ÉMERGENCE — Mémoire persistante des agents (RAG)

Problématique & pistes de correction (v1)

ÉMERGENCE — Mémoire persistante des agents (RAG)
Problématique & pistes de correction (version 1)
Date: 2025-08-17 13:14

— ARBO■LOCK —
Référence architecture: arborescence_synchronisée_20250816.txt (aucun mouvement d'arbo dans ce livrable).

1) Contexte
• Projet: ÉMERGENCE (C:\dev\emergenceV8).
• Objectif: obtenir une mémoire inter■sessions pour les agents via RAG (collections Chroma 'emergence_documents' / 'emergence_knowledge').
• Politique actuelle du ChatService: OFF policy: stateless (donc pas de mémoire intrinsèque de session; la mémoire doit venir du RAG).

2) Symptômes observés
• Les agents répondent "à vide" malgré use_rag=True.
• Les requêtes RAG partent bien (collection ouverte + query), mais rien n'est retrouvé.
• Le jardinier /api/memory/tend-garden consolide soit "rien", soit une session différente et vide.
• Aucun log ne montre d'insertion vectorielle (0 concepts/résumés plantés).

3) Faits clés tirés des logs (chronologie)
• Boot OK: DB initialisée, migrations passées, SentenceTransformer chargé, Chroma prêt. Erreurs PostHog = télémétrie, non bloquant.
• 14:59:07 — Session WS '409ad852■…' créée et sauvegardée (enveloppe de session OK).
• 14:59:22 — Message "De quoi a■t■on parlé…" (use_rag=True) → 'emergence_documents' ouverte, query envoyée.
• 14:59:25 — Appel /api/memory/tend-garden → le jardinier traite une AUTRE session 'be252140■…' considérée "vide", puis annonce 0 concept.
• 15:03:36 — Nouvelle ronde: "Aucune session à consolider".
• Vérif DB ad hoc: le script externe a échoué sur finalized_at (colonne absente du schéma réel) → indique divergence de schéma côté inspection, pas forcément côté app.

4) Diagnostic consolidé
• Vectorisation tardive/inexistante: la session active n'est pas jardinée (filtre sur sessions finalisées, ou sélection erronée), et la session 'be252140■…' est vide.
• RAG est donc interrogé avant toute plantation utile → résultats vides et impression d'amnésie.
• Persistance "enveloppe de session" confirmée, mais pas de trace claire d'écriture dans 'session_messages' pour la session active dans ces logs.
• Les erreurs PostHog n'ont pas d'impact sur la mémoire.

5) Causes racines probables (ordonnées)
1. Le jardinier ne sélectionne que les sessions finalisées ou ne cible pas la bonne session (scope/filtre).
2. Les messages de la session active ne sont pas présents au moment du jardinage (timing, transaction ou condition d'écriture par rôle).
3. Le flux RAG est déclenché trop tôt (avant jardinage) et sans fallback sémantique (ex: résumé de session en RAM).
4. Divergence de schéma ou d'hypothèses entre code et outils d'inspection (ex: colonne finalized_at non existante).

6) Impacts
• Mémoire perçue comme inexistante (agents stateless + vecteur vide).
• Tests fonctionnels RAG faussés: query sur collection vide → aucune récupération de contexte.
• Perte de confiance sur la fiabilité des consolidations.

7) Pistes concrètes de correction (alignées à l'architecture actuelle, sans refactor sauvage)
A. Jardinage "au fil de l'eau" (événementiel, debounce)
  • Déclencher l'analyse+consolidation dès qu'un message 'user' ou 'assistant' est persisté, avec un debounce (p.ex. 500–1500 ms) pour grouper plusieurs messages successifs.
  • Cibler explicitement la session active (ID) au lieu d'un balayage global, puis propager vers

‘emergence_documents’ avec métadonnées [user_id, session_id, role].

B. Tâche périodique interne (background task)
• Lancer une ronde toutes les X secondes (ex: 10–30 s) qui inclut les sessions NON finalisées et récentes.
• Critères: sessions updated_at < now – 2s et non consolidées; éviter la même session à répétition via un marqueur “dernier_scan_at”.

C. Finalisation proactive et hooks
• À la déconnexion WS ou après N secondes d’inactivité: finaliser la session puis lancer l’analyse immédiatement (hook on_finalized). Minimiser la fenêtre “RAG vide”.

D. Persistance messages: vérification stricte
• Vérifier que chaque message est effectivement écrit dans ‘session_messages’ (contrôle de retour, logs dédiés, compteur par session).
• S’assurer que les rôles ‘system’/‘tool’ éventuels sont filtrés ou correctement sérialisés selon la politique d’analyse.

E. Sélection de la bonne session pour le jardinier
• Ajuster la requête de sélection: inclure sessions actives/non finalisées modifiées récemment pour la consolidation.
• Exclure explicitement les sessions sans messages (short-circuit clair).

F. RAG: robustesse de la requête et métadonnées
• Indexer systématiquement: [user_id, session_id, timestamp, role, source="summary|concept"].
• Lors de la query, filtrer par user_id et, si pertinent, booster la session la plus récente.
• En cas d’absence d’items, fournir un fallback (ex: réponse courte “pas encore de contenu consolidé” plutôt que silence).

G. Observabilité (indispensable avant livraison)
• Ajouter des logs structurés pour: “message_persisted”, “session_selected_for_gardening”, “analysis_result(count)”, “vectors_inserted(count)”.
• Sanity checks automatiques: comparer le nombre de messages en DB vs. items consolidés; alerter si écart.

H. Gouvernance des états (stateless vs mémoire)
• Assumer le mode stateless du ChatService; la mémoire utilisateur doit être apportée par RAG → renforcer la promesse côté UI (ex: badge “mémoire dispo” quand col.count()>0).

8) Plan d’action rapide (sans changer l’arborescence)
1. Activer un hook post■message (ou tâche périodique) pour consolider la session active récemment modifiée.
2. Garantir que la sélection du jardinier INCLUE les sessions non finalisées avec messages > 0.
3. Écrire des logs explicitant le nombre de messages DB et d’items vectorisés par session, à chaque ronde.
4. Enrichir/filtrer les métadonnées: [user_id, session_id, role].
5. Ajouter un message de fallback UI quand RAG==vide sur une session récente.
6. Scénario de test standardisé: envoyer “Retiens ceci: …”, consolidation (auto), puis “De quoi a■t■on parlé…”, vérifier col.count()>0.
7. Une fois validé localement, commit ciblé (backend/features/memory/*, core/session_manager.py) et reprise des tests Cloud Run.

9) Décisions à figer lors de l’implémentation
• Déclencheur privilégié: “au fil de l’eau” (événementiel) ou “périodique”?
• Fenêtre de debounce et SLA attendu (latence mémoire perçue).
• Métadonnées minimales obligatoires pour la query RAG (user_id obligatoire).

10) Annexe: indicateurs de succès
• Taux de sessions avec au moins 1 item vectorisé dans les 2 s suivant un message > 95%.
• Temps moyen entre message et insertion RAG < 1.5 s.
• Réponses with use_rag=True contenant un extrait indexé dans ≥ 80% des cas pour les sessions > 2 messages.