send
locked', () => {
```

```
-
+ describe('PromptInput live-session surfaces', () => {
+     it('keeps EasyAgents read-only and unlocks normalPrompt editing only after replay
arming', () => {
+
```

```
-         };
-         const normalPrompt: PromptLiveSessionViewModel = {
-
+         };
+         const normalPromptStreaming: PromptLiveSessionViewModel = {
+
```

```
-                 stopEnabled: false,
-
+                 stopEnabled: false,
+                 showRetryAction: false,
+                 retryEnabled: false,
+             },
+         };
+         const normalPromptReplayFailed: PromptLiveSessionViewModel = {
+             ...normalPromptStreaming,
+             composerInteractive: false,
+             banner: {
+                 showStopAction: false,
+                 stopEnabled: true,
+
```

```
-         expect(easyAgents.showComposer).toBe(false);
-         expect(normalPrompt).toEqual(
-
+         expect(easyAgents.showComposer).toBe(false);
+         expect(normalPromptStreaming).toEqual(
+
```

```
-         );
-         expect(normalPrompt.banner?.showStopAction).toBe(false);
-
+         );
+         expect(normalPromptReplayFailed).toEqual(
+             expect.objectContaining({ composerInteractive: false }),
+         );
+         expect(normalPromptStreaming.banner?.showStopAction).toBe(false);
+
```

AICode/src/frontend/src/app/prompt/PromptInput/__test__/PromptInput.batchLiveViewerMode.test.tsx
Copy
Review
Accept

reading project: *usePromptRunCommands.hook.ts* [L80-L125]
patching file: *usePromptRunCommands.startAcceptance.test.tsx*
AICode/src/frontend/src/app/prompt/Prompt/__test__/usePromptRunCommands.startAcceptance.test.tsx
Grouped
(what is it?)

patching file: *usePromptRunCommands.startAcceptance.test.tsx*
AICode/src/frontend/src/app/prompt/Prompt/__test__/usePromptRunCommands.startAcceptance.test.tsx
Grouped
(what is it?)

🔧 ✅ *patching file*: *usePromptLiveSession.hook.ts*
AICode/src/frontend/src/app/prompt/liveSession/core/usePromptLiveSession.hook.ts
Grouped
(what is it?)

🔧 ✅ *patching file*: *usePromptLiveSession.handoff.test.tsx*
AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.handoff.test.tsx
Grouped
(what is it?)

🔧 ✅ *patching file*: *usePromptLiveSession.handoff.test.tsx*
AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.handoff.test.tsx
Grouped
(what is it?)

🔧 ✅ *reading project*: *usePromptLiveSession.hook.ts* [L200-L380], *AiStreamStartArgs.type.ts* [L1-L80], *SidebarStore.queryWindow.listeners.manager.ts* [L190-L285], *SidebarQueryRunningBridge.ts* [L155-L205], *NormalPromptLiveSessionSource.service.ts* [L175-L270], *LiveSessionRegistry.service.ts* [L315-L390], *Prompt.tsx* [L480-L535], *SidebarStore.rehydrate.manager.ts* [L1-L105]
🔧 ✅ *patching file*: *SidebarStore.rehydrate.manager.ts*
AICode/src/extension/src/extension/panels/sidebar/SidebarStore.rehydrate.manager.ts
Grouped
(what is it?)

🔧 ✅ *reading project*: *SidebarStore.rehydrate.manager.test.ts* [L90-L220]
🔧 ✅ *patching file*: *SidebarStore.rehydrate.manager.test.ts*
AICode/src/extension/src/extension/panels/sidebar/__test__/SidebarStore.rehydrate.manager.test.ts
Grouped
(what is it?)

🔧 ✅ *reading project*: 🔎 e *"Hypothèses réservation source"*, *delete.endpoint.ts* [L75-L105], *delete.endpoint.ts* [L100-L145], 🔎 e *"EasyAgents wake-up doc"*, *easyAgents.programming-doc-readme.md* [L995-L1018]
🔧 ✅ *patching file*: *easyAgents.programming-doc-readme.md*
AICode/src/extension/src/aiTools/easyAgents/easyAgents.programming-doc-readme.md
Copy
Review
Accept
PatchFull file

```
- a settled source that never reached the durable baseline is evicted by the generic core
and never becomes watchable or handoff-ready.
- observer-inactive and destruction-claim-release wake-ups are generic registry signals;
EasyAgents keeps the last-viewer kill and Retry business decisions in its adapter/runtime
layer.
- a settled source that never reached the durable baseline is evicted by the generic core
and never becomes watchable or handoff-ready.
- observer-inactive and destruction-claim-release wake-ups are generic registry signals;
EasyAgents keeps the last-viewer kill and Retry business decisions in its adapter/runtime
layer. If the last Prompt lease was released before the child completed handoff, the
final observer handoff re-emits the observer-inactive wake-up so panel cleanup retries
only after business authority is gone.
```

AICode/src/extension/src/aiTools/easyAgents/easyAgents.programming-doc-readme.md
Copy

Review
Accept

🔧 ✅ *reading project*: 🔎 e *"Isomorphisme stream id"*, *isomorphism-invariant-rule-between-streaming-and-rehydration-of-chat-sessions.programming-doc-readme.md* [L120-L220], *isomorphism-invariant-rule-between-streaming-and-rehydration-of-chat-sessions.programming-doc-readme.md*

🔧 ✅ *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 `meta.error` is durable before append, including synthetic EOF/throw terminals
and auto-accept recovery.
- Every `meta.error` is durable before append, including synthetic EOF/throw terminals
and auto-accept recovery.
- A terminal `meta.error` observed from Orch is never replaced by a synthetic terminal if
its durable patch fails; the transport fails and the registry remains non-handoffable.
```

```
- Terminal data followed by transport failure remains one visible terminal with
`outcome='transportError'`.
- Normal Prompt handoff is direct after replay, Queue A, Queue B, and frontend
finalization converge; no second Open or composer hydration occurs.
- Terminal data followed by transport failure remains one visible terminal with
`outcome='transportError'`.
- 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.
- Normal Prompt handoff is direct after replay, Queue A, Queue B, and frontend
finalization converge; no second Open or composer hydration occurs.
- Before baseline installation and replay arming, the composer remains non-editable but
its unique Stop control can abort the exact source reservation. After arming, draft and
attachment editing are allowed while every Send path remains locked.
```

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*: *liveSession.programming-doc-readme.md*

AICode/src/extension/src/aiTools/orch/liveSession/liveSession.programming-doc-readme.md

Grouped
(what is it?)

🔧 ✅ *patching file*: *PromptInput.batchLiveViewerMode.test.tsx*

AICode/src/frontend/src/app/prompt/PromptInput/__test__/PromptInput.batchLiveViewerMode.test.tsx

Grouped
(what is it?)

🔧 ✅ *patching file*: *usePromptLiveSession.normalPrompt.test.tsx*

AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.normalPrompt.test.tsx

Grouped
(what is it?)

🔧 ✅ *reading project*: usePromptRunCommands.hook.ts [L370-L540], PromptInput.tsx [L1970-L2035], 🔎 e *"Clear transport stop"*, 🔎 e *"Stop availability clear API"*
🔧 ✅ *patching file*: usePromptLiveSession.hook.ts
AICode/src/frontend/src/app/prompt/liveSession/core/usePromptLiveSession.hook.ts
Grouped
(what is it?)

🔧 ✅ *patching file*: Prompt.tsx
AICode/src/frontend/src/app/prompt/Prompt/Prompt.tsx
Grouped
(what is it?)

🔧 ✅ *patching file*: Prompt.tsx
AICode/src/frontend/src/app/prompt/Prompt/Prompt.tsx
Grouped
(what is it?)

🔧 ✅ *patching file*: usePromptLiveSession.normalPrompt.test.tsx
AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.normalPrompt.test.tsx
Grouped
(what is it?)

🔧 ✅ *reading project*: usePromptLiveSession.hook.ts [L335-L435], Prompt.tsx [L660-L755]
🔧 ✅ *patching file*: Prompt.tsx
AICode/src/frontend/src/app/prompt/Prompt/Prompt.tsx
Grouped
(what is it?)

🔧 ✅ *reading project*: SidebarStore.rehydrate.manager.test.ts [L180-L240]
🔧 ✅ *patching file*: SidebarStore.rehydrate.manager.test.ts
AICode/src/extension/src/extension/panels/sidebar/__test__/SidebarStore.rehydrate.manager.test.ts
Grouped
(what is it?)

🔧 ✅ *patching file*: AiStreamMergedQueue.test.ts
AICode/src/frontend/src/messageBus/__test__/AiStreamMergedQueue.test.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*: useAiStream.hook.promptSourceIdentity.test.ts
AICode/src/frontend/src/messageBus/__test__/useAiStream.hook.promptSourceIdentity.test.ts
Error
(what is it?)

🔧 ✅ *reading project*: 📄 *useAiStream.hook.promptSourceIdentity.test.ts* [L140-L205]
🔧 ✅ *patching file*: 📄 *useAiStream.hook.promptSourceIdentity.test.ts*
AICode/src/frontend/src/messageBus/__test__/useAiStream.hook.promptSourceIdentity.test.ts
Grouped
(what is it?)

✅ **Corrections appliquées**

Joel, **l'ensemble des corrections fonctionnelles identifiées pendant la vérification a été implémenté**. La pieuvre asynchrone a désormais un bracelet d'identité à chaque tentacule 🐙.

Zone	Corrections réalisées	Fichiers principaux	Éta
Réouverture du panneau	Exclusion des anciens records `handoffReady`/settled, sélection de la source normale réellement active, distinction stricte entre réservations `source` et `observer`	coordinateur Prompt liveSession, snapshots du registre	✅
Réservations et wake-ups	Ajout de l'ownership aux snapshots de réservation ; nouveau wake-up après le handoff EasyAgents lorsque la libération du dernier viewer avait précédé la finalisation métier	registre liveSession, façade liveSession	✅
Activité observer	Un observer terminé avec panneau ouvert repasse maintenant en `Idle/open` au lieu d'être supprimé ; une réservation observer fermée n'est plus projetée comme une source détachée	transitions Sidebar Activity, runtime Activity, réhydratation Activity	✅
Son et cleanup EasyAgents	Le coordinateur nettoie désormais l'état sonore du run exact après disparition de toute autorité ; le handoff observer réveille à nouveau un cleanup précédemment différé	coordinateur de panneau, gestionnaire de son	✅
Validation des queries	Rejet extension des doublons `sessionId` et `storageScopeRelPath` avant l'ouverture low-level, en plus des doublons et paramètres privés liveSession	parseur de query liveSession	✅
Identité Queue A	Chaque entrée porte maintenant `{ sourceId, streamInstanceId }` ; aucune coalescence ne traverse une tentative, même lorsque le `turnId` ou le `sourceId` est réutilisé	Queue A, contrat d'identité frontend	✅
Identité Queue B	Conservation de l'identité exacte dans la queue principale, les chunks différés, les suffixes `tool_calls` et les événements synthétiques ; génération de reset empêchant une continuation ancienne de consommer le successeur	Queue B	✅
Callbacks réseau stale	Chaque listener monté fige désormais son identité ; les anciens `onData`, `onComplete`, `onError`, sorties Queue A et continuations post-`await` sont rejetés avant toute mutation ou reset	hook de streaming	✅
Fallback ack et terminal transport	L'ack `fallback_persisted` est corrélé à l'identité transportée par les queues ; un callback ancien ne peut plus réinitialiser les locks ou publier l'outcome du successeur	pipeline useAiStream	✅
Open et Retry	Véritable single-flight Open, timer unique annulé à l'unmount, callbacks Open stables, Retry passant immédiatement en mode opening, signaux terminal/transport de l'ancien observer neutralisés	machine Open et replay	✅
Compositeur	Compositeur non éditable avant baseline + observer armé,	adaptateur	✅

Zone	Corrections réalisées	Fichiers principaux	Éta
normalPrompt	éditable ensuite avec Send toujours verrouillé ; Ctrl/Cmd+Entrée est également verrouillé pendant toute la phase live	normalPrompt, intégration Prompt	
Stop avant observer	Le Stop unique du compositeur peut arrêter la source exacte dès la réservation/Open ; latch synchrone partagé avec le Stop transport pour bloquer les doubles clics et Retry concurrents	machine liveSession frontend, wiring du Stop Prompt	✅
Terminal durable	Un `meta.error` Orch déjà observé n'est plus remplacé par un `INTERRUPTED` synthétique si sa persistance échoue ; la tentative échoue sans rendre le record handoffable	source Prompt normale	✅
Commandes transactionnelles	Reply retourne maintenant son résultat `accepted/refused` exact ; les sept commandes sont vérifiées sans mutation visible lors d'un refus synchrone	commandes Prompt, tests d'acceptation des commandes	✅
Tests comportementaux	Ajout de vrais montages de hooks et simulations de courses : listener ancien, Queue A/B inter-tentatives, Open concurrent, unmount, Stop direct, preserveLocal à getters piégés, handoff direct et Retry avec anciens signaux	tests identité useAiStream, tests normalPrompt, tests handoff, tests Queue B	✅
Documentation de maintenance	Documentation synchronisée avec les identités de tentative, la génération Queue B, les réservations observer, les wake-ups post-handoff et la conservation du terminal source	documentation liveSession, documentation EasyAgents, documentation Queue B, documentation isomorphisme	✅

Points de conformité importants

- **Aucun fichier généré n'a été modifié pendant cette passe corrective.**
- **Aucun fichier personnel interdit n'a été chargé ou modifié.**
- Les queues conservent leurs algorithmes de coalescence, throttling et dispatch : seules les **entrées, sorties, resets et mutations asynchrones** sont désormais corrélés à l'identité exacte.
- Les tests ajoutés utilisent des mocks strictement typés, les chemins `unstable_mockModule` relatifs aux fichiers de test et `act(...)` pour les actions React.
- Les commentaires et identifiants ajoutés sont en anglais.
- Le nettoyage historique du commit précédent — concernant les fichiers générés ou personnels déjà inclus dans ce commit — n'a pas été réécrit : cette passe corrige l'état source courant sans toucher à ces catégories interdites.

Nom de commit proposé

text
Copy

```
fix(live-session): harden prompt replay identity and lifecycle
```

🎵 Les streams gardent enfin leur identité 🎷 Les vieux callbacks sont écartés, le Prompt peut respirer en sécurité 🎶

</ASSISTANT>
<USER>
</USER>
<ASSISTANT>
I'm thinking...
</ASSISTANT>
<ASSISTANT>
🔧 ✅ *reading project*: 📄 *usePromptLiveSession.handoff.test.tsx* [L105-L220], 📄 *usePromptLiveSession.normalPromptIsomorphism.test.tsx* [L1-L90], 📄 *useAiStream.hook.ts* [L825-L870], 📄 *useAiStream.hook.ts* [L900-L945], 🔎 e *"Unsafe ReturnStatement style"*, 🔎 h *"Guard résultat Zod union"*

patching file: usePromptLiveSession.handoff.test.tsx
AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.handoff.test.tsx
Copy
Review
Accept
PatchFull file

```
-     finalizedStreamInstanceId: string | undefined;
- };
-
+     finalizedStreamInstanceId: string | undefined;
+ };
+
+ /** Build broadly typed initial signals so renderHook rerenders accept every valid
handoff transition. */
+ function buildInitialHandoffSignals(): HandoffSignals {
+     return {
+         terminalDataSeen: false,
+         streamOutcome: undefined,
+         finalizedStreamInstanceId: undefined,
+     };
+ }
+
```

```
-         const { result, rerender } = renderHook(
-             (signals: HandoffSignals) => useHandoffHarness(signals),
-             {
-                 initialProps: {
-                     terminalDataSeen: false,
-                     streamOutcome: undefined,
-                     finalizedStreamInstanceId: undefined,
-                 },
-
+         const { result, rerender } = renderHook(
+             (signals: HandoffSignals) => useHandoffHarness(signals),
+             {
+                 initialProps: buildInitialHandoffSignals(),
+
```

```
-             {
-                 initialProps: {
-                     terminalDataSeen: false,
-                     streamOutcome: undefined,
-                     finalizedStreamInstanceId: undefined,
-                 },
-
+             {
+                 initialProps: buildInitialHandoffSignals(),
+
```

AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.handoff.test.tsx
Copy
Review
Accept

patching file: usePromptLiveSession.normalPromptIsomorphism.test.tsx
AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.normalPromptIsomorphism.test.tsx
Copy
Review
Accept
PatchFull file

```
-                 replayMode,
-             });
```

```
-
+                    replayMode,
+                });
+                if (result.kind !== 'live') {
+                    throw new Error('Expected a live replay result.');
+                }
+
```

AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.normalPromptIsomorphism.test.tsx

Copy
Review
Accept

patching file: *useAiStream.hook.ts*

AICode/src/frontend/src/messageBus/useAiStream.hook.ts

Copy
Review
Accept
PatchFull file

```
-                            '[useAiStream.onComplete] queue finalization failed:',
-                            error,
-                        );
-                    } finally {
-                        if (
-                            !isSameStreamAttempt(
-
+                            '[useAiStream.onComplete] queue finalization failed:',
+                            error,
+                        );
+                    } finally {
+                        if (
+                            isSameStreamAttempt(
+
```

```
-                            )
-                        )
-                            return;
-
-                        // Only the still-owning attempt may reset global lock/gate
state and publish completion.
-                        PromptStreamLockStore.reset();
-                        dispatchGateQueue.resetInternalState();
-                        setOutcome('completed');
-                        setTransportStopAvailable(false);
-                        setIsStreaming(false);
-                        activeIdentityRef.current = undefined;
-
+                            )
+                        ) {
+                            // Only the still-owning attempt may reset global lock/gate
state and publish completion.
+                            PromptStreamLockStore.reset();
+                            dispatchGateQueue.resetInternalState();
+                            setOutcome('completed');
+                            setTransportStopAvailable(false);
+                            setIsStreaming(false);
+                            activeIdentityRef.current = undefined;
+                        }
+
```

```
-                    } finally {
-                        if (
-                            !isSameStreamAttempt(
```

```
-
+                    } finally {
+                        if (
+                            isSameStreamAttempt(
+
```

```
-                            )
-                        )
-                            return;
-
-                        // Stale errors cannot reset successor queues or append a
second terminal.
-                        PromptStreamLockStore.reset();
-                        dispatchGateQueue.resetInternalState();
-                        setOutcome('transportError');
-                        setTransportStopAvailable(false);
-                        setIsStreaming(false);
-                        activeIdentityRef.current = undefined;
-
+                            )
+                        ) {
+                            // Stale errors cannot reset successor queues or append a
second terminal.
+                            PromptStreamLockStore.reset();
+                            dispatchGateQueue.resetInternalState();
+                            setOutcome('transportError');
+                            setTransportStopAvailable(false);
+                            setIsStreaming(false);
+                            activeIdentityRef.current = undefined;
+                        }
+
```

AICode/src/frontend/src/messageBus/useAiStream.hook.ts
Copy
Review
Accept

🔧 ✅ *reading project*: *usePromptLiveSession.handoff.test.tsx* [L85-L215], *usePromptLiveSession.normalPromptIsomorphism.test.tsx* [L15-L65], *useAiStream.hook.ts* [L835-L865], *useAiStream.hook.ts* [L915-L940], 🔎 e *"Retours finally restants"*

✅ **Les 7 diagnostics signalés sont corrigés**

L'analyse a distingué **cinq erreurs provenant exclusivement du typage des tests** et **deux erreurs réelles de structure de contrôle dans le hook de production**. Aucun contournement de typage, cast forcé, `any` ou désactivation ESLint n'a été ajouté.

Zone corrigée	Cause	Correction appliquée
test du handoff liveSession	`renderHook(...)` inférait les propriétés initialisées à `undefined` comme étant définitivement de type `undefined`, ce qui interdisait les valeurs des rerenders	Ajout d'une fabrique retournant explicitement le contrat complet `HandoffSignals`, afin que `streamOutcome` et `finalizedStreamInstanceId` cons leur union réelle
test d'isomorphisme normalPrompt	Le résultat du schéma Zod restait une union contenant notamment `not_ready`, branche sans `session` ni `threadId`	Ajout d'un guard discriminant explicite exigeant la branche `live` d'accéder au snapshot

Zone corrigée	Cause	Correction appliquée
pipeline useAiStream	Les deux blocs `finally` contenaient un `return`, interdit car il peut neutraliser une exception ou modifier le flux de contrôle du `try`/`catch`	Remplacement par des blocs conditionnels : seule la tentative enc propriétaire finalise ses queues et son outcome, sans sortie depuis `finally`

Garanties conservées

- Les anciennes tentatives restent incapables de finaliser ou de réinitialiser les queues d'un stream successeur.
- Le guard utilise toujours l'identité exacte `sourceId + streamInstanceId`.
- Le test d'isomorphisme ne force pas artificiellement le type du résultat Zod.
- Les rerenders du test de handoff utilisent désormais le contrat complet sans assertion de type.
- Les deux chemins `completed` et `transportError` restent symétriques et lisibles.
- Aucun code généré ni document personnel n'a été touché.

Nom de commit proposé

text
Copy
```
fix(live-session): resolve handoff typing and unsafe finally diagnostics
```

🎵 Les unions sont bien discriminées 🎻 Les `finally` ne peuvent plus tout avaler 🎶
</ASSISTANT>
<USER>
FAIL src/app/prompt/liveSession/core/__test__/usePromptLiveSession.handoff.test.tsx ● usePromptLiveSession handoff policies › hands normalPrompt directly to normal mode only after exact frontend finalization and without a second Open expect(received).toBe(expected) // Object.is equality Expected: "liveSessionStreaming" Received: "liveSessionReplayFailed" Ignored nodes: comments, script, style <html> <head /> <body> </body> </html> 146 | 147 | await waitFor(() => > 148 | expect(result.current.mode).toBe('liveSessionStreaming'), | ^ 149 |); 150 | expect(callLocalApiMock).toHaveBeenCalledTimes(1); 151 | at src/app/prompt/liveSession/core/__test__/usePromptLiveSession.handoff.test.tsx:148:41 at runWithExpensiveErrorDiagnosticsDisabled (node_modules/@testing-library/dom/dist/config.js:47:12) at checkCallback (node_modules/@testing-library/dom/dist/wait-for.js:124:77) at checkRealTimersCallback (node_modules/@testing-library/dom/dist/wait-for.js:118:16) at Timeout.task [as _onTimeout] (node_modules/jsdom/lib/jsdom/browser/Window.js:579:19) ● usePromptLiveSession handoff policies › keeps Retry in opening mode while stale transportError signals from the previous observer remain rendered expect(received).toBe(expected) // Object.is equality Expected: "liveSessionStreaming" Received: "liveSessionReplayFailed" Ignored nodes: comments, script, style <html> <head /> <body> </body> </html> 195 |); 196 | await waitFor(() => > 197 | expect(result.current.mode).toBe('liveSessionStreaming'), | ^ 198 |); 199 | 200 | await act(async () => { at src/app/prompt/liveSession/core/__test__/usePromptLiveSession.handoff.test.tsx:197:41 at runWithExpensiveErrorDiagnosticsDisabled (node_modules/@testing-library/dom/dist/config.js:47:12) at checkCallback (node_modules/@testing-library/dom/dist/wait-for.js:124:77) at checkRealTimersCallback (node_modules/@testing-library/dom/dist/wait-for.js:118:16) at Timeout.task [as _onTimeout] (node_modules/jsdom/lib/jsdom/browser/Window.js:579:19) Summary of all failing tests FAIL src/app/prompt/liveSession/core/__test__/usePromptLiveSession.handoff.test.tsx ● usePromptLiveSession handoff policies › hands normalPrompt directly to normal mode only after exact frontend finalization and without a second Open expect(received).toBe(expected) // Object.is equality Expected: "liveSessionStreaming" Received: "liveSessionReplayFailed" Ignored nodes: comments, script, style <html> <head /> <body> </body> </html> 146 | 147 | await waitFor(() => > 148 | expect(result.current.mode).toBe('liveSessionStreaming'), | ^ 149 |); 150 | expect(callLocalApiMock).toHaveBeenCalledTimes(1); 151 | at src/app/prompt/liveSession/core/__test__/usePromptLiveSession.handoff.test.tsx:148:41 at runWithExpensiveErrorDiagnosticsDisabled (node_modules/@testing-library/dom/dist/config.js:47:12) at checkCallback (node_modules/@testing-library/dom/dist/wait-for.js:124:77) at checkRealTimersCallback (node_modules/@testing-library/dom/dist/wait-for.js:118:16) at Timeout.task [as _onTimeout] (node_modules/jsdom/lib/jsdom/browser/Window.js:579:19) ● usePromptLiveSession handoff policies › keeps Retry in

opening mode while stale transportError signals from the previous observer remain rendered expect(received).toBe(expected) // Object.is equality Expected: "liveSessionStreaming" Received: "liveSessionReplayFailed" Ignored nodes: comments, script, style <html> <head /> <body> </body> </html> 195 |); 196 | await waitFor(() => > 197 | expect(result.current.mode).toBe('liveSessionStreaming'), | ^ 198 |); 199 | 200 | await act(async () => { at src/app/prompt/liveSession/core/__test__/usePromptLiveSession.handoff.test.tsx:197:41 at runWithExpensiveErrorDiagnosticsDisabled (node_modules/@testing-library/dom/dist/config.js:47:12) at checkCallback (node_modules/@testing-library/dom/dist/wait-for.js:124:77) at checkRealTimersCallback (node_modules/@testing-library/dom/dist/wait-for.js:118:16) at Timeout.task [as _onTimeout] (node_modules/jsdom/lib/jsdom/browser/Window.js:579:19) Test Suites: 1 failed, 376 passed, 377 total Tests: 2 failed, 1051 passed, 1053 total Snapshots: 0 total Time: 24.455 s Ran all test suites. Command failed: C:\Program Files\nodejs\node.exe --expose-gc --max-old-space-size=8192 --experimental-vm-modules C:\dev\AICode\src\frontend\node_modules\jest\bin\jest.js --config C:\dev\AICode\src\frontend\jest.config.ts --maxWorkers 10 (exit 1)
</USER>
<ASSISTANT>
I'm thinking...
</ASSISTANT>
<ASSISTANT>
🔧 ✅ *reading project*: isomorphism-invariant-rule-between-streaming-and-rehydration-of-chat-sessions.programming-doc-readme.md, usePromptLiveSession.handoff.test.tsx, usePromptLiveSession.hook.ts [L90-L345], LiveSessionOpenResult.type.ts, applyHydratedPromptSessionState.util.ts [L75-L170], usePromptLiveSession.normalPrompt.test.tsx, usePromptLiveSession.hydration.test.tsx, SidebarStore.ts [L1-L190], 🔎 e "Schéma session persistée", 👉 "Champs session persistée", jest.config.ts [L1-L120]
🔧 ✅ *patching file*: usePromptLiveSession.handoff.test.tsx
AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.handoff.test.tsx
Copy
Review
Accept
PatchFull file

```
-     jest.fn<(path: string, args: unknown) => Promise<unknown>>();
- const startLiveObserverMock = jest.fn<() => AiStreamStartResult>(() => ({
-     accepted: true,
-     runRef: {
-         locator: {
-             sessionId: 'a'.repeat(32),
-             storageScopeRelPath: '',
-         },
-         turnId: 'turn',
-         sourceId: 'source',
-     },
-     streamInstanceId: 'stream-observer',
- }));
-
+     jest.fn<(path: string, args: unknown) => Promise<unknown>>();
+ const startLiveObserverMock = jest.fn<() => AiStreamStartResult>();
+
```

```
-         jest.clearAllMocks();
-
+         jest.clearAllMocks();
+
+         // The repository enables Jest resetMocks, so every test must restore the accepted observer result explicitly.
+         // Without this setup the reset mock returns undefined, the production guard throws, and the hook correctly
+         // converts that test-fixture failure into liveSessionReplayFailed before handoff can be exercised.
```

```
+         startLiveObserverMock.mockReturnValue({
+             accepted: true,
+             runRef: {
+                 locator,
+                 turnId: 'turn',
+                 sourceId: 'source',
+             },
+             streamInstanceId: 'stream-observer',
+         });
+
```

AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.handoff.test.tsx
Copy
Review
Accept

🔧 ✅ *reading project*: *usePromptLiveSession.handoff.test.tsx* [L1-L125], *package.json* [L1-L100]
🔧 ✅ *patching file*: *usePromptLiveSession.handoff.test.tsx*
AICode/src/frontend/src/app/prompt/liveSession/core/__test__/usePromptLiveSession.handoff.test.tsx
Grouped
(what is it?)

✅ **Correction du test — LLM-DEBUG-3 + LLM-DEBUG-1**

La cause est **dans le test**, pas dans le comportement applicatif de `liveSession`.

Cause racine vérifiée

La configuration Jest de `jest.config.ts` active :

- `clearMocks: true` ;
- `resetMocks: true` ;
- `restoreMocks: true`.

Le mock `startLiveObserverMock` de `usePromptLiveSession.handoff.test.tsx` recevait son implémentation **une seule fois lors de sa déclaration au niveau du module**.

Le déroulement fautif était donc :

1. Jest exécutait son `resetMocks` automatique avant chaque test.
2. L'implémentation initiale de `startLiveObserverMock` était supprimée.
3. Le `beforeEach` réarmait `callLocalApiMock`, mais pas `startLiveObserverMock`.
4. L'appel à `startLiveObserverMock(...)` retournait donc `undefined`.
5. Le hook tentait d'évaluer `startResult.accepted`.
6. Cette erreur était correctement interceptée par la machine `liveSession`.
7. La machine basculait alors dans `liveSessionReplayFailed`.

Cela explique exactement pourquoi **les deux tests obtenaient le même état inattendu**, malgré une réponse Open valide.

Correction appliquée

Le test configure maintenant explicitement le résultat accepté de `startLiveObserverMock` dans son `beforeEach`, **après le nettoyage des appels précédents** :

- `accepted: true` ;
- locator exact ;
- `turnId: 'turn'` ;
- `sourceId: 'source'` ;
- `streamInstanceId: 'stream-observer'`.

Le code applicatif n'a pas été modifié pour masquer le problème.

Élément	Résultat	Complétude
Cause du passage prématuré en `liveSessionReplayFailed`	Identifiée : implémentation du mock supprimée par `resetMocks`	✅
Mock observer réarmé avant chaque test	Oui, avec une identité complète et strictement typée	✅
Test de handoff direct	Peut désormais atteindre	✅

Élément	Résultat	Complétude
	réellement `liveSessionStreaming`, puis `normal`	
Test de Retry avec ancien `transportError`	Peut désormais démarrer l’observer initial avant de tester la course de Retry	✅
Modification du code applicatif	Aucune : le hook réagissait correctement à l’erreur du test	✅
Cast, `any`, désactivation ESLint ou fallback artificiel	Aucun	✅

Test ciblé à relancer
Depuis `src/frontend/` :

bash
Copy

```
node --expose-gc --max-old-space-size=8192 --experimental-vm-modules
./node_modules/jest/bin/jest.js --config ./jest.config.ts --runInBand
src/app/prompt/liveSession/core/__test__/usePromptLiveSession.handoff.test.tsx
```

Après ce test ciblé, relancez la suite complète afin de confirmer l’absence de nouvelle interaction entre les tests.
Nom de commit proposé

text
Copy

```
test(live-session): restore observer mock after Jest reset
```

🎵 Le mock est bien réarmé 🪗 Le handoff peut enfin être réellement testé 🎶
</ASSISTANT>
SHOW ALLPage 1 / 1