🏗️ PHASE 2 - ARCHITECTURE MÉMOIRE PERSISTANTE

🎯 MODULES NOUVEAUX À CRÉER

1. 📚 Session Manager (`core/session_manager.py`)

Responsabilité : Gestion lifecycle des sessions + sauvegarde complète

```
class SessionManager:
    def create_session(user_id: str) -> Session
    def save_session_complete(session: Session) -> bool
    def get_session_by_id(session_id: str) -> Session
    def search_sessions_by_date(start_date, end_date) -> List[Session]
    def get_recent_sessions(limit: int = 10) -> List[Session]
```

2. 🔍 Temporal Search (`core/temporal_search.py`)

Responsabilité : Recherche intelligente dans l'historique complet

```
class TemporalSearch:
    def search_by_theme(theme: str, timeframe: str) -> List[SearchResult]
    def search_by_agent(agent: str, query: str) -> List[SearchResult]
    def search_evolution(concept: str, start_date, end_date) -> EvolutionTimeline
    def find_related_discussions(topic: str) -> List[RelatedSession]
```

3. 🧠 Agent Memory (`core/agent_memory.py`)

Responsabilité : Mémoire contextuelle pour chaque agent

```
class AgentMemory:
    def remember_interaction(agent: str, context: dict) -> bool
    def recall_previous_discussions(agent: str, query: str) -> List[Memory]
    def get_agent_history_with_user(agent: str, user_id: str) -> AgentHistory
    def suggest_conversation_continuity(agent: str) -> List[Suggestion]
```

4. 📊 Memory Analytics (`core/memory_analytics.py`)

Responsabilité : Analyse patterns + insights temporels

```
class MemoryAnalytics:
    def detect_thinking_patterns(user_id: str) -> List[Pattern]
    def analyze_concept_evolution(concept: str, user_id: str) -> Evolution
    def generate_monthly_insights(user_id: str, month: str) -> Insights
    def find_cross_domain_connections(user_id: str) -> List[Connection]
```

🗄️ EXTENSIONS DATABASE

Nouvelles Tables SQLite :

```
-- Sessions complètes avec métadonnées
CREATE TABLE sessions (
    id TEXT PRIMARY KEY,
    user_id TEXT NOT NULL,
    start_time TIMESTAMP,
    end_time TIMESTAMP,
    session_data JSON,  -- Export JSON complet
    themes JSON,        -- Tags thématiques auto/manuels
    agents_used JSON,   -- Liste agents utilisés
    total_cost REAL,
    message_count INTEGER,
    modes_used JSON     -- [dialogue, triangle, documents]
);

-- Mémoire agents par utilisateur
CREATE TABLE agent_memories (
    id TEXT PRIMARY KEY,
    agent_name TEXT NOT NULL,
    user_id TEXT NOT NULL,
    memory_type TEXT,   -- [context, pattern, preference]
    memory_data JSON,
    importance_score REAL,
    created_at TIMESTAMP,
    last_accessed TIMESTAMP
);

-- Timeline des concepts/thèmes
CREATE TABLE concept_timeline (
    id TEXT PRIMARY KEY,
    user_id TEXT NOT NULL,
    concept_name TEXT,
    session_id TEXT,
    mentioned_at TIMESTAMP,
    context_snippet TEXT,
    sentiment_score REAL,
```

```
    evolution_stage TEXT
);

-- Patterns de pensée détectés
CREATE TABLE thinking_patterns (
    id TEXT PRIMARY KEY,
    user_id TEXT NOT NULL,
    pattern_type TEXT,
    pattern_data JSON,
    confidence_score REAL,
    first_detected TIMESTAMP,
    last_confirmed TIMESTAMP,
    occurrences INTEGER
);
```

🔗 INTÉGRATION AVEC PHASE 1

Extensions modules existants :

`core/rag_manager.py` V5

- Ajout méthodes recherche temporelle
- Intégration avec Session Manager
- Context enrichi avec mémoire historique

`core/agents.py` V5

- Intégration Agent Memory
- Prompts enrichis avec contexte historique
- Suggestions de continuité conversations

`interface/backend/main.py` V5

- Nouvelles routes `/api/memory/*`
- WebSocket handlers pour timeline
- Export sessions automatique

`interface/frontend/*` V5

- Nouvelle tab "Timeline"
- Recherche temporelle UI
- Visualisation patterns de pensée

🚀 ROADMAP IMPLÉMENTATION

ÉTAPE 1 : Session Manager (Core)

1. Créer `SessionManager` avec sauvegarde JSON enrichie
2. Intégrer au WebSocket pour capture automatique
3. Tests sauvegarde/récupération sessions

ÉTAPE 2 : Temporal Search

1. Créer `TemporalSearch` avec requêtes intelligentes
2. Index FTS5 sur sessions + concepts
3. API endpoints recherche temporelle

ÉTAPE 3 : Agent Memory

1. Créer `AgentMemory` avec context historique
2. Intégrer aux prompts agents pour continuité
3. Suggestions conversation intelligentes

ÉTAPE 4 : Memory Analytics

1. Créer `MemoryAnalytics` avec pattern detection
2. Dashboard insights temporels
3. Visualisations évolution pensée

ÉTAPE 5 : Interface Timeline

1. Tab Timeline avec navigation temporelle
2. Recherche conversationnelle
3. Visualisation patterns + insights

💡 CAS D'USAGE CIBLES VALIDÉS

Recherche temporelle
"ÉMERGENCE, qu'est-ce qu'on a dit sur l'empathie en médecine fin mai ?"
→ TemporalSearch.search_by_theme("empathie médecine", "mai 2025")

Continuité agent
"Anima, tu te souviens de ma réflexion sur X ? J'ai du nouveau..."
→ AgentMemory.recall_previous_discussions("anima", "réflexion X")

Évolution tracking
"Nexus, comment ma pensée sur Y a évolué depuis mars ?"
→ TemporalSearch.search_evolution("Y", "mars 2025", "now")

Pattern insights
"Montre-moi mes patterns de pensée ce mois-ci"
→ MemoryAnalytics.generate_monthly_insights("FG", "juin 2025")

🔧 AVANTAGES ARCHITECTURE

✅ **Modularité** → Chaque composant indépendant et testable ✅ **Rétrocompatibilité** → Phase 1 reste intacte ✅ **Évolutivité** → Facile d'ajouter nouvelles fonctionnalités ✅ **Performance** → Index SQLite + FTS5 pour recherches rapides ✅ **Privacy** → Données locales, contrôle total FG

🎯 MÉTRIQUES SUCCÈS PHASE 2

- [ ] **Sessions sauvegardées** avec métadonnées complètes
- [ ] **Recherche temporelle** < 500ms sur 1000+ sessions
- [ ] **Agents contextualisés** avec mémoire précédente
- [ ] **Timeline navigation** intuitive par date/thème
- [ ] **Pattern detection** automatique concepts récurrents
- [ ] **Insights mensuels** génération automatique
- [ ] **Cross-domain connections** médecine ↔ littérature ↔ philosophie

---

PRÊT POUR L'IMPLÉMENTATION FG ? 🚀

On commence par **ÉTAPE 1 : Session Manager** ?