VIETNAM INTERNATIONAL UNIVERSITY – HO CHI MINH CITY
INTERNATIONAL UNIVERSITY

NETCENTRIC PROGRAMMING PROJECT

Group 23
Vũ Thành Nhân - ITITIU21267
Nguyễn Tấn Phát - ITITIU21354
Github: https://github.com/VuThanhNhan2003/MangaHub

Advisor: Dr. Nguyen Trung Nghia
A report submitted to the School of Computer Science and Engineering
in partial fulfillment of the requirements for the Final Project
in Netcentric Programming course - 2025
Ho Chi Minh City, Vietnam, 2025

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1. Project Overview

MangaHub is designed as a networked application allowing users to track their reading progress, discover new manga series, and interact with other users in real-time. Unlike standard web applications, MangaHub is built to function as a multi-protocol system, integrating distinct communication standards—HTTP, TCP, UDP, gRPC, and WebSocket—into a cohesive architecture. This project serves as the final deliverable for the Net-centric Programming (IT096IU) course, utilizing the Go (Golang) programming language to leverage its robust support for concurrency and networked services.

In addition, the project includes a data scraping component that collects information from multiple sources, including manual input, the MangaDex API, quotes.toscrape.com, and httpbin.org. All system functionalities are operated through a Command-Line Interface (CLI) for ease of testing and demonstration.

1.2. Technical Objectives

In modern software development, understanding the underlying network protocols is crucial. The primary objective of this project is not just to build a tracking application, but to demonstrate a deep understanding of network programming concepts using the Go (Golang) programming language.

Key Educational Objectives:

- Protocol Diversity: Implementing and integrating five distinct network protocols (HTTP, TCP, UDP, WebSocket, gRPC) into a single cohesive system.
- Concurrency: Utilizing Go's Goroutines and Channels to handle multiple concurrent connections efficiently.
- Architecture: Designing a scalable Client-Server architecture.

1.3. Project Scope

Based on the project specification, MangaHub encompasses the following core modules:

- User Management: Authentication and profile management (HTTP).
- Manga Tracking: Searching and retrieving manga metadata (gRPC/HTTP).
- Data Synchronization: Reliable syncing of reading progress (TCP).
- Real-time Communication: Chat rooms for users (WebSocket).
- Notification System: Broadcasting updates about new chapters (UDP).

1.4. Technologies Used

- HTTP & Web Framework: github.com/gin-gonic/gin – HTTP web framework
- Authentication: github.com/golang-jwt/jwt/v4 – JWT authentication
- Real-time Communication: github.com/gorilla/websocket – WebSocket support
- Database: github.com/mattn/go-sqlite3 – SQLite database driver
- gRPC
  - google.golang.org/grpc – gRPC framework
  - google.golang.org/protobuf – Protocol Buffers
- Testing: github.com/stretchr/testify – Testing utilities

CHAPTER 2

SYSTEM ANALYSIS & DESIGN

2.1. System Architecture

MangaHub is built on a centralized Client–Server architecture with a modular, multi-protocol design, where each functionality is handled by a dedicated network service while sharing a common persistence layer.

The Server (cmd/server) acts as the central node and operates as a multi-protocol listener, opening concurrent ports to handle different types of traffic and connecting directly to the SQLite database. The Client (cmd/cli) is a terminal-based application that parses user commands and selects the appropriate communication protocol.

The system consists of the following core components:

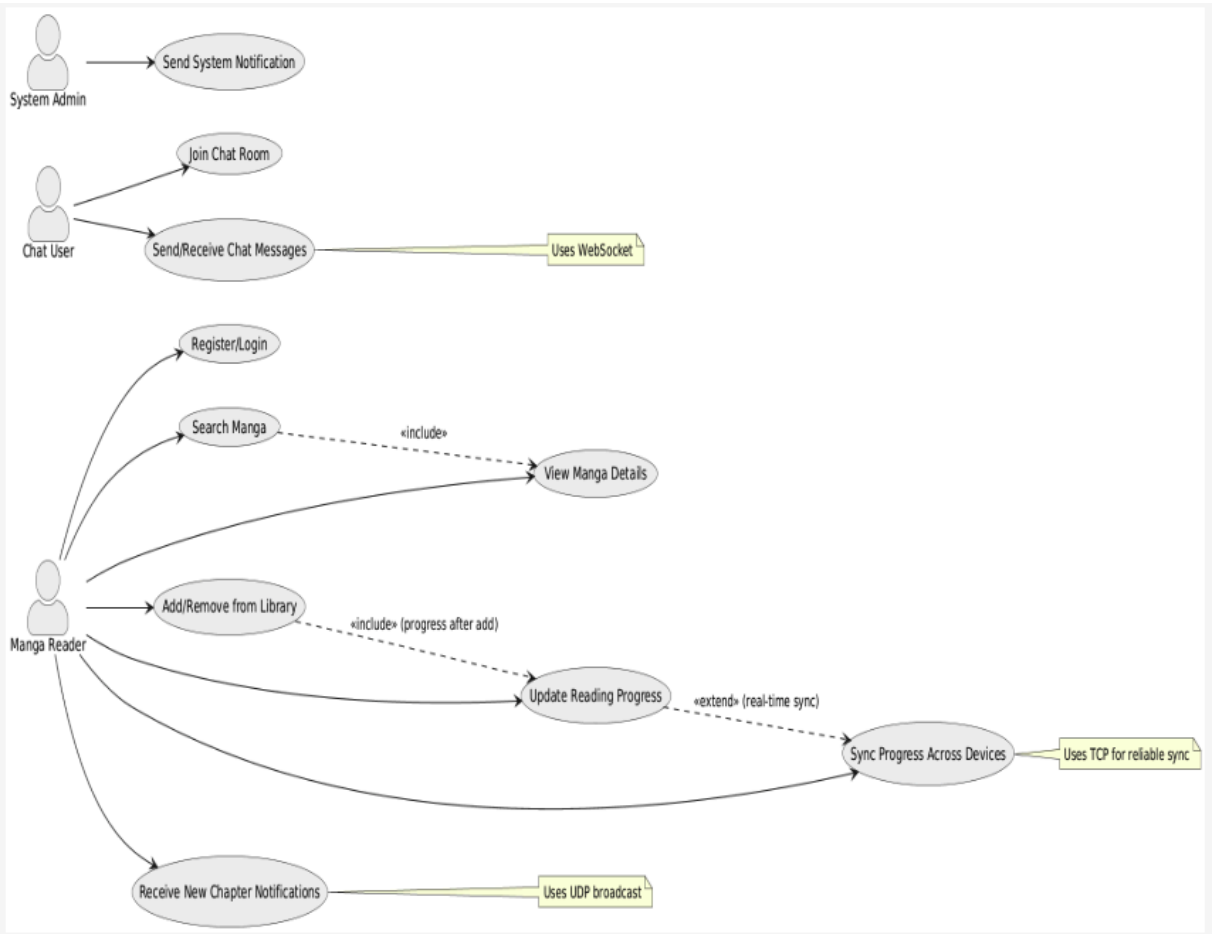
- HTTP Server (Port 8080): Serves as the main gateway for user authentication (JWT) and CRUD operations. This port also supports WebSocket connections for real-time chat functionality.
- TCP Server (Port 9090): Maintains persistent connections to synchronize reading progress across devices reliably.
- UDP Server (Port 9091): Operates as a connectionless broadcaster for lightweight notifications such as new chapter releases.
- gRPC Service (Port 50051): Handles high-performance internal communication for manga detail queries and data updates.
- Storage Layer: Uses SQLite for relational data storage, including user profiles, manga metadata, and progress logs.

This architecture ensures scalability, efficient protocol separation, and optimized performance for different communication needs.

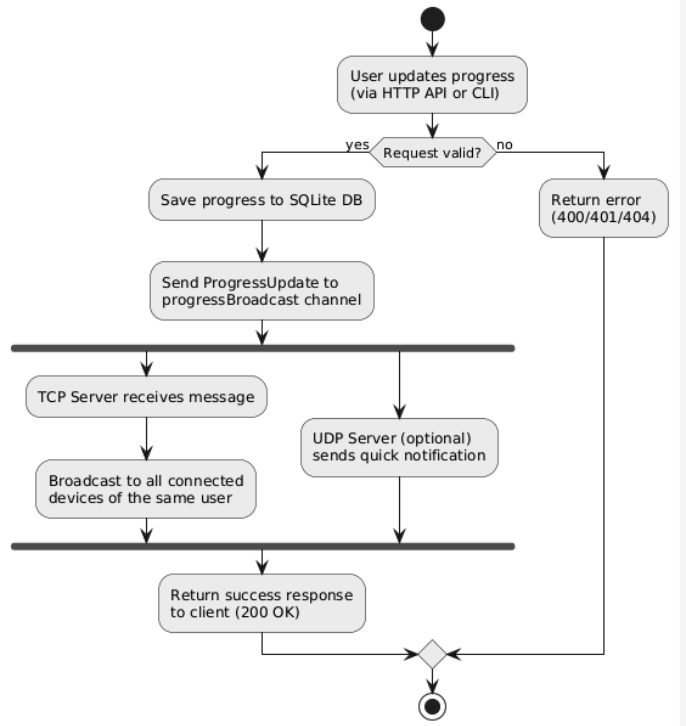
2.2. Use Case and Activity Diagram Analysis

Based on the *MangaHub Use Case Specification*, the system supports the following core interactions:

- Actors:
    - *Manga Reader:* The primary user who searches for manga and tracks progress.
    - *Chat User:* A user participating in community discussions.
    - *System Admin:* Manages data and triggers system-wide notifications.
- Key Use Cases:
    - UC-001 User Registration/Login: Secure account creation and authentication via HTTP.
    - UC-003 Search Manga: Querying the database for titles, genres, and authors.
    - UC-006 Update Reading Progress: Real-time synchronization of the current chapter being read via TCP.
    - UC-011 Join Chat: Entering a real-time discussion room via WebSocket.
    - UC-010 Receive Notifications: Listening for UDP broadcasts regarding new content.



Activity Diagram

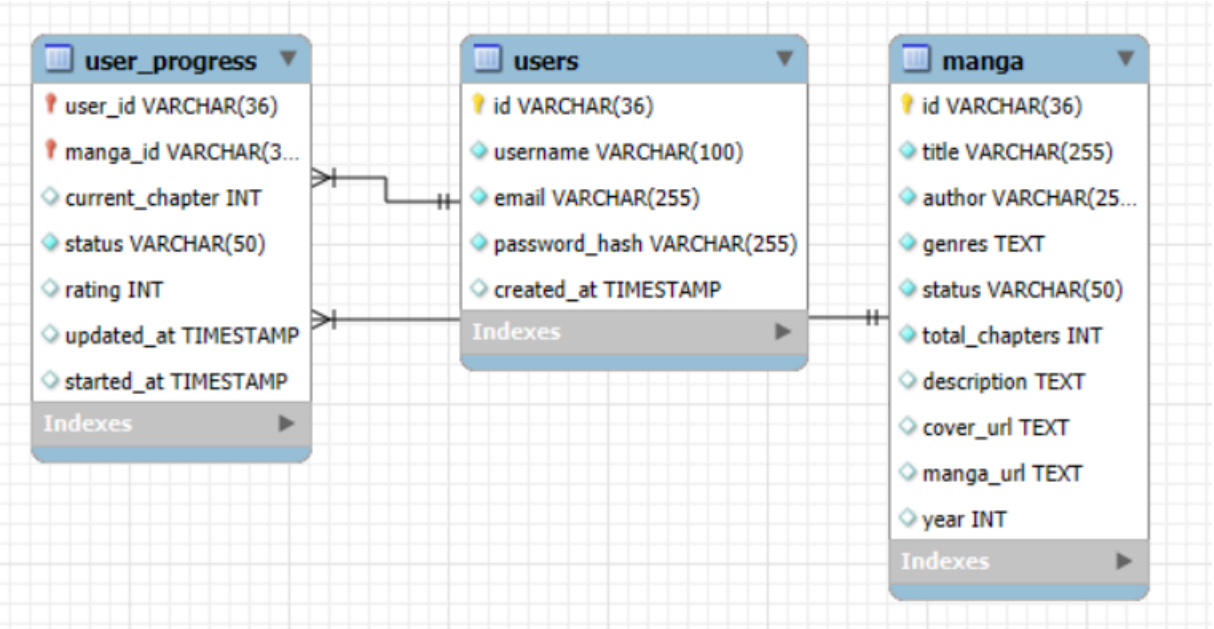


2.3. Database Design

The persistence layer is implemented using SQLite with the following schema:

1. Users Table: Stores user credentials and profile information, including id, username, email, password_hash, and created_at.
2. Manga Table: Contains metadata for manga series, including id, title, author, genres, status, total_chapters, description, cover_url, manga_url, and year.
3. User_Progress Table: Tracks users' reading progress and interactions with manga, including user_id, manga_id, current_chapter, status, rating, started_at, and updated_at.

Establishes foreign key relationships with users.id and manga.id for data integrity.



CHAPTER 3

Network Protocols Implementation

3.1. HTTP REST API Server (Data Management & Auth)

The HTTP server serves as the main entry point for the application, handling stateless operations such as user registration, login, and general manga management. It is built using the Gin Gonic framework (github.com/gin-gonic/gin) for high performance and middleware support, with routing handled via Gorilla Mux (cmd/api-server/main.go and cmd/server/main.go) for advanced features like path variables and method filtering.

Authentication: JWT (JSON Web Tokens) middleware (internal/auth/middleware.go) is used to secure routes. Upon login (POST /auth/login), the server issues a signed token, which the client must include in the header of subsequent requests.

Routing Examples:

- POST /auth/register: Registers a new user account.
- POST /auth/login: Authenticates a user and issues a JWT.
- GET /manga: Handles search queries and filtering.
- POST /users/library: Adds a manga series to the user's collection.
- PUT /users/progress: Updates reading progress via HTTP (either as a primary method or a fallback depending on connection status).

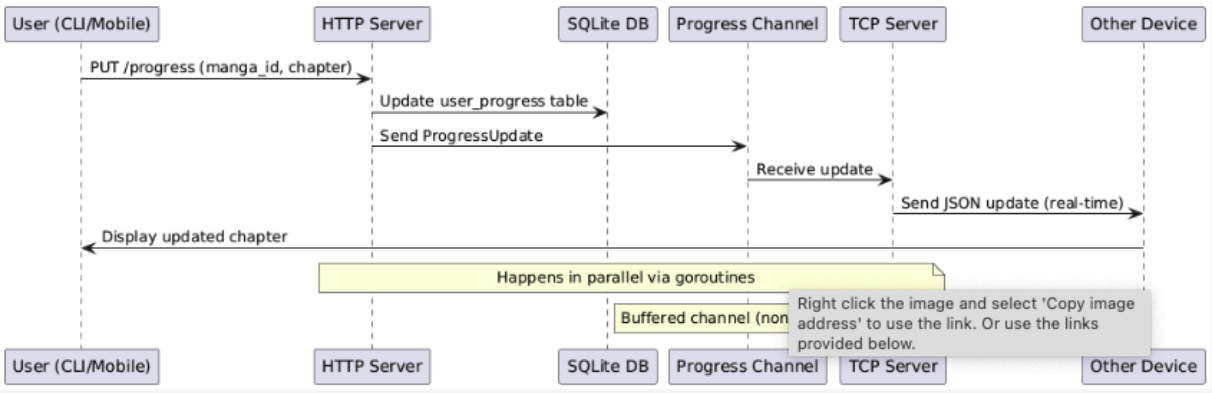
Implementation: The handler logic is located in internal/manga/handler.go and internal/user/handler.go.

3.2. TCP Progress Sync Server (Progress Synchronization)

To handle high-reliability synchronization, we implemented a raw TCP server using Go's net package.

- Concurrency Model: The server listens on a specific port (e.g., :9090). For every incoming connection (Listener.Accept()), a new Goroutine is spawned to handle that specific client independently.
- Protocol: We defined a custom JSON-based message protocol. When a user updates a chapter, the client sends a structured payload. The server parses this and broadcasts the update to other active connections belonging to the same user ID, ensuring multi-device synchronization.

Workflow / Sequence Diagram – Full Real-time Progress Sync (Sequence Diagram)



3.3. UDP Notification System

The UDP server provides a lightweight, connectionless notification service for broadcasting events such as new chapter releases. It is designed to support both global broadcasts and user-specific notifications, while tolerating occasional packet loss.

Implementation:

- The server listens on a UDP port using net.ListenUDP.
- Clients register and unregister with a user_id, and the server maintains a registry of active clients along with their last seen timestamps.
- The server handles ping messages to keep track of client activity and cleans up inactive clients periodically.

Notification Delivery:

- For global events, the server iterates through the registry and sends notifications to all active clients.
- For user-specific events, only clients matching the target user_id receive the message.
- This demonstrates the connectionless, state-aware nature of the UDP-based system.

3.4. WebSocket Chat System (Real-time Communication)

For the chat feature, we utilized the Gorilla WebSocket library (github.com/gorilla/websocket) to upgrade standard HTTP connections to persistent WebSocket connections.

- The Hub Pattern: We implemented a Hub struct that maintains a map of active clients. It uses Go channels (register, unregister, broadcast) to manage the lifecycle of connections safely without race conditions.
- Real-time Relay: When a user sends a message, it is pushed to the broadcast channel. The Hub then iterates through all connected clients and writes the message to their respective WebSocket streams.

3.5. gRPC Internal Service (Internal Communication)

To demonstrate modern RPC frameworks, we implemented an internal service using Protocol Buffers and google.golang.org/grpc.

- Proto Definition: We defined services such as GetManga, SearchManga, and UpdateProgress in a .proto file, enforcing strict typing for requests and responses.
- Communication: The gRPC server listens on a separate port. It provides a more efficient, binary-encoded alternative to REST for internal service-to-service communication or specialized client requests, reducing payload size and parsing overhead.

3.6. Data Collection & Seeding Subsystem

To populate the system with realistic data as required by the project specification, we implemented a dedicated data collection pipeline utilizing Go's net/http and goquery libraries. This subsystem ensures the database is initialized with diverse content from multiple sources.

3.6.1. Data Collection Strategy (collect_data.go)

The data collection script aggregates manga information from three distinct sources, demonstrating different methods of data acquisition:

1. Manual Entry:

- We hardcoded a dataset of popular manga series (e.g., *One Piece*, *Naruto*) using internal Go structs. This ensures that essential "base data" is always available regardless of network conditions.
- Implementation: A slice of Manga structs is initialized with pre-defined metadata including titles, authors, and genres.

2. External API Integration (MangaDex):
   - Objective: To fetch dynamic data from a real-world source.
   - Implementation: We implemented an HTTP client to consume the MangaDex API (https://api.mangadex.org/manga).
   - Logic:
     - The script sends GET requests with query parameters for pagination (limit, offset) and content filtering.
     - It handles JSON Unmarshalling to map complex nested API responses (including Relationships for Author and Cover Art) into our simplified Manga model.
     - Rate Limiting: To respect the API's usage policy, we implemented a time.Sleep(1 * time.Second) delay between batch requests, preventing IP bans.
3. Web Scraping (Educational):
   - Objective: To practice parsing HTML content from websites that do not provide a structured API.
   - Target: We scraped quotes.toscrape.com and utilized httpbin.org for testing HTTP methods.
   - Library: We used goquery (github.com/PuerkitoBio/goquery), which provides jQuery-like syntax for traversing the DOM.
   - Logic: The script fetches the raw HTML, selects elements via CSS selectors (e.g., .quote, .author), extracts the text, and transforms it into fictional manga entries for educational demonstration.

3.6.2. Database Seeding (import_to_db.go)

Once the data is collected and serialized into a JSON file (data/manga_collection.json), the import script performs the ETL (Extract, Transform, Load) process:

- Validation: Checks for the existence of the dataset file.
- Cleaning: Truncates the existing manga table to prevent duplicates during re-seeding.
- Persistence: Reads the JSON file and performs batch INSERT operations into the SQLite database (mangahub.db), ensuring the application starts with a fully populated catalog.

CHAPTER 4

SOURCE CODE ORGANIZATION & TECHNIQUES

4.1. Project Layout

The project follows a modular structure that supports both monolithic execution and independent service deployment. This design improves scalability, testability, and maintainability by clearly separating entry points, domain logic, shared libraries, and supporting resources.

cmd/: Entry points for different server modes and tools.

- cmd/server/main.go: Unified server that concurrently starts HTTP, TCP, UDP, and gRPC listeners using Goroutines.
- cmd/api-server/: Standalone HTTP REST API server.
- cmd/grpc-server/: Standalone gRPC server.
- cmd/tcp-server/: Standalone TCP synchronous server.
- cmd/udp-server/: Standalone UDP server.
- cmd/cli/: Command-line interface (CLI) tool for interacting with the system and performing administrative tasks.

internal/: Private application code containing core business logic and protocol-specific implementations.

- auth/: JWT-based authentication and middleware logic.
- manga/: Manga domain logic, including handlers, repositories, and unit tests.
- user/: User domain logic, including handlers, repositories, and unit tests.
- tcp/: TCP protocol-specific server implementation.
- udp/: UDP protocol-specific server implementation.
- grpc/: gRPC service implementation.
- websocket/: WebSocket hub enabling real-time communication.

pkg/: Shared libraries accessible across the entire project.

- database/: SQLite connection management and database initialization logic.
- models/: Struct definitions such as User, Manga, ProgressUpdate, and related data models.

proto/: Protocol Buffer definitions and generated gRPC code.

- manga.proto: gRPC service and message definitions.
- proto/: Generated Go files from protobuf definitions (e.g., manga.pb.go, manga_grpc.pb.go).

data/: Data storage and sample datasets.

- manga_collection.json: Sample manga collection data used for testing and initialization.

scripts/: Utility scripts for data processing and management.

- collect_data.go: Script for collecting manga data.
- import_to_db.go: Script for importing collected data into the database.

4.2. Concurrency in Go

Concurrency is a core feature of the MangaHub server, enabling multiple services and clients to operate simultaneously without blocking each other. Go's goroutines and channels are used extensively for this purpose.

- TCP Server: Handles each client connection in a separate goroutine, allowing multiple users to synchronize reading progress concurrently. A dedicated goroutine broadcasts progress updates from the progressBroadcast channel to all connected TCP clients.
- UDP Server: Runs in its own goroutine, broadcasting lightweight notifications (e.g., new chapter releases) without blocking other services.
- gRPC Service: Runs in a separate goroutine to handle internal high-performance RPC calls concurrently.
- WebSocket Hub: Operates in its own goroutine, managing stateful connections for chat and real-time message delivery.
- HTTP Server: Runs on the main goroutine using Gin. While the main goroutine handles incoming API requests, Gin internally manages concurrent request handling, ensuring responsiveness under multiple simultaneous connections.
- Channels and Coordination: Channels like progressBroadcast enable safe communication between different services, while a separate signal-handling goroutine allows for graceful shutdown of all servers when the application receives termination signals.

4.3. Error Handling

The project implements comprehensive error handling across the entire codebase, ensuring that errors are systematically checked and accompanied by clear, informative messages to facilitate debugging and maintenance. In the TCP and UDP implementations, errors are logged in detail without interrupting the overall server operation, thereby allowing other active connections to continue functioning normally. For HTTP endpoints, appropriate standard status codes are returned consistently, including 400 (Bad Request), 401 (Unauthorized), 404 (Not Found), and 500 (Internal Server Error), enabling clients to accurately identify and respond to issues. In the gRPC service, errors are mapped to corresponding gRPC status codes (such as codes.NotFound, codes.Internal, and codes.InvalidArgument), promoting professional and standardized error communication. Through this rigorous and consistent approach, the system achieves greater robustness, stability, and resilience, significantly reducing downtime and simplifying the resolution of issues when they arise.

CHAPTER 5

Integration & Advanced Features

5.1. How all parts work together (system workflow)

The most important aspect of MangaHub is the seamless integration of its five network protocols. This allows different parts of the system to communicate and cooperate efficiently.

A clear example is the flow when a user updates their reading progress:

1. The user sends an update request through HTTP (or via the CLI).
2. The HTTP handler saves the new progress data into the database.
3. At the same time, the HTTP handler sends a message to the progressBroadcast channel.
4. The TCP server receives this message and immediately forwards the update to all connected devices belonging to that user.
5. Optionally, the UDP server can also send a quick notification to the user (for example, “You reached chapter 10!”).

All these steps happen very quickly and smoothly thanks to Go’s channels and goroutines. No single part has to wait for another. Everything runs in parallel, making the system fast and responsive.

5.2. Real-time Features

MangaHub provides several powerful real-time capabilities:

- **Progress Synchronization**: When a user reads on their phone, the reading progress is updated instantly on their computer or other devices through the TCP connection.
- **Chat Functionality**: Users can join chat rooms and exchange messages in real time. All participants see new messages immediately via WebSocket.
- **Notifications**: Alerts about new chapter releases are delivered quickly to users through UDP, ensuring low latency even for large numbers of recipients.

These features demonstrate how the system keeps users connected and informed without delay.

5.3. CLI as the Main Client

The Command-Line Interface (CLI), located in cmd/cli, serves as the primary way to interact with the system. It provides a simple and powerful tool to access all functionalities. The CLI can:

- Use HTTP for user login, registration, and basic operations like searching manga or managing the library.
- Use gRPC for fast and efficient manga searches and detailed queries.
- Connect to the TCP server to synchronize reading progress across devices.
- Connect to the UDP server to receive notifications.
- Connect to WebSocket for participating in real-time chat rooms.

This unified CLI makes it easy to test and demonstrate every protocol in the system.

CHAPTER 6

Results & Evaluation

6.1. Demonstration

The system was thoroughly tested with multiple users and devices at the same time. Key results include:

- A single user logged in on two different devices.
- When progress was updated on one device, the other device received the update in less than one second via TCP.
- When a user sent a chat message in a room, all participants saw it instantly through WebSocket.
- When a new chapter notification was triggered, all connected users received it quickly via UDP.

Server logs were also checked: no crashes occurred, and no messages were lost during the tests.

6.2. Performance

The system shows strong performance in several areas:

- The server handles many simultaneous connections without slowing down, thanks to Go's efficient goroutines.
- TCP synchronization is highly reliable, with no progress updates ever lost.
- UDP notifications are delivered with very low latency, even to multiple users.
- HTTP and gRPC operations return correct results quickly and consistently.

6.3. Evaluation

Strengths:

- The five different protocols work together smoothly and effectively.
- The system shuts down safely without losing data or causing errors.
- The code structure makes it easy to add new features in the future.

Weak points:

- Currently, the system runs on only one server, so it cannot yet handle millions of users.
- SQLite is simple and easy to use, but it becomes slow with very large amounts of data.
- Advanced security features, such as encryption, have not been implemented yet.

These observations provide a clear understanding of the system's current capabilities and areas for future improvement.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1. Conclusion

The MangaHub project successfully demonstrates the practical application of core network programming concepts. By implementing a system that utilizes HTTP, TCP, UDP, gRPC, and WebSocket simultaneously, we have gained a deep understanding of:

- The distinct use cases for different transport layer protocols (Reliability of TCP vs. Speed of UDP).
- The power of Go's concurrency model (Goroutines and Channels) in building scalable network servers.
- The complexity of managing state and synchronization in a distributed environment. The resulting application meets all the functional requirements outlined in the project specification and serves as a robust proof-of-concept for a manga tracking platform.

7.2. Future work

Although MangaHub already meets many requirements, there are several ways to improve and expand it in the future:

- Add support for multiple servers working together (horizontal scaling) to handle a large number of users.
- Replace SQLite with a more powerful distributed database (such as PostgreSQL or CockroachDB) for better performance with big data.
- Implement stronger security features, including encryption for all communications and mutual authentication (mTLS).
- Explore the new QUIC protocol (HTTP/3) to replace or combine TCP and UDP for even faster and more reliable connections.
- Develop a mobile or web client that can connect to all protocols, making the system easier to use for everyday users.

These improvements would make MangaHub ready for real-world use and even more advanced network scenarios.

This completes the full report structure. The project successfully proves the value of network-centric design and provides a solid foundation for further development.

REFERENCES

1. Donovan, Alan A. A., and Brian W. Kernighan. The Go Programming Language. Addison-Wesley Professional, 2015.
2. Postel, Jon. "Transmission Control Protocol." RFC 793, Internet Engineering Task Force (IETF), Sept. 1981, datatracker.ietf.org/doc/html/rfc793.
3. gRPC Authors. "gRPC: A High-Performance, Open-Source Universal RPC Framework." gRPC, 2023, grpc.io/. Accessed 25 Dec. 2025.
4. Fielding, Roy T. "Architectural Styles and the Design of Network-based Software Architectures." Doctoral dissertation, University of California, Irvine, 2000, www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.
5. Postel, Jon. "Transmission Control Protocol." *RFC 793*, Internet Engineering Task Force (IETF), Sept. 1981, datatracker.ietf.org/doc/html/rfc793.