**Banking Reward Application**

**Deliverables:**

* **Codebase: Finalized and pushed to GitHub.**
* **Documentation: README, API documentation (Swagger/Postman), ER diagram.**
* **Database Schema: SQL scripts or exported schema.**
* **Demo: A live demo or recorded video showcasing the application.**

**Project Timeline:**

| **Phase** | **Tasks** | **Timeline** |
| --- | --- | --- |
| **Kick-Off** | **Define scope, assign roles, set up repo** | **Day 1** |
| **Phase 1** | **Build core features (user mgmt, rewards)** | **Week 1–2** |
| **Phase 2** | **Advanced features (referral, insights)** | **Week 3–4** |
| **Phase 3** | **Testing and debugging** | **Week 5** |
| **Finalization** | **Documentation and presentation prep** | **Week 6** |

**Core Features**

**1. User Management**

* Users can register with details like name, email, phone number, and account number.
* Secure login using authentication (e.g., username and password).
* Role-based access:
  + **Admin**: Manage rewards, monitor user activity, and view system reports.
  + **Customer**: Earn and redeem rewards, view account and reward details.

**2. Rewards System**

* Reward Points Allocation:
  + Users earn points for banking activities such as:
    - Deposits.
    - Fund transfers.
    - Loan repayments.
  + Define reward rules, e.g.:
    - Earn 1 point for every $100 deposited.
    - Earn 2 points for every $500 transferred.
* Redemption System:
  + Users can redeem points for predefined rewards (e.g., gift cards, cashback).
  + Redemption thresholds, e.g., 100 points = $5 cashback.

**3. Transactions and Points Tracking**

* Maintain a transaction log for users, including:
  + Banking activity details (type, amount, date).
  + Points earned per transaction.
* Reward point summary for users:
  + Total points earned.
  + Points redeemed.
  + Current balance of reward points.

**4. Rewards Catalog**

* Admin can manage a catalog of available rewards:
  + Reward types (e.g., cashback, discounts, gift vouchers).
  + Points required for redemption.
  + Availability status (active/inactive).

**5. Notifications**

* Send notifications to users:
  + On earning reward points (e.g., "You’ve earned 10 points for your recent deposit!").
  + On successful redemption (e.g., "Your $5 cashback has been processed.").
* Optional: Email or SMS notifications for major activities.

**Optional/Advanced Features (For Extra Credit)**

**1. Tiered Rewards System**

* Introduce user tiers (e.g., Silver, Gold, Platinum) based on account activity or point balance.
* Higher tiers offer more benefits, such as:
  + Bonus points (e.g., 10% extra for Gold members).
  + Faster redemption options.

**2. Referral Program**

* Allow users to refer friends and earn bonus points upon successful registration of referrals.

**3. Analytics and Insights**

* Provide users with insights into their rewards activity:
  + Monthly points summary.
  + Most frequent rewards redeemed.
* Admin dashboard:
  + Total points earned/redeemed by all users.
  + Top users with the highest points.

**4. Reward Expiry**

* Introduce an expiry system for reward points:
  + Points older than a certain period (e.g., 1 year) will expire.
* Notify users before point expiry.

**5. API Integration**

* Integrate external APIs for offering rewards, such as e-commerce gift cards or travel miles.

**Non-Functional Requirements**

1. **Security**:
   * Use Spring Security for authentication and role-based authorization.
   * Encrypt sensitive data like passwords.
   * Validate inputs to prevent SQL injection or XSS attacks.
2. **Scalability**:
   * Design the rewards system to handle increasing users and transactions.
   * Use pagination for displaying transaction and rewards history.
3. **Maintainability**:
   * Follow a layered architecture (Controller, Service, Repository).
   * Use meaningful naming conventions and comments for clarity.
4. **Performance**:
   * Optimize queries for transaction and points calculation.
   * Cache frequently accessed data (e.g., rewards catalog).

**Database Design**

**Tables:**

1. **Users**:
   * user\_id (PK), name, email, password, account\_number, role.
2. **Transactions**:
   * transaction\_id (PK), user\_id (FK), type (deposit/transfer/loan), amount, date.
3. **Reward\_Points**:
   * reward\_id (PK), user\_id (FK), points\_earned, points\_redeemed, balance, date.
4. **Rewards\_Catalog**:
   * reward\_id (PK), reward\_name, points\_required, type (cashback/voucher), status.
5. **Redemption\_History**:
   * redemption\_id (PK), user\_id (FK), reward\_id (FK), points\_used, date.

**Sample Endpoints**

1. **User Operations**:
   * POST /register: Register a new user.
   * POST /login: User login.
   * GET /users/{userId}/rewards: View reward points and history.
2. **Transaction Operations**:
   * POST /transactions: Record a new transaction.
   * GET /transactions/{userId}: Fetch user transaction history.
3. **Rewards Management**:
   * GET /rewards: Fetch the rewards catalog.
   * POST /rewards/redeem: Redeem reward points.
   * POST /rewards/add: Add new rewards (admin only).

**Deliverables for the Project**

1. **Codebase**:
   * Backend with Spring Boot (RESTful services).
   * Database schema and scripts.
2. **Documentation**:
   * API documentation (Swagger or Postman collection).
   * README with setup instructions.
3. **Presentation**:
   * Brief project demo showing user workflows.

**Readiness Items**

1. **Technical Preparation**
   * Ensure all team members have the required tools installed:
     + **Java Development Kit (JDK)**.
     + **IDE**: IntelliJ IDEA, Eclipse, or Spring Tool Suite.
     + **Database**: MySQL/PostgreSQL with access credentials.
     + **Version Control**: Git installed and GitHub accounts ready.
     + **Postman** (or an alternative) for testing APIs.
2. **Environment Setup**
   * Configure development environments:
     + Install Spring Boot dependencies.
     + Set up a database connection in application.properties.
     + Verify database drivers (e.g., mysql-connector-java) are included in the project.
3. **Framework Configuration**
   * Create a basic Spring Boot project with:
     + A layered architecture: **Controller**, **Service**, **Repository**.
     + A sample @RestController endpoint to test API functionality.
   * Set up JPA/Hibernate for database interaction.
4. **Database Readiness**
   * Design and create a preliminary database schema:
     + Tables for Users, Transactions, Reward\_Points, Rewards\_Catalog.
   * Use tools like MySQL Workbench or pgAdmin for database visualization.
5. **Coding Standards and Guidelines**
   * Agree on coding practices:
     + Use meaningful variable and method names.
     + Maintain consistent indentation and formatting.
     + Include comments and Javadoc for public methods.
6. **Documentation Preparation**
   * Set up initial documentation templates:
     + **API Design**: Outline endpoints, methods, and request/response formats.
     + **ER Diagram**: Visualize database schema.
     + **Project Overview**: Describe the purpose and features.
7. **Testing Framework**
   * Include JUnit in the project and write a sample unit test.
   * Decide testing strategy:
     + Unit testing for service methods.
     + Integration testing for API endpoints.
     + Manual testing for user flows.
8. **Risk and Contingency Planning**
   * Identify potential risks (e.g., team member unavailability, technical issues).
   * Prepare backup plans, such as redistributing tasks or simplifying features if needed.
9. **Collaboration Readiness**
   * Use project management tools like Trello, Asana, or GitHub Projects to track tasks.
   * Agree on a daily/weekly standup to discuss progress and blockers.