SwiftLogistics System Functionalities & Implementation Requirements

User Role Functionalities

1. E-commerce Clients (Client Portal)

- **Login/Authentication**: Secure client login system
- **Order Submission**: Submit delivery orders through web portal
- **Order Status Tracking**: View current status of submitted orders
- **Real-time Delivery Tracking**: Track deliveries in real-time
- **Order History**: View past orders and delivery records

2. Drivers (Mobile App)

- **View Daily Manifest**: Access assigned delivery list for the day
- **Route Optimization**: View optimized delivery routes
- **Real-time Updates**: Receive route changes and high-priority deliveries
- **Delivery Status Updates**: Mark packages as "delivered" or "failed"
- **Failure Reasons**: Record reasons for failed deliveries
- **Proof of Delivery**: Capture digital signatures or photos
- **Push Notifications**: Receive urgent updates and route changes

3. System Administrators (Backend Management)

- **System Integration Management**: Monitor CMS, ROS, WMS connections
- **Order Processing**: Handle high-volume order intake
- **Transaction Management**: Ensure distributed transaction consistency
- **System Monitoring**: Monitor system health and performance

What You Need to Implement

Core Components (All Members Participate)

1. **Mock Backend Systems** (Required for Prototype)

- **Mock CMS**: Simulate SOAP-based XML API for client management
- **Mock ROS**: Simulate RESTful API for route optimization

- **Mock WMS**: Simulate TCP/IP messaging for warehouse operations

2. Middleware Architecture

- **API Gateway**: Single entry point for all client requests
- **Service Orchestration**: Coordinate between different systems
- **Protocol Translation**: SOAP ↔ REST ↔ TCP/IP conversion
- **Message Broker**: Handle asynchronous communication (RabbitMQ/Kafka)
- **Service Registry**: Manage service discovery

3. Client Applications

- **Web Portal**: Client interface for order management and tracking
- **Mobile App**: Driver interface (can be web-based mobile app)
- **Real-time UI**: Live updates using WebSockets/Server-Sent Events

4. Core Business Logic

- **Order Processing Workflow**: End-to-end order handling
- **Real-time Tracking System**: Live status updates
- **Notification System**: Push notifications and alerts
- **Authentication & Authorization**: Secure access control

5. Data Management

- **Database Design**: Store orders, clients, drivers, delivery status
- **Data Synchronization**: Keep systems in sync
- **Transaction Management**: Handle distributed transactions

6. Integration Patterns

- **Message Queue Implementation**: Asynchronous processing
- **Event-Driven Architecture**: Real-time updates
- **Circuit Breaker Pattern**: Handle system failures
- **Retry Mechanisms**: Ensure reliability

7. Security Features

- **API Security**: Authentication tokens, encryption

- **Data Encryption**: Secure data transmission
- **Input Validation**: Prevent security vulnerabilities

Development Distribution Suggestions

Since all 8-9 members will be involved in development:

Frontend Team (2-3 members)

- Client web portal
- Driver mobile interface
- Real-time UI components

Backend Services Team (2-3 members)

- Mock system implementations
- API development
- Business logic services

Integration Team (2-3 members)

- Message brokers setup
- Protocol translation services
- Service orchestration

Infrastructure Team (1-2 members)

- Database design and setup
- Security implementation
- System monitoring and logging

Key Implementation Notes

- **Focus on Architecture**: Emphasis should be on middleware design patterns
- **Open Source Only**: Use technologies like Spring Boot, Node.js, RabbitMQ, PostgreSQL
- **Minimal but Functional**: Working prototype demonstrating core workflow
- **Real-time Demo**: Show live order submission to delivery completion
- **Documentation**: Architectural diagrams and technology justification required