MangaHub - Manga & Comic Tracking System

Network Programming (Net Centric Programming) – IT096IU

Term Project Description - Revised Scope

Instructor: Lê Thanh Sơn - Nguyễn Trung Nghĩa
Course: Net-centric Programming - IT096IU
Programming Language: Go
Team Size: 2 students per group
Timeline: 10-11 weeks

Objectives

- To allow students to gain practical experience of network application development using Go programming language with realistic scope
- To allow students to experience all five required communication protocols (TCP, UDP, HTTP, gRPC, WebSocket) through hands-on implementation
- To strengthen understanding of networking concepts through manageable, progressive implementation
- To develop foundational skills in concurrent programming and basic distributed system patterns
- To create a working system that demonstrates network programming competency within academic constraints

Project Deliverables

- **Source code and documentation** must be submitted on Blackboard before due date. Zip all your files and name it GroupXX_MangaHub.zip (ex: Group01_MangaHub.zip)
- **A demonstration session** will be held at the end of the course showing all five protocols working together
- **Fail to show up** during the demonstration session will result in **ZERO grading** for project

Due Date

- **Final Submission:** 23:59 on demo day
- **Demo Session:** Will be announced later

Project Task: MangaHub - Manga Tracking System

You will build **MangaHub**, a manga tracking system that demonstrates network programming concepts through practical implementation. The system will use all five required protocols in a cohesive application while maintaining realistic scope for 11-week development by student teams.

Programming Language Requirements: Go. You must implement TCP, UDP, HTTP, gRPC, and WebSocket communication.

Manga Database

Data Collection Requirements

Build a basic manga database through manageable data collection:

- **Manual Entry:** 100 popular manga series with essential metadata
- **Simple API Integration:**100 additional series from MangaDx API (or others legal API)using basic calls
- **Educational Practice:** Limited web scraping from practice sites (quotes.toscrape.com, httpbin.org)
- **JSON Storage:** Store data in JSON format for simplicity

The manga database must include: - At least **30-40 different manga series** across major genres

- At least **15-20 series per major genre** (shounen, shoujo, seinen, josei, etc.)
- Basic metadata: title, author, genres, status, chapter count, description

Simplified Data Structure

```
{
  "id": "one-piece",
  "title": "One Piece",
  "author": "Oda Eiichiro",
  "genres": ["Action", "Adventure", "Shounen"],
  "status": "ongoing",
  "total_chapters": 1100,
  "description": "A young pirate's adventure...",
  "cover_url": "https://example.com/covers/one-piece.jpg"
}
```

User Data Management

```
{
  "user_id": "user123",
```

```json
  "username": "manga_lover",
  "reading_lists": {
    "reading": [
      {
        "manga_id": "one-piece",
        "current_chapter": 1095,
        "status": "reading",
        "last_updated": "2024-01-20T10:30:00Z"
      }
    ],
    "completed": [],
    "plan_to_read": []
  }
}
```

System Architecture

Core Components

*1. **HTTP REST API Server** (25 points)*

Basic RESTful service with essential endpoints:

```go
// Core server structure
type APIServer struct {
    Router     *gin.Engine
    Database   *sql.DB
    JWTSecret  string
}
```

Essential Endpoints: - POST /auth/register - User registration

- POST /auth/login - User authentication
- GET /manga - Search manga with basic filters
- GET /manga/{id} - Get manga details
- POST /users/library - Add manga to library
- GET /users/library - Get user’s library
- PUT /users/progress - Update reading progress

Requirements: - JWT-based authentication

- SQLite database integration
- JSON request/response handling
- Basic error handling and logging

- Input validation

*2. **TCP Progress Sync Server** (20 points)*

Simple TCP server for basic progress broadcasting:

```go
// Basic TCP server
type ProgressSyncServer struct {
    Port        string
    Connections map[string]net.Conn
    Broadcast   chan ProgressUpdate
}

type ProgressUpdate struct {
    UserID      string `json:"user_id"`
    MangaID     string `json:"manga_id"`
    Chapter     int    `json:"chapter"`
    Timestamp   int64  `json:"timestamp"`
}
```

Requirements: - Accept multiple TCP connections

- Broadcast progress updates to connected clients
- Handle client connections and disconnections
- Basic JSON message protocol
- Simple concurrent connection handling with goroutines

*3. **UDP Notification System** (15 points)*

Basic UDP broadcaster for chapter notifications:

```go
// Simple UDP notifier
type NotificationServer struct {
    Port      string
    Clients   []net.UDPAddr
}

type Notification struct {
    Type      string `json:"type"`
    MangaID   string `json:"manga_id"`
    Message   string `json:"message"`
    Timestamp int64  `json:"timestamp"`
}
```

Requirements: - UDP server listening for client registrations

- Broadcast chapter release notifications

- Handle client list management
- Basic error logging

*4. **WebSocket Chat System** (15 points)*

Simple real-time chat for manga discussions:

```go
// Basic WebSocket hub
type ChatHub struct {
    Clients    map[*websocket.Conn]string
    Broadcast  chan ChatMessage
    Register   chan ClientConnection
    Unregister chan *websocket.Conn
}

type ChatMessage struct {
    UserID    string `json:"user_id"`
    Username  string `json:"username"`
    Message   string `json:"message"`
    Timestamp int64  `json:"timestamp"`
}
```

Requirements: - WebSocket connection handling

- Real-time message broadcasting
- User join/leave functionality
- Basic connection management

*5. **gRPC Internal Service** (10 points)*

Simple gRPC service for internal communication:

```
service MangaService {
  rpc GetManga(GetMangaRequest) returns (MangaResponse);
  rpc SearchManga(SearchRequest) returns (SearchResponse);
  rpc UpdateProgress(ProgressRequest) returns (ProgressResponse);
}
```

Requirements: - Protocol Buffer definitions for 2-3 services

- Basic gRPC server implementation
- Simple client integration
- Unary RPC calls

6. ***Database Layer*** *(10 points)*

```sql
-- Simplified schema
CREATE TABLE users (
    id TEXT PRIMARY KEY,
    username TEXT UNIQUE,
    password_hash TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE manga (
    id TEXT PRIMARY KEY,
    title TEXT,
    author TEXT,
    genres TEXT, -- JSON array as text
    status TEXT,
    total_chapters INTEGER,
    description TEXT
);

CREATE TABLE user_progress (
    user_id TEXT,
    manga_id TEXT,
    current_chapter INTEGER,
    status TEXT,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (user_id, manga_id)
);
```

Network Communication Requirements

Protocol Implementation Expectations

1. **HTTP Services**
    - RESTful API with proper HTTP methods and status codes
    - Basic JWT authentication
    - Error handling with appropriate HTTP responses
    - Simple CORS support for web clients
2. **TCP Socket Communication**
    - Basic server accepting multiple connections
    - JSON-based message protocol
    - Concurrent connection handling with goroutines
    - Graceful connection termination

3. **UDP Broadcasting**
    - Simple UDP server for notifications
    - Client registration mechanism
    - Basic broadcast functionality
    - Error handling for network failures
4. **gRPC Services**
    - Protocol Buffer message definitions
    - Basic service
    - Client-server communication
    - Simple error handling
5. **WebSocket Connections**
    - WebSocket upgrade handling
    - Real-time message broadcasting
    - Connection lifecycle management
    - Basic client management

Performance Requirements

Scalability Targets

- Support **50-100 concurrent users** during testing
- Handle **30-40 manga series** in database
- Process basic search queries within **500ms**
- Support **20-30 concurrent TCP connections**
- WebSocket chat with **10-20 simultaneous users**

Reliability Standards

- **80-90% uptime** during demonstration period
- Basic error handling and recovery
- Simple logging for debugging
- Graceful degradation when services are unavailable

Development Timeline (10 Weeks) - JUST FOR HINT

Phase 1: Foundation (Weeks 1-2)

Week 1: Project Setup & HTTP Basics - Go project structure setup

- Basic HTTP server with Gin framework
- User registration and login endpoints
- SQLite database setup and basic schema

Week 2: Core HTTP API - Manga data model and CRUD endpoints

- User library management endpoints
- Basic JWT authentication middleware
- API testing and validation

Week 3: Data Collection & Integration - Manual manga data entry (20-30 series)

- Simple MangaDx API integration
- Data validation and storage
- API endpoint completion

Phase 2: Network Protocols (Weeks 3-7)

Week 3 TCP Implementation - Basic TCP server setup

- Connection handling with goroutines
- Simple message protocol design
- Progress update broadcasting

Week 4: TCP Integration & Testing - Connect TCP server to HTTP API

- Client connection testing
- Error handling and logging
- Integration with user progress system

Week 5: UDP Notification System - UDP server implementation

- Client registration mechanism
- Basic notification broadcasting

- Integration testing

Week 6: WebSocket Chat - WebSocket server setup

- Basic chat functionality
- Connection management
- Real-time message broadcasting

Week 7: gRPC Service - Protocol Buffer definitions

- Basic gRPC service implementation
- Client integration
- Service testing

Phase 3: Integration & Testing (Weeks 8-10)

Week 8: System Integration - Connect all protocols together

- End-to-end testing
- Bug fixes and stability improvements

Week 9: User Interface & Documentation - Simple web interface (optional)

- API documentation
- Code documentation and comments

Phase 4: Demo Preparation (Week 10)

Week 10: Demo & Presentation

- Demo script preparation
- Live demonstration practice
- Final code review and cleanup

Implementation Guidelines

Recommended Go Libraries

Core Framework: - `github.com/gin-gonic/gin` - HTTP web framework - `github.com/golang-jwt/jwt/v4` - JWT authentication - `github.com/gorilla/websocket` - WebSocket support - `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

Project Structure

```
mangahub/
├── cmd/
│   ├── api-server/main.go        # HTTP API server
│   ├── tcp-server/main.go        # TCP sync server
│   ├── udp-server/main.go        # UDP notification server
│   └── grpc-server/main.go       # gRPC service server
├── internal/
│   ├── auth/                     # Authentication logic
│   ├── manga/                    # Manga data management
│   ├── user/                     # User management
│   ├── tcp/                      # TCP server implementation
│   ├── udp/                      # UDP server implementation
│   ├── websocket/                # WebSocket chat implementation
│   └── grpc/                     # gRPC service implementation
├── pkg/
│   ├── models/                   # Data structures
│   ├── database/                 # Database utilities
│   └── utils/                    # Helper functions
├── proto/                        # Protocol Buffer definitions
├── data/                         # JSON data files
├── docs/                         # Documentation
├── docker-compose.yml            # Development environment
└── README.md                     # Setup instructions
```

Grading Criteria

Total: 30% of course grade (100-point scale)

Core Protocol Implementation (40 points)

- **HTTP REST API (15 pts):** Complete endpoints with authentication and database integration
- **TCP Progress Sync (13 pts):** Working server with concurrent connections and broadcasting
- **UDP Notifications (18 pts):** Basic notification system with client management
- **WebSocket Chat (10 pts):** Real-time messaging with connection handling
- **gRPC Service (7 pts):** Basic service with 2-3 working methods

System Integration & Architecture (20 points)

- **Database Integration (8 pts):** Working data persistence with proper schema and relationships
- **Service Communication (7 pts):** All protocols integrated and working together seamlessly
- **Error Handling & Logging (3 pts):** Comprehensive error handling across all components
- **Code Structure & Organization (2 pts):** Proper Go project organization and modularity

Code Quality & Testing (10 points)

- **Go Code Quality (5 pts):** Proper Go idioms, error handling patterns, and concurrent programming
- **Testing Coverage (3 pts):** Unit tests for core functionality and integration tests
- **Code Documentation (2 pts):** Clear comments and function documentation

Documentation & Demo (10 points)

- **Technical Documentation (5 pts):** API documentation, setup instructions, and architecture overview
- **Live Demonstration (5 pts):** Successfully demonstrate all five protocols working with Q&A

Completed one or more random Bonus features to full fill 10 points

Bonus Features (Extra Credit)

Advanced Protocol Features (5-10 points)

- **Enhanced TCP Synchronization (10 pts):** Implement basic conflict resolution for concurrent updates

  ```go
  type ConflictResolution struct {
      Strategy    string // "last_write_wins", "merge", "user_choice"
      Timestamp   int64
      DeviceID    string
      Resolution  string
  }
  ```

- **WebSocket Room Management (10 pts):** Multiple chat rooms for different manga discussions

```go
type ChatRoom struct {
    ID           string
    MangaID      string
    Participants map[string]*websocket.Conn
    Messages     []ChatMessage
}
```

- **UDP Delivery Confirmation (5 pts):** Implement acknowledgment system for reliable notifications
- **gRPC Streaming (10 pts):** Add server-side streaming for real-time updates

Enhanced Data Management (5-10 points)

- **Advanced Search & Filtering (5 pts):** Implement full-text search with multiple filters

```go
type SearchFilters struct {
    Genres       []string
    Status       string
    YearRange    [2]int
    Rating       float64
    SortBy       string // "popularity", "rating", "recent"
}
```

- **Data Caching with Redis (10 pts):** Implement Redis for frequently accessed data
- **Recommendation System (10 pts):** Basic collaborative filtering based on user reading patterns

```go
type RecommendationEngine struct {
    UserSimilarity  map[string]float64
    MangaSimilarity map[string][]string
    UserProfiles    map[string]UserProfile
}
```

Social & Community Features (5-10 points)

- **User Reviews & Ratings (8 pts):** Allow users to review and rate manga

```go
type Review struct {
    UserID    string
    MangaID   string
    Rating    int    // 1-10
    Text      string
    Timestamp int64
    Helpful   int    // helpful votes
}
```

- **Friend System (5 pts):** Add/remove friends and view friends' reading activity

- **Reading Lists Sharing (6 pts):** Share reading lists with other users
- **Activity Feed (7 pts):** Show recent activities from friends (completed manga, reviews)

Performance & Scalability (5-10 points)

- **Connection Pooling (6 pts):** Implement proper connection pooling for database and external APIs
- **Rate Limiting (5 pts):** Implement rate limiting for API endpoints to prevent abuse
- **Horizontal Scaling (8 pts):** Design system to support multiple server instances
- **Performance Monitoring (7 pts):** Add metrics collection and basic monitoring dashboard
- **Load Balancing (10 pts):** Implement load balancing for multiple service instances

Advanced User Features (5-12 points)

- **Reading Statistics (8 pts):** Detailed reading analytics and progress tracking

```go
type ReadingStats struct {
    TotalChaptersRead int
    ReadingTimeMinutes int
    FavoriteGenres    []string
    ReadingStreak     int
    MonthlyGoals      map[string]int
}
```

- **Notification Preferences (5 pts):** Customizable notification settings per user
- **Reading Goals & Achievements (10 pts):** Set reading goals and unlock achievements
- **Data Export/Import (10 pts):** Export user data to JSON/CSV, import from other services
- **Multiple Reading Lists (5 pts):** Support custom reading lists beyond basic categories

API & Integration Enhancements (5-10 points)

- **External API Integration (10 pts):** Integrate with additional manga APIs (MyAnimeList, AniList)
- **Webhook System (10 pts):** Allow external services to receive notifications via webhooks

- **API Versioning (10 pts):** Implement proper API versioning with backward compatibility
- **OpenAPI Documentation (5 pts):** Generate interactive API documentation
- **Mobile-Optimized Endpoints (10 pts):** Specialized endpoints optimized for mobile apps

Security & Reliability (5-10 points)

- **Advanced Authentication (10 pts):** Implement refresh tokens and secure session management
- **Input Sanitization (5 pts):** Comprehensive input validation and sanitization
- **Automated Backups (10 pts):** Implement automated database backup system
- **Health Checks (5 pts):** Add health check endpoints for all services
- **Graceful Shutdown (10 pts):** Implement graceful shutdown for all servers

Development & Deployment (5-10 points)

- **Docker Compose Setup (10 pts):** Complete containerization with multi-service setup
- **CI/CD Pipeline (10 pts):** Automated testing and deployment pipeline
- **Environment Configuration (5 pts):** Proper environment-based configuration management
- **Database Migrations (7 pts):** Automated database schema migration system
- **Monitoring & Alerting (8 pts):** Basic system monitoring with alert notifications

Bonus Feature Selection Strategy

Recommended Bonus Features by Difficulty

For Teams Finishing Core Features Early (Weeks 8-9): - Enhanced TCP Synchronization (10 pts)

- Advanced Search & Filtering (10 pts)
- User Reviews & Ratings (10 pts)
- Reading Statistics (10 pts)

For Advanced Teams with Extra Time: - Data Caching with Redis (10 pts)

- Recommendation System (10 pts)
- Friend System (10 pts)
- CI/CD Pipeline (10 pts)

Quick Implementation Bonuses (1-2 days each): - Notification Preferences (5 pts)

- Multiple Reading Lists (5 pts)
- Health Checks (5 pts)
- Input Sanitization (5 pts)

Total Maximum Points: 120 points

- Core Project: 100 points
- Bonus Features: Up to 20 additional points
- Final Grade Calculation: min(Total Points, 100) for the 30% course component

Success Criteria

Minimum Requirements for Passing

- All five network protocols implemented and functional
- Basic user authentication and authorization
- Manga data storage and retrieval
- Progress tracking and synchronization
- Real-time chat functionality
- Successful live demonstration

Expected Learning Outcomes

- Understanding of network programming concepts in Go
- Experience with concurrent programming using goroutines
- Knowledge of different communication protocols and their use cases
- Basic distributed system integration skills
- Foundation for advanced network programming concepts

This revised specification provides a challenging but achievable project that maintains educational value while fitting realistic time and skill constraints for junior/senior level students with limited Go experience.

Regulations on AI Chatbot Usage

1. **Permitted Uses**
   - AI chatbots (e.g., ChatGPT, Gemini, Copilot) may be used for **idea brainstorming, language refinement, grammar checking, and summarization**.

   - Students may use AI to **explore programming approaches**, but final implementation must be their own.
2. **Prohibited Uses**
   - Submitting AI-generated code, documentation, or reports **without meaningful modification** is strictly prohibited.

   - Using AI to solve entire project tasks or bypass the learning objectives will be treated as **academic misconduct**.
3. **Transparency**
   - Students must **acknowledge in the report** if AI tools were used, including a brief description of how they were applied.
4. **Responsibility**
   - The student team bears full responsibility for the **accuracy, originality, and ethical use** of any AI-assisted content.

   - Any violation of this policy will result in penalties in accordance with the university's academic integrity code.

Examples of Acceptable AI Usage

Students may refer to the following examples as guidance for responsible AI usage:

- **Brainstorming Ideas:** Using AI to suggest potential project structures or approaches, then critically evaluating and modifying them.

- **Language Support:** Asking AI to refine grammar, spelling, or clarity in reports, while ensuring the technical content is student-generated.

- **Summarization:** Using AI to summarize research papers or documentation, followed by student verification and deeper reading.

- **Code Assistance:** Requesting AI to explain syntax errors or give pseudocode examples, but writing and testing the actual implementation independently.

- **Learning Aid:** Using AI to clarify complex concepts (e.g., gRPC, concurrency, WebSocket handling) as a study guide, not as final deliverables.

⚠️ **Note:** Any AI-generated content must be acknowledged and reviewed by the student. Blindly copying AI output into deliverables is not acceptable.