Academic Manga Microservices Project Guide

Table of Contents

- Comprehensive 7-Week Development Plan for Go-Based Network Programming
- Executive Summary
- Project Architecture Overview
- Expandable Project Structure
- 7-Week Development Timeline
- Core Service Implementations (Pseudocode)
- Learning Resources & Materials
- Academic Requirements Checklist
- Risk Management & Success Strategies
- Conclusion

Comprehensive 7-Week Development Plan for Go-Based Network Programming

Executive Summary

This document provides a complete development roadmap for building a **manga reading platform using Go microservices architecture** for academic evaluation. The project demonstrates proficiency in **5 network protocols** (HTTP, TCP, UDP, WebSocket, gRPC) while meeting specific scalability targets and maintaining academic focus.

Key Academic Achievements:

- 50-100 concurrent HTTP API users
- 20-30 concurrent TCP connections for progress sync
- UDP broadcasting for chapter notifications
- 10-20 concurrent WebSocket users for real-time chat
- gRPC internal service communication
- 80-90% system uptime during demonstration

Project Architecture Overview

Microservices Design

The system consists of **5 core microservices**, each demonstrating different network programming concepts:

1. **HTTP API Service (Port 8080)** - RESTful API with authentication, CRUD operations, and search functionality
2. **TCP Sync Service (Port 8081)** - Concurrent TCP connections for reading progress synchronization
3. **UDP Notify Service (Port 8082)** - Broadcast notifications for new chapter releases
4. **WebSocket Chat Service (Port 8083)** - Real-time chat rooms and comment sections
5. **gRPC Internal Service (Port 9000)** - High-performance internal service communication

Technology Stack

- **Programming Language:** Go 1.21+
- **Database:** SQLite (perfect for academic scope)
- **Cache:** Redis for session storage and performance
- **API Framework:** Gin for HTTP services
- **WebSocket:** Gorilla WebSocket for real-time communication
- **Authentication:** JWT with refresh tokens
- **Testing:** Testify framework for comprehensive testing
- **Containerization:** Docker with docker-compose
- **CI/CD:** GitHub Actions workflow

Expandable Project Structure

The project follows Go best practices with clear expansion paths:

```
manga-microservices-academic/
├── services/                    #  Core microservices
│   ├── http-api/               # HTTP API Service
│   │   ├── handlers/           #  EXPANDABLE: Request handlers
│   │   ├── middleware/         #  EXPANDABLE: Auth, CORS, logging
│   │   ├── models/            #  EXPANDABLE: Data models
│   │   ├── repository/        #  EXPANDABLE: Data layer
│   │   └── services/          #  EXPANDABLE: Business logic
│   ├── tcp-sync/              # TCP Sync Service
│   ├── udp-notify/            # UDP Notification Service
│   ├── websocket-chat/        # WebSocket Chat Service
│   └── grpc-internal/         # gRPC Internal Service
├── shared/                    #  Common components
├── proto/                     #  Protocol definitions
├── data/                      #  Database and migrations
├── scripts/                   #  Automation scripts
├── tests/                     #  Project-wide tests
└── docs/                      #  Documentation
```

Expansion Capabilities:

- **EXPANDABLE:** Can add features without breaking core structure
- **FUTURE:** Post-academic enhancement opportunities
- **CORE:** Essential academic components

7-Week Development Timeline

Week 1: Project Setup & Infrastructure

Focus: Development environment and project foundation

Member A Tasks:

- Project structure creation using Go standard layout
- Docker compose configuration for all services
- SQLite database schema design and migrations

- Redis setup and configuration

Member B Tasks:

- Go module initialization and dependency management
- Git repository setup with proper branching strategy
- Protocol buffer definitions for gRPC services
- CI/CD pipeline foundation with GitHub Actions

Learning Materials:

- "Standard Go Project Layout" (GitHub)
- "Eleven tips for structuring Go projects" (Alex Edwards)
- Docker official Go guide

Deliverables:
✅ Complete development environment
✅ Project structure with all service directories
✅ Database schemas for all services
✅ Docker setup with hot reloading

Week 2: HTTP API Service Development

Focus: RESTful API with authentication

Member A Tasks:

- REST endpoint implementation using Gin framework
- Database repository layer with GORM
- Core CRUD operations for manga and users
- Performance optimization for concurrent users

Member B Tasks:

- JWT authentication middleware implementation
- API testing suite with comprehensive coverage
- Postman collection for API documentation
- CORS and security middleware

Learning Materials:

- Gin framework documentation
- JWT authentication best practices
- Go HTTP server optimization techniques

Academic Requirements:

- Support 50-100 concurrent users
- Proper HTTP status codes and error handling
- JWT-based authentication system

Deliverables:
✅ Complete HTTP API with all endpoints
✅ Authentication system with refresh tokens

✅ CRUD operations for manga management
✅ API documentation and testing suite

Week 3: TCP Sync Service Development

Focus: Concurrent TCP socket programming

Member A Tasks:

- TCP server implementation with connection pooling
- JSON-based protocol design for progress updates
- Concurrent connection handling (20-30 connections)
- Database synchronization logic

Member B Tasks:

- TCP client implementation with retry logic
- Load testing scripts for concurrent connections
- Connection monitoring and health checking
- Protocol documentation and testing tools

Learning Materials:

- "Master TCP & UDP Sockets" (YouTube - codeHeim)
- "Socket Programming in Go - Complete Guide" (Kelche)
- Go net package documentation

Academic Requirements:

- Handle 20-30 concurrent TCP connections
- JSON-based message protocol
- Graceful connection termination
- Progress synchronization accuracy

Pseudocode Structure:

```
// TCP Server Core Logic
MAIN_FUNCTION:
    INITIALIZE connection_pool (max 30 concurrent)
    START TCP listener on port 8081

    WHILE server_running:
        ACCEPT new connection
        IF connection_count &lt; MAX_CONNECTIONS:
            SPAWN goroutine handle_connection(connection)

HANDLE_CONNECTION function:
    WHILE connection_active:
        message = READ_JSON_MESSAGE from connection
        VALIDATE and PROCESS progress update
        SAVE to database
        SEND success response
```

Deliverables:
✅ TCP sync service handling 20-30 concurrent connections
✅ JSON protocol for progress updates

✓ Load testing validation
✓ Connection monitoring dashboard

Week 4: UDP Notify & gRPC Services

Focus: Stateless UDP broadcasting and efficient gRPC communication

Member A Tasks:

- UDP broadcast server with subscriber management
- gRPC service implementation with protocol buffers
- Internal service communication patterns
- Broadcasting logic for notifications

Member B Tasks:

- UDP client registration and testing
- gRPC client implementation for other services
- Service integration testing
- Cross-service communication validation

Learning Materials:

- "Complete Golang and gRPC Microservices" (YouTube - Tiago)
- gRPC official Go tutorial
- UDP networking concepts

Academic Requirements:

- UDP broadcasting to registered clients
- Basic error handling for network failures
- gRPC internal service communication
- Protocol buffer message definitions

UDP Pseudocode Structure:

```
// UDP Notification System
HANDLE_REGISTRATIONS function:
    WHILE server_running:
        message = UDP_READ_FROM(buffer)
        SWITCH message.action:
            CASE "register": REGISTER_SUBSCRIBER()
            CASE "unregister": UNREGISTER_SUBSCRIBER()

BROADCAST_NOTIFICATION function:
    FOR each subscriber in registry:
        UDP_SEND_TO(subscriber.address, notification)
```

Deliverables:
✓ UDP notification system with client registration
✓ gRPC internal service with all required methods
✓ Cross-service communication working
✓ Protocol definitions and documentation

Week 5: WebSocket Chat Service

Focus: Real-time bidirectional communication

Member A Tasks:

- WebSocket server setup with Gorilla WebSocket
- Real-time message handling and broadcasting
- Room management system
- Connection lifecycle management

Member B Tasks:

- Chat UI development with HTML/JavaScript
- WebSocket client implementation and testing
- Multi-room chat functionality
- User interface for chat interaction

Learning Materials:

- "Mastering WebSockets With Go" (YouTube)
- Gorilla WebSocket documentation
- Real-time chat implementation examples

Academic Requirements:

- Support 10-20 concurrent users per room
- WebSocket upgrade handling
- Real-time message broadcasting
- Connection lifecycle management

WebSocket Pseudocode Structure:

```
// WebSocket Chat System
HANDLE_WEBSOCKET_UPGRADE function:
    websocket_connection = UPGRADE_CONNECTION(request)
    client = CREATE_CLIENT(connection, user_info)
    ADD_CLIENT_TO_ROOM(room, client)

    SPAWN goroutine read_messages(client)
    SPAWN goroutine write_messages(client)

BROADCAST_TO_ROOM function:
    FOR each client in room:
        client.send_channel &lt;- message
```

Deliverables:
✅ Real-time chat system with WebSocket
✅ Multi-room support for manga discussions
✅ Web interface for chat interaction
✅ Concurrent user testing (10-20 users)

Week 6: Integration & Testing

Focus: System integration and comprehensive testing

Member A Tasks:

- Full service integration
- Performance optimization across all services
- System monitoring and logging
- Load testing for academic requirements

Member B Tasks:

- End-to-end testing suite
- Integration test scenarios
- Bug tracking and resolution
- Documentation completion

Learning Materials:

- "Testing Approaches in Microservices Using Go" (Dev.to)
- Go testing best practices
- Performance testing strategies

Academic Requirements:

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

Testing Strategy:

- **Unit Testing:** >70% coverage for individual services
- **Integration Testing:** Cross-service communication
- **Load Testing:** Academic scalability requirements
- **End-to-End Testing:** Complete user scenarios

Deliverables:
✅ Fully integrated microservices system
✅ 80%+ test coverage across all services
✅ Performance benchmarks meeting academic requirements
✅ Comprehensive documentation

Week 7: Deployment & Final Demo

Focus: Production deployment and academic presentation

Member A Tasks:

- Production deployment with Docker
- Performance monitoring setup
- System optimization and tuning
- Technical demo environment

Member B Tasks:

- Academic documentation finalization
- Demo script and scenario preparation
- User guide and API documentation
- Video demo recording

Academic Requirements:

- Professional deployment demonstration
- All protocol requirements met
- Scalability targets achieved
- Complete technical documentation

Deliverables:
✅ Production-ready deployment
✅ Academic presentation materials
✅ Complete project documentation
✅ Demo video and live presentation

Core Service Implementations (Pseudocode)

HTTP API Service (Port 8080)

```
MAIN_FUNCTION:
    INITIALIZE database_connection (SQLite)
    INITIALIZE gin_router
    SETUP middleware (CORS, JWT, logging)
    REGISTER routes (public and protected)
    START server on port 8080

HANDLE_LOGIN function:
    VALIDATE credentials against database
    GENERATE JWT token (24h expiry)
    RETURN {token, user_data} or 401 error

JWT_MIDDLEWARE function:
    EXTRACT Authorization header
    VALIDATE token signature and expiration
    SET user context for protected routes
```

TCP Sync Service (Port 8081)

```
MAIN_FUNCTION:
    START TCP listener on port 8081
    WHILE server_running:
        ACCEPT connection (max 30 concurrent)
        SPAWN goroutine handle_connection()

HANDLE_CONNECTION function:
    WHILE connection_active:
        READ JSON progress update
        VALIDATE and SAVE to database
        SEND acknowledgment response
```

UDP Notify Service (Port 8082)

```
SUBSCRIBER_REGISTRY:
    subscribers: map[user_id]UDPAddress

HANDLE_REGISTRATIONS:
    WHILE server_running:
        READ registration message
        ADD/REMOVE subscriber from registry

BROADCAST_NOTIFICATION:
    FOR each subscriber:
        SEND UDP notification (fire-and-forget)
```

WebSocket Chat Service (Port 8083)

```
HANDLE_WEBSOCKET_UPGRADE:
    UPGRADE HTTP to WebSocket
    CREATE client object
    ADD to room
    SPAWN read/write goroutines

BROADCAST_MESSAGE:
    FOR each client in room:
        SEND via client channel
```

gRPC Internal Service (Port 9000)

```
SERVICE_IMPLEMENTATION:
    GET_USER_PROGRESS: Query database, return protobuf response
    UPDATE_PROGRESS: Validate and save progress data
    GET_MANGA_METADATA: Retrieve manga information
    NOTIFY_NEW_CHAPTER: Trigger UDP notifications
```

Learning Resources & Materials

Essential Video Tutorials

1. **"Complete Golang and gRPC Microservices"** - 42 minutes
    - Perfect for Week 4 gRPC implementation
2. **"Master TCP & UDP Sockets – Complete Guide"** - 10 minutes
    - Essential for Weeks 3-4 network programming
3. **"Mastering WebSockets With Go"** - In-depth tutorial
    - Required for Week 5 real-time features
4. **"Go high-performance microservices tutorial"**
    - Advanced concepts for Week 6 optimization

Technical Documentation

5. **gRPC Go Basics Tutorial** (Official)
    - Complete gRPC implementation guide
6. **Socket Programming in Go - Complete Guide** (Kelche)
    - Comprehensive TCP/UDP implementation
7. **Go Project Layout Standards** (GitHub)
    - Industry-standard project organization
8. **Testing Approaches in Microservices** (Dev.to)
    - Testing strategies and best practices

Code Examples & Repositories

9. **Cinema Microservices Example** (GitHub)
    - Real-world microservices architecture reference
10. **TCP/UDP Examples** (Multiple repositories)
    - Practical socket programming implementations

Academic Requirements Checklist

Protocol Implementation ✅

- [ ] HTTP: RESTful API with JWT auth and CORS
- [ ] TCP: Concurrent connections with JSON protocol
- [ ] UDP: Broadcasting with client registration
- [ ] WebSocket: Real-time communication with lifecycle management
- [ ] gRPC: Internal service communication with protobuf

Scalability Targets ✅

- [ ] 50-100 concurrent HTTP users
- [ ] 20-30 concurrent TCP connections
- [ ] 30-40 manga series in database
- [ ] Search queries within 500ms
- [ ] 10-20 simultaneous WebSocket users

Reliability Standards ✅

- [ ] 80-90% uptime during demonstration
- [ ] Basic error handling and recovery
- [ ] Simple logging for debugging
- [ ] Graceful degradation capabilities

Testing & Documentation ✅

- [ ] Unit tests for all services
- [ ] Integration tests for service communication
- [ ] Load testing for scalability validation
- [ ] Complete API and protocol documentation
- [ ] Academic project report

Risk Management & Success Strategies

High-Risk Areas

1. **TCP Connection Stability** (Week 3)
   - Mitigation: Incremental testing, connection pooling
2. **UDP Broadcast Reliability** (Week 4)
   - Mitigation: Client registration validation, error handling
3. **WebSocket Scalability** (Week 5)
   - Mitigation: Proper connection cleanup, resource management
4. **Service Integration** (Week 6)
   - Mitigation: Service-by-service integration, comprehensive testing

Success Factors

- **Early Protocol Testing:** Test each protocol individually before integration
- **Academic Focus:** Prioritize academic requirements over additional features
- **Incremental Development:** Build and test one service at a time
- **Documentation:** Maintain detailed logs for academic evaluation
- **Team Communication:** Daily standups and weekly planning sessions

Conclusion

This comprehensive guide provides everything needed for a successful academic microservices project in Go. The 7-week timeline balances learning objectives with practical implementation, ensuring all network programming concepts are thoroughly demonstrated while meeting academic scalability and reliability requirements.

The expandable project structure allows for future enhancements beyond academic submission, providing a solid foundation for continued learning and professional development in Go microservices architecture.

Key Success Metrics:

- ✅ 5 network protocols implemented and demonstrated
- ✅ All academic scalability targets achieved
- ✅ 80%+ test coverage and documentation
- ✅ Production-ready deployment with Docker
- ✅ Comprehensive learning experience in Go networking
  [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39]

⁂

1. https://www.youtube.com/watch?v=nno9JA1KHzw
2. https://www.youtube.com/watch?v=ea_4Ug5WWYE
3. https://www.linode.com/docs/guides/developing-udp-and-tcp-clients-and-servers-in-go/
4. https://grpc.io/docs/languages/go/basics/
5. https://encore.cloud/resources/go-frameworks
6. https://github.com/jeroendk/go-tcp-udp
7. https://groups.google.com/g/grpc-io/c/mtHxsI93LXc
8. https://viblo.asia/p/dung-golang-microservice-boilerplate-theo-clean-architecture-1Je5EzG0KnL
9. https://dzone.com/articles/socket-programming-in-go
10. https://earthly.dev/blog/golang-grpc-example/
11. https://www.linkedin.com/pulse/building-microservices-go-e-commerce-integrations-magento-chis-qeaof
12. https://www.kelche.co/blog/go/socket-programming/
13. https://zetcode.com/golang/socket/
14. https://github.com/golang-standards/project-layout
15. https://dev.to/mquanit/testing-approaches-in-microservices-using-go-48g
16. https://www.reddit.com/r/golang/comments/1ejot21/containerized_local_microservice_development/
17. https://www.reddit.com/r/golang/comments/1gboht0/best_practices_for_structuring_large_go_projects/
18. https://www.apriorit.com/dev-blog/golang-microservices-load-testing
19. https://github.com/mmorejon/microservices-docker-go-mongodb
20. https://forum.golangbridge.org/t/recommended-project-structure/35058
21. https://blog.gopheracademy.com/advent-2014/testing-microservices-in-go/
22. https://go.dev/doc/modules/layout
23. https://www.reddit.com/r/golang/comments/1cgbm8e/grpc_vs_websockets/
24. https://www.youtube.com/watch?v=lMrWO7OUMdY
25. https://docs.docker.com/guides/golang/build-images/
26. https://www.alexedwards.net/blog/11-tips-for-structuring-your-go-projects
27. https://threedots.tech/post/microservices-test-architecture/
28. https://blog.devgenius.io/heres-what-containerizing-a-go-microservice-with-docker-taught-me-about-modern-software-design-3b47ba0f1490
29. https://avivcarmi.com/finding-the-best-go-project-structure-part-1/
30. https://blog.stackademic.com/effective-testing-of-microservices-in-go-best-practices-and-tips-f4edc60d1655
31. https://dev.to/rafalsz/docker-compose-and-devcontainers-for-microservices-development-44b8
32. https://www.youtube.com/watch?v=1ZbQS6pOlSQ
33. https://www.lambdatest.com/learning-hub/microservices-testing
34. https://dev.to/manavkush/learning-microservices-with-go-the-journey-begins-34m9
35. https://www.youtube.com/watch?v=_FU6QC3q1mk
36. https://github.com/johanbrandhorst/gopherjs-grpc-websocket
37. https://www.velotio.com/engineering-blog/build-a-containerized-microservice-in-golang
38. https://leapcell.io/blog/mastering-socket-programming-in-go
39. https://www.youtube.com/watch?v=pKpKv9MKN-E