Design Document for CyCredit.io

COM S 3090 – Fall 2025

Team: 1_Swarna_8

Team Members:
- Mekhi San
- Ash Bhuiyan
- Carson Heyer
- Chase Phelps

Block Diagram

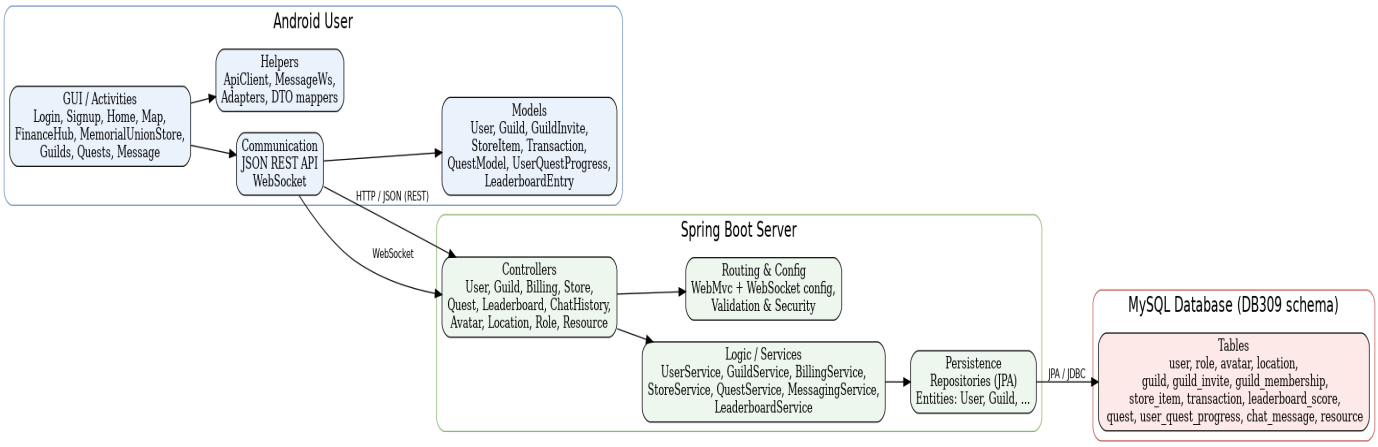

Figure 1. CyCredit.io High-Level Block Diagram (Android Client ↔ Spring Boot Backend ↔ MySQL Database).

Design Description

Overall Architecture

CyCredit.io uses a three-tier architecture with an Android client, a Spring Boot backend, and a MySQL database. The Android client manages all user interactions such as login, signup, navigating the campus map, joining guilds, shopping at the Memorial Union store, tracking finances in the Beardshear Finance Hub, completing quests, and sending messages. The backend exposes REST and WebSocket APIs, enforces game rules, and persists long-term state to a MySQL database.

Android Client

The Android app is organized into activities, network helpers, and data models. Activities such as LoginActivity, SignupActivity, HomeActivity, MapActivity, FinanceHubActivity, MemorialUnionStoreActivity, GuildsActivity, QuestsActivity, AchievementsActivity, MissionsActivity, and MessageActivity are responsible for rendering screens and handling user input. These activities use network classes such as ApiClient and MessageWs to call the backend over HTTP/JSON and WebSockets. Model classes (for example, Guild, GuildInvite, GuildMembership, QuestModel, UserQuestModel, StoreItem, Transaction, and LeaderboardEntry) represent the data exchanged between the app and the server.

Spring Boot Backend

The backend code is organized in the onetoone package with subpackages for users, guilds, billing, store, quests, messaging, leaderboard, avatar, location, roles, and resources. Controllers (UserController, GuildController, BillingController, StoreController, QuestController, ChatHistoryController, LeaderboardController, AvatarController, LocationController, RoleController, and ResourceController) define the public API endpoints. Services encapsulate business logic such as validating purchases, computing credit-like metrics, updating quest progress, awarding achievements, and computing leaderboard rankings. Repository interfaces wrap the JPA entity classes (User, Guild, Transaction, StoreItem, QuestEntity, UserQuestProgressEntity, ChatMessage, LeaderboardScore, etc.) and provide type-safe access to the MySQL database.

Database Layer

The MySQL schema stores all persistent game data. The user table records players and their roles; guild-related tables (guild, guild_invite, guild_membership) manage community relationships; quest tables (quest, user_quest_progress) track mission definitions and each user's progress; billing and transaction tables store purchases and credit-like behavior; store_item defines what can be bought at the Memorial Union store; leaderboard_score and chat_message support leaderboards and chat history. Foreign keys enforce relationships between these tables to keep data consistent.

Interfaces Between Subsystems

The primary interface between the Android client and the backend is an HTTP/JSON REST API, with WebSocket endpoints for real-time messaging and leaderboard updates. The interface between the backend and the database is expressed through JPA repositories using JDBC under the hood. Internally, controllers call services, and services call repositories, which cleanly separates presentation, business logic, and data persistence and makes it easier to test and evolve each part independently.

Database Schema