# **Rewards Application - High-Level Design Document**

## **1. Introduction**

This document outlines the high-level design of the Rewards App (REST API) application. The purpose of this document is to describe the architecture, key components, and technologies used in developing the application.

### **1.1. Purpose**

The purpose of this Rewards App is to provide a standardized interface for external systems to interact with the application. It allows CRUD operations on various resources such as Customers, Transactions & Rewards etc..,

### **1.2. Scope**

This document covers:

* Overview of the API architecture
* Key design components and their responsibilities
* Data flow and interactions between components
* Security considerations
* Technologies used

## **2. Architecture Overview**

### **2.1. Layered Architecture**

The application will follow a layered architecture:

* **Presentation (Controller) Layer:** Handles client requests (HTTP) and responses.
* **Service Layer:** Implements the business logic.
* **Data Access (JPA) Layer:** Interacts with the database for CRUD operations.
* **Database:** Stores persistent data.

### **2.2. RESTful API Design Principles**

* **Stateless:** Each API request from the client to the server must contain all the information needed to process the request.
* **Resource-based URIs:** Each resource (e.g., Customers, Transactions) will be represented by a unique URI.
* **HTTP Methods:**
  + GET: Retrieve resources
  + POST: Create new resources
  + PUT/PATCH: Update existing resources
  + DELETE: Remove resources
* **Content Negotiation:** Support for JSON as the primary format for requests and responses.

## **3. Key Design Components**

### **3.1. API Gateway**

An API Gateway serves as the entry point to the system. It routes requests to appropriate Rest API endpoints and handles cross-cutting concerns like:

* Authentication and authorization
* Logging
* Request/response transformation
* Error Handling

### **3.2. Controllers**

Each controller will map to a resource in the system (e.g., CustomerController, TransactionController, RewardController). Controllers handle HTTP requests, delegate tasks to the service layer, and return appropriate HTTP responses.

### **3.3. Services**

The service layer contains the business logic of the application. It processes the data, applies business rules, and interacts with the data access layer to persist data.

### **3.4. Data Access Layer (Repositories)**

The data access layer is responsible for interacting with the database. It provides CRUD operations on entities using:

* **ORM (e.g., Hibernate or JPA):** Manages database interaction, ensuring object-relational mapping.
* **Custom Queries:** For more complex data access patterns.

### **3.5. Exception Handling**

Global exception handling will be implemented to capture and handle errors gracefully. Error responses will follow standard formats like:

* 400 Bad Request
* 401 Unauthorized
* 403 Forbidden
* 404 Not Found
* 500 Internal Server Error

## **4. Data Model**

### **4.1. Entities and Relationships**

* **Customer Entity:** Represents application users / customers.
* **Transaction Entity:** Represents orders of the customers in the system.
* **Reward Entity:** Represents reward points offered to customers based on their orders,.

**Sample UML:**

|  |  |  |
| --- | --- | --- |
| CUSTOMER | TRANSACTION | REWARD |
| CUSTOMER\_ID  CUSTOMER\_NAME  EMAIL\_ID  PASSWORD  PHONE\_NUMBER | TRANSACTION\_ID  AMOUNT  CUSTOMER\_ID  DATE  SPENT\_DETAILS | REWARD\_ID  CUSTOMER\_ID  POINTS  REWARD\_MONTH  REWARD\_YEAR |

### **4.2. Database Schema**

The database schema will be designed using an RDBMS (Here H2 Database is used for demo purpose). Each entity will map to a table, and relationships will be defined using foreign keys.

## **5. Security**

### **5.1. Authentication**

* **JWT:** Authentication will be handled using JSON Web Tokens (JWT)

### **5.2. Encryption**

* **Data Encryption:** Sensitive data such as passwords will be encrypted using industry-standard algorithms (e.g., bcrypt for password hashing).

## **6. Technologies**

* **Programming Language:** Java.
* **Framework:** Spring Boot (Java),
* **Database:** H2, (Can be migrated to any of RDBMS like PostgreSQL, MySQL, MongoDB by changing DB related properties)
* **ORM:** JPA(Java)
* **API Documentation:** Swagger/ Open API
* **CI/CD Pipeline:** GitHub Actions

## **7. API Endpoints**

### **7.1. User Management**

* POST/customer: Register customer / user
* POST /customer/login: Authenticate registered used and return JWT Token
* GET /customer/logout: Logout of the session

### **7.2. Order / Transaction Management**

* GET /order/{transactionId}: Retrieve the order based on id
* POST /order: Create a new order / transaction
* PUT /order: Update a specific transaction
* DELETE /order/{transactionId}: Delete a transaction
* POST /order/postMultiple: Creates multiple orders / transactions

### **7.3. Reward Management**

* GET /reward/all: Retrieve list of rewards for all customers
* POST /api/reward: Create a new reward
* GET /customer/{customerId}: Retrieve reward points for specific customer
* GET /reward/total: Retrieves reward points for all customers based on Month and also total reward points

## **8. Logging and Monitoring**

### **8.1. Logging**

* **Log Levels:** Error, Warning, Info, Debug logs will be implemented. Using slf4j