Name: **Muhammad Ahmad Mamoon**

Roll no: **20L-2128**

Section: **8A**

Task 1:

Design the backend system for Uber's ride-hailing service.

1. User Authentication and Authorization

- **Technologies**: Implement OAuth 2.0 for secure and flexible authentication. Use JWT (JSON Web Tokens) for secure data transfer.
- **Design**: Incorporate two-factor authentication for added security. Ensure role-based access control (RBAC) to distinguish between rider, driver, and admin roles.

2. Geo-location and Mapping

2.1 Real-time Tracking of Drivers and Riders

- **GPS Integration**: Utilize GPS data from drivers' and riders' smartphones.
- **Real-Time Processing**: Implement WebSocket or similar technology for real-time data communication.
- **Mapping API**: Use third-party services like Google Maps API for mapping and geolocation services.

2.2 Calculating and Updating ETA

- **Algorithm**: Combine real-time traffic data, average speed, and distance to calculate ETA.
- **Dynamic Updates**: Use a background service to continuously update ETAs based on changing traffic conditions.

3. Ride Matching and Dispatching

3.1 Matching Algorithm

- **Factors**: Consider proximity, driver ratings, vehicle type, and current traffic conditions.
- **Efficiency**: Implement a nearest-neighbor search algorithm, optimized with geospatial indexing.

3.2 Handling Peak Hours

- **Dynamic Pricing**: Implement surge pricing to balance supply and demand.
- **Load Balancing**: Use queue management to prioritize ride requests.

4. Real-time Updates and Notifications

- **Technology**: Use push notification services like Firebase Cloud Messaging (FCM) for instant updates.
- **Features**: Include status updates, ETA changes, and other relevant notifications.

5. Payment Processing

5.1 Payment System Design

- **Gateways**: Integrate with multiple payment gateways for flexibility.
- **Fare Calculation**: Algorithm based on distance, time, demand, and type of service.
- **Refund Handling**: Automated system to handle cancellations and refunds, with manual override capabilities.

6. Scalability and Fault Tolerance

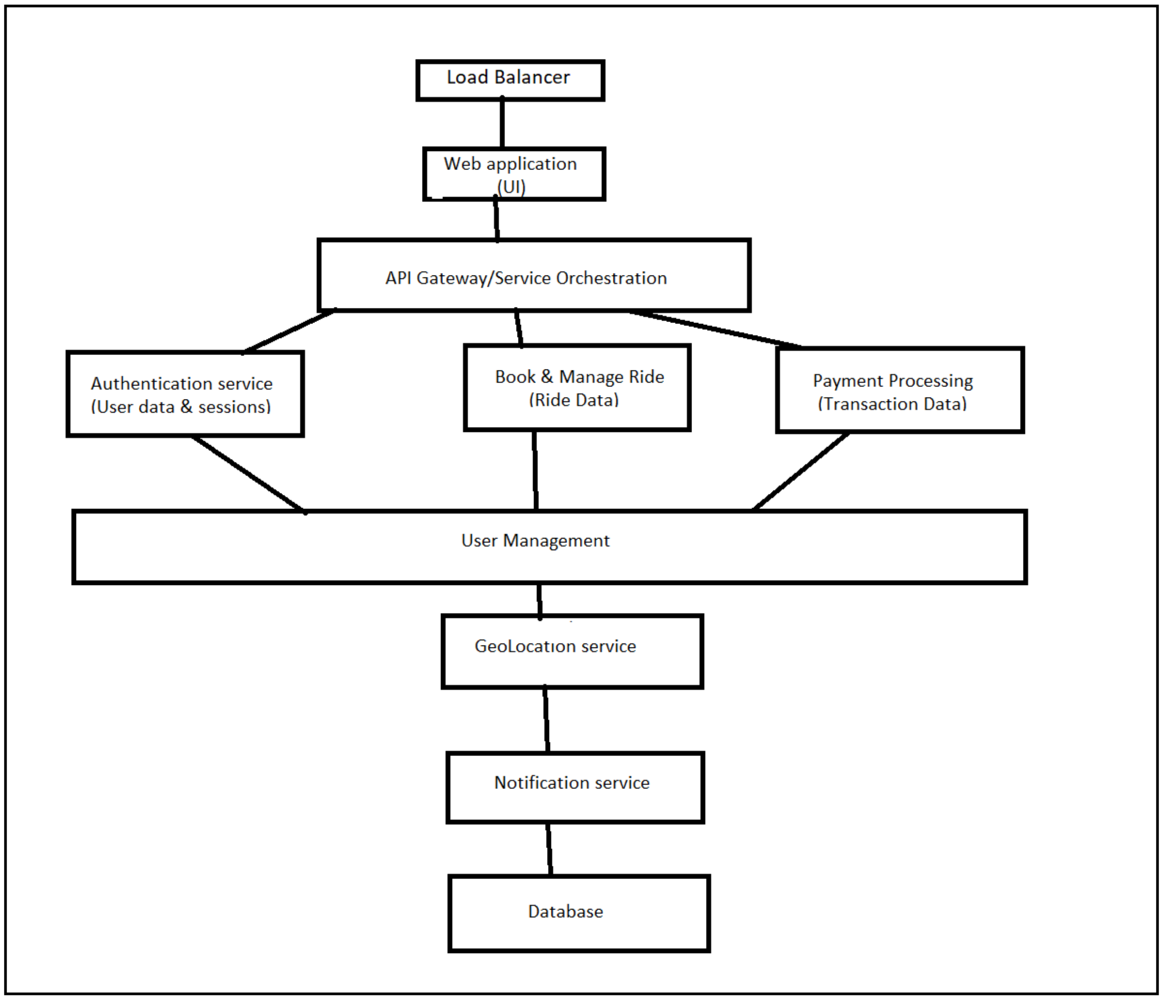
- **Microservices Architecture**: Adopt a microservices architecture for scalability.
- **Load Balancing**: Use load balancers to distribute traffic across servers.
- **Database Scaling**: Implement scalable database solutions like sharding.

7. Driver and Rider Ratings

- **Implementation**: Post-ride review system for both drivers and riders.
- **Impact**: Use ratings to influence matching algorithms and ensure quality service.

8. Data Security and Privacy

- **Encryption**: Use end-to-end encryption for data transmission.
- **Data Storage**: Follow best practices for secure data storage and access.
- **Compliance**: Ensure compliance with GDPR and other relevant data protection regulations.


Load Balancer
Web application (UI)
API Gateway/Service Orchestration
Authentication service (User data & sessions)
Book & Manage Ride (Ride Data)
Payment Processing (Transaction Data)
User Management
GeoLocation service
Notification service
Database

- **Load Balancer**: Distributes incoming traffic among multiple instances for scalability and reliability.
- **API Gateway / Service Orchestration**: Manages incoming requests, handles authentication, and directs requests to appropriate services.
- **Authentication**: Deals with user authentication, authorization, and oversees access tokens and permissions.

- **Ride Service**: Central service responsible for managing ride requests, matching drivers and riders, tracking rides, and dispatching.
- **Payment Service**: Handles payment processing, fare calculation, payment gateways, refunds, and invoices.
- **User Management**: Administers user profiles, preferences, and various user-related functions.
- **Geolocation Service**: Uses mapping APIs for real-time tracking of drivers and riders.
- **Notification Service**: Sends live updates and notifications to users at different ride stages.
- **Database Layer**: Stores and manages data using a schemaless infrastructure, ensuring scalability, fault tolerance, and efficient data retrieval.

Task 2:

Design the backend System for Shopify's e-commerce platform

1. Product Catalog and Inventory Management

1.1 Database Schema Design

- Tables: Create separate tables for Products, Categories, Inventory, and ProductVariations.
- Relationships: Use foreign keys to connect Products to Categories and ProductVariations, and Inventory to Products.
- Normalization: Normalize the database to reduce redundancy and improve data integrity.

1.2 Handling Variations and Categories

- Variations: Use a polymorphic association to handle multiple types of product variations.
- Categories: Implement a hierarchical structure for categories to allow nesting.

2. User Authentication and Authorization

- Authentication: Implement token-based authentication (JWT).

- Authorization: Use RBAC to define roles and permissions for different types of users.

3. Shopping Cart and Checkout

3.1 Shopping Cart System

- Session Storage: Store cart items in a session or cookie for unregistered users.
- Persistent Storage: Use a database to store cart items for registered users.
- Update Mechanism: Allow real-time updates to quantities and item removal.

3.2 Checkout Process

- Payment Gateways: Integrate with multiple payment gateways (e.g., Stripe, PayPal).
- Order Confirmation: Send automated confirmation emails and update order status in the database.

4. Order Processing and Fulfillment

4.1 Processing Orders

- Workflow: Implement state machines to manage order status transitions.
- Fulfillment Coordination: Use a message queue system to communicate with fulfillment services.

4.2 Returns and Refunds

- Policy Enforcement: Define clear policies and conditions for returns.
- Refund Processing: Integrate with payment gateways for processing refunds.

5. Search, Analytics, and Recommendations

5.1 Search Functionality

- Search Engine: Use ElasticSearch for powerful full-text search capabilities.
- Indexing: Regularly update the search index as products are added or updated.

5.2 Product Recommendations

- User Behavior Tracking: Implement tracking of user activity and browsing history.
- Recommendation Engine: Use machine learning algorithms to provide personalized product recommendations.

6. Scalability and High Availability

6.1 Handling Large Traffic

- Horizontal Scaling: Use cloud services that allow for easy horizontal scaling.
- Caching: Implement caching strategies using tools like Redis.

6.2 Load Balancing and Availability

- Load Balancers: Deploy load balancers to distribute traffic evenly across servers.
- Replication: Use database replication to enhance availability and performance.

7. Security Measures

7.1 Protecting User Data

- Data Encryption: Use TLS for data in transit and at-rest encryption for sensitive data.
- Compliance: Follow PCI DSS guidelines for handling transactions.

7.2 Guarding Against Threats

- Input Validation: Prevent SQL injection and XSS attacks with proper input validation.
- Regular Audits: Conduct security audits and penetration testing regularly.

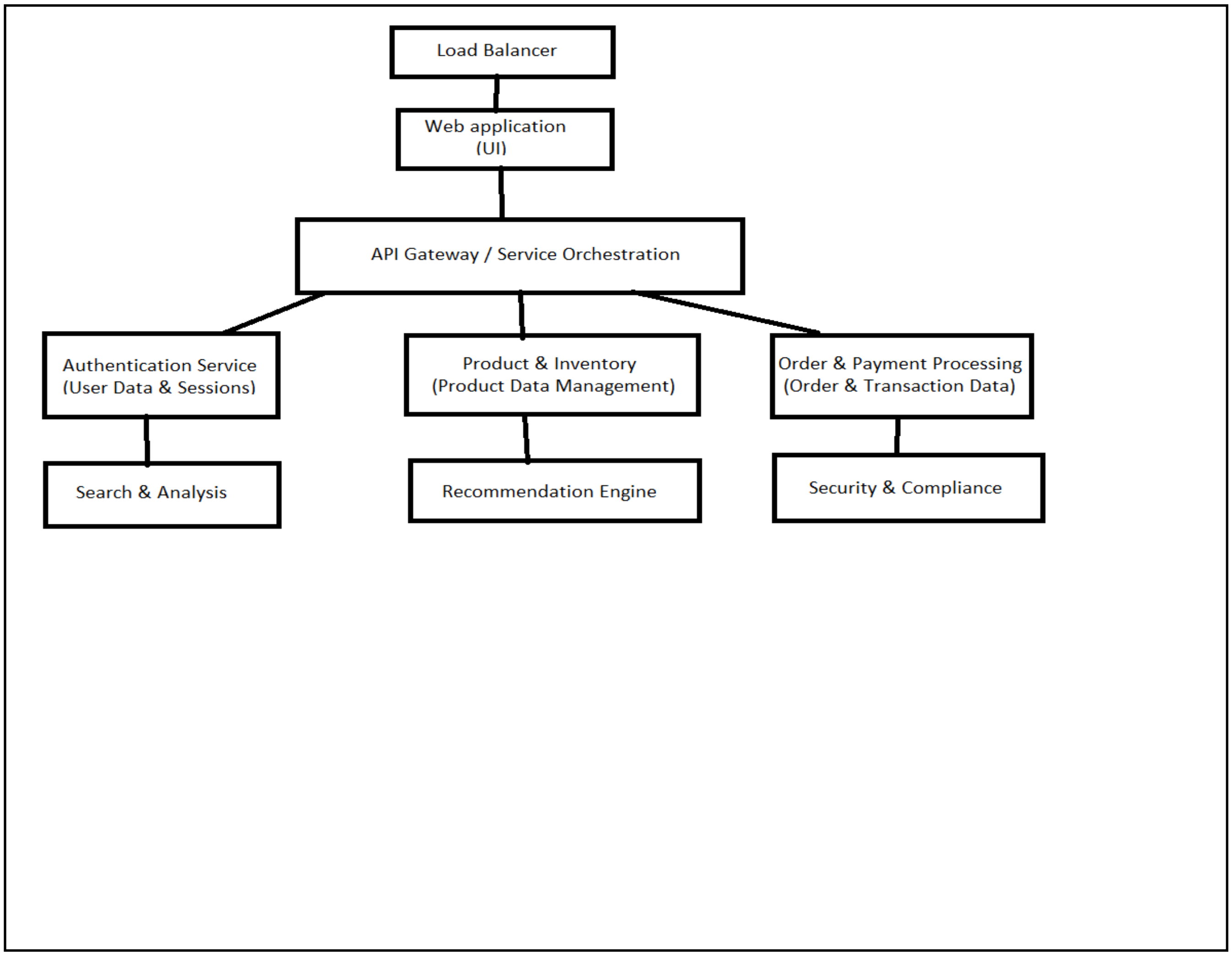
8. Customer Reviews and Ratings

8.1 Review System

- Moderation: Allow for moderation of reviews to prevent inappropriate content.
- Verification: Implement verified purchase badge to distinguish reviews.

9.1 Managing Reviews and Ratings

- Fake Review Prevention: Use algorithms to detect and flag potential fake reviews.
- Ratings Algorithm: Create an algorithm that factors in the age of the review and verified purchases.


Load Balancer
Web application (UI)
API Gateway / Service Orchestration
Authentication Service (User Data & Sessions)
Product & Inventory (Product Data Management)
Order & Payment Processing (Order & Transaction Data)
Search & Analysis
Recommendation Engine
Security & Compliance

- **Load Balancer**: Handles incoming traffic and spreads it across multiple instances for scalability and dependability.
- **Web Application (Frontend/UI):** Represents the user interface where customers engage with the platform.
- **API Gateway / Service Orchestration**: Manages requests, oversees authentication, and directs requests to specific backend services.
- **Authentication Service**: Manages user authentication, authorization, and oversees user sessions.
- **Product & Inventory Management**: Controls product information, variations, categories, and inventory levels, managing the product catalog.
- **Order & Payment Processing**: Processes orders, supervises payment gateways, and tracks transactional data associated with orders.
- **User Data & Sessions**: Stores user-related information and session data to maintain user states during interactions.

- **Search & Analytics Engine**: Executes product search functionalities and provides analytics on customer behavior and trends.
- **Recommendation Engine**: Uses user behavior and purchase history to generate personalized product recommendations.
- **Security & Compliance**: Implements security measures to safeguard user data and ensures adherence to industry standards and regulations.