

CSCI 578 – Team Project Documentation

PocketLLM Portal

Fall 2025 Group 3

1.1 Rationale for Selecting the Architecture

Our team selected the **React + Spring Boot** architecture from one team member's Assignment #3 submission. The selection was made based on the following criteria:

Quality of Architecture

The original design clearly separated concerns between the frontend (React), backend (Spring Boot), and LLM runtime (llama.cpp).

The backend followed a clean MVC structure, and the frontend used modular, reusable components.

Team Familiarity

Several team members had prior exposure to React or Spring Boot, making this architecture the most feasible to implement within four weeks.

Extensibility

The chosen architecture supports straightforward additions such as admin tools, caching system, user management enhancements, and logging.

Documentation & Community Support

Both React and Spring Boot have strong documentation and large ecosystem support, reducing friction in development.

Model Quality

The chosen design included clear UML diagrams, defined interfaces, and a maintainable structure suitable for iterative development.

1.2 Selected Capability Subset for Implementation

We selected a meaningful subset of features from the original model that provide end-to-end functionality while remaining feasible within the 4-week timeline.

Implemented Features

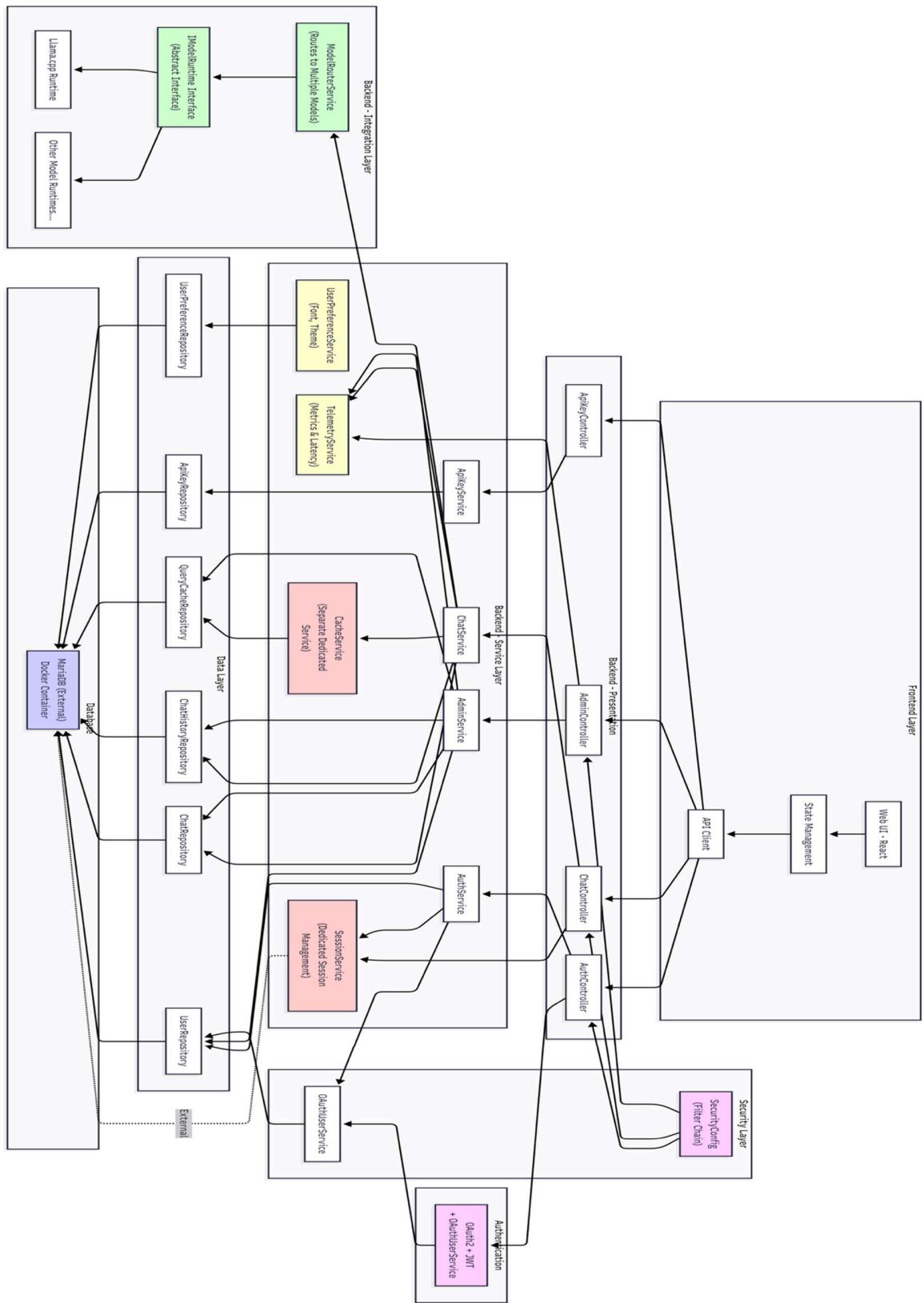
Feature	Priority	Status	Description
User Authentication	High	Implemented	Signup, login, logout, session validation
Chat Management	High	Implemented	Create chats, view list, load history, delete chats
Local LLM Integration	High	Implemented	Phi-3-mini-4k model via llama.cpp
Query Caching	High	Implemented	Cache normalized queries; track hit counts and timestamps
Admin Dashboard	Medium	Implemented	System metrics, cache analytics, chat overview
Admin Statistics	Medium	Implemented	Total users, messages, conversations, cache entries
Admin Chat Controls	Medium	Implemented	View all chats, delete chats
Admin Cache Controls	Medium	Implemented	Top cached queries, clear-all-cache

Deferred / Not Implemented

Feature	Reason for Deferral
OAuth2 Authentication	Required external provider setup; too heavy for timeframe
User Preferences	Non-essential to core workflow
Full telemetry/metrics	Time constraints; partial metrics implemented (Spring Boot Actuator)

1.3 Prescriptive Architecture (Updated)

1.3.1 Component Diagram (Prescriptive Architecture)



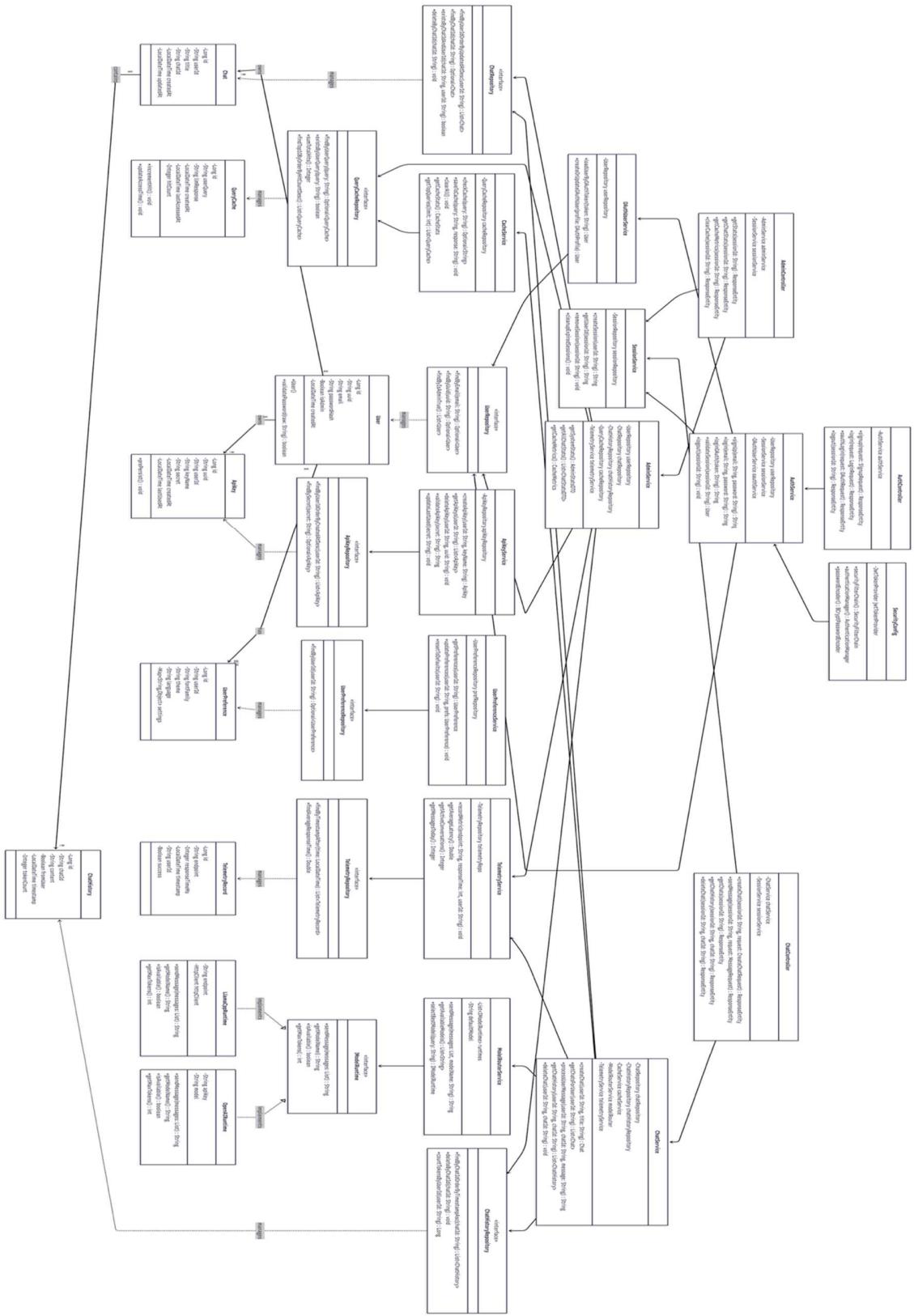
Represents four major deployable components:

- React Frontend
- Spring Boot Backend
- LLM Server (llama.cpp)
- Embedded H2 Database

Communication paths:

- Frontend → Backend REST API
- Backend → LLM Server (HTTP)
- Backend → Database (JPA / H2)
- Frontend → Admin routes for dashboards

1.3.2 Class Diagram (Backend – Implemented)



Key classes and relationships:

Entities: User, Chat, ChatHistory, QueryCache

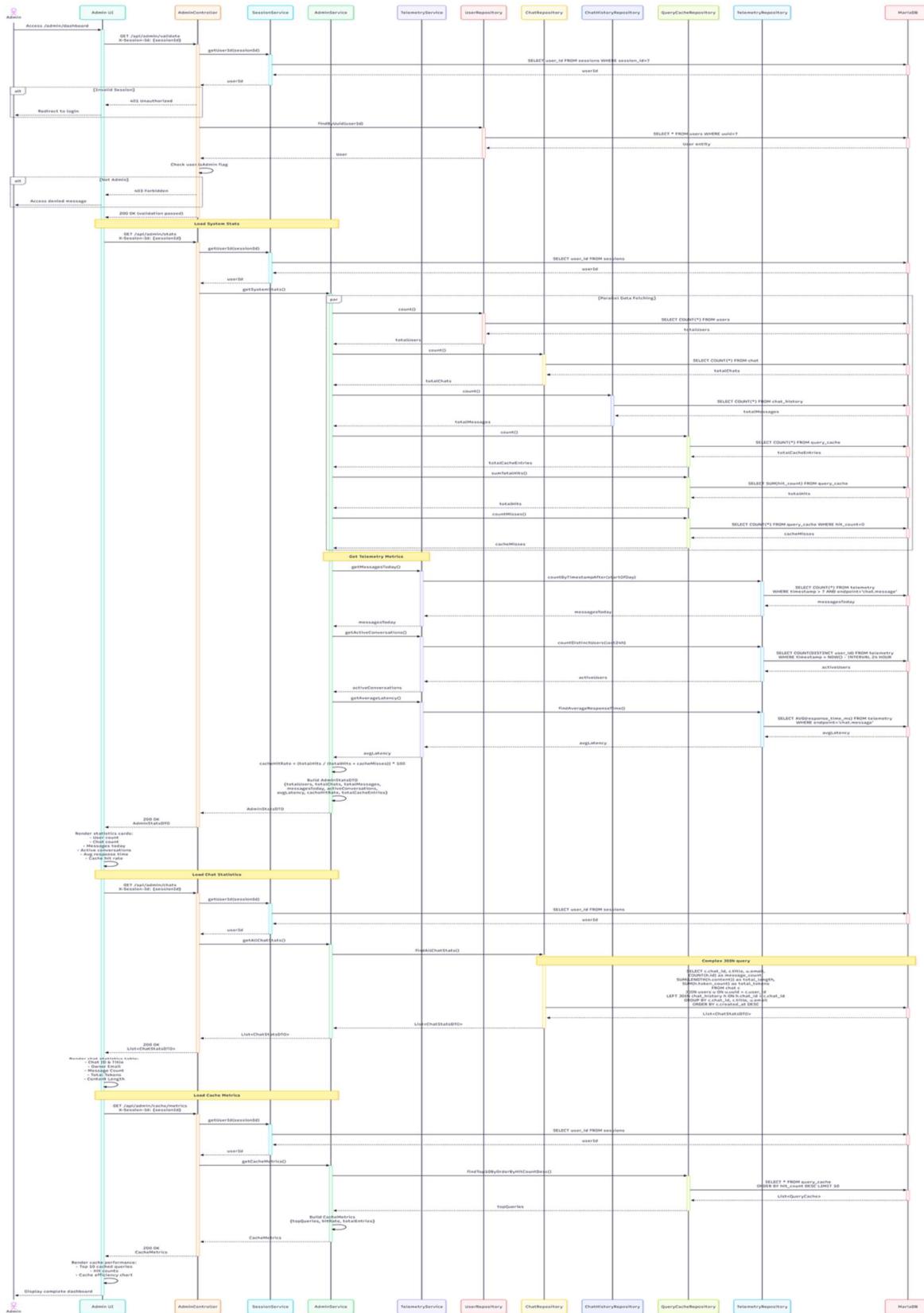
Repositories: UserRepository, ChatRepository, ChatHistoryRepository, QueryCacheRepository

Services: AuthService, ChatService, AdminService

Controllers: AuthController, ChatController, AdminController

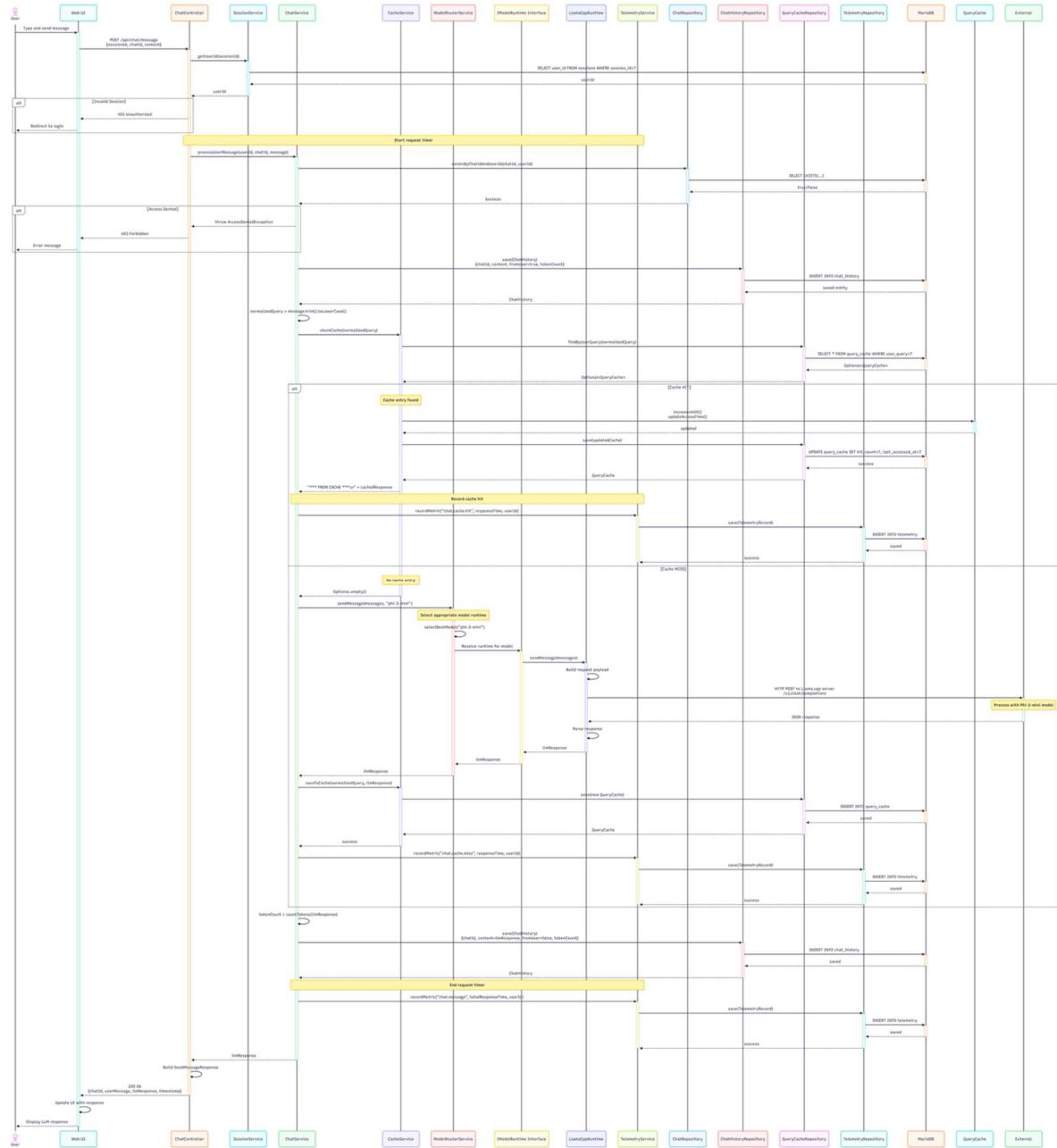
LLM Client: LlmClient

1.3.3 Sequence Diagram—Chat Flow with Caching



1. User sends message
2. Frontend sends POST /chat/message
3. Backend normalizes query
4. Cache lookup
 - o If hit → return cached response
 - o If miss → send request to LLM
5. Save LLM response
6. Return to user

1.3.4 Sequence Diagram—Admin Dashboard Flow



1. Admin logs in
 2. Admin opens /admin
 3. Frontend validates session + isAdmin
 4. Backend aggregates statistics
 5. Backend returns: user counts, chat stats, cache metrics
 6. Admin dashboard displays results

2. Departures from the Architecture (Descriptive vs Prescriptive)

2.1 Comparison Table

Component	Original Design	Actual Implementation	Reason for Change
Database	MariaDB external	Sqlite-JDBC	Simplifies Docker setup; fewer containers
Authentication	OAuth2 + JWT	Session-based	OAuth complexity too high; sessions sufficient
Security	Full Spring Security	Basic CORS + manual session validation	Reduced overhead
CacheService	Standalone service	Integrated into ChatService	Tightly coupled to chat workflow
Model Router	Multiple model routing	Single LlmClient	Only one model deployed
IModel- Runtime In- terface	Multiple runtime options	Removed	Abstraction unnecessary
TelemetryService	Full metrics	Partial metrics in Admin-Service and Spring Boot Actuator	Time limits
UserPreferenceService	Personalization	Not implemented	Low priority
Session- Service	Dedicated module	In-memory SessionStore	Good enough for prototype
Frontend	Centralized API Client and shared component logic	Each page defined in its own page.jsx with isolated logic using simple fetch() calls	Simplifies development; decoupled pages reduce dependency and make debugging easier

2.2 Components Implemented as Designed

- AdminController
- AdminService
- Admin Dashboard UI
- AdminStatsDTO
- ChatStatsDTO

These components match the original architecture with no significant deviation.

2.3 New Components Added (Not in Original Design)

Component	Purpose
QueryCache Entity	Stores normalized queries + responses
QueryCacheRepository	CRUD + analytics queries
Cache hit tracking	hitCount, lastAccessedAt
Session validation endpoint	Used by frontend to guard admin pages
isAdmin field on User	Enables role checking

3. Final Deliverable Summary (Document Portion Only)

Required Deliverable	Status
Updated Architectural Design	Completed
Rationale for selection	Completed
Capability subset definition	Completed
UML descriptions	Completed
Descriptive vs Prescriptive comparison	Completed
List of implemented components	Completed
List of added/changed components	Completed