**High-Level Design Document for Rewards System**

**1. Introduction**

* **Project Overview:**  
  The Rewards System is designed to manage customer transactions, calculate reward points,. The system handles CRUD operations for customer transactions, calculates rewards, and provides APIs for users to interact with.
* **Objective:**  
  The goal of this project is to provide an API-driven system that allows users to record transactions and retrieve rewards for customers. The system will include roles for different users (e.g. Admin,User) to perform certain operations like deleting transactions.
* **Technologies Used:**
  + Java 8
  + Spring Boot
  + Spring Security
  + JPA (Java Persistence API)
  + Oracle Database (or any database for storing data)
  + Maven (for dependency management)

**2. System Architecture**

The architecture of the Rewards System follows a layered design pattern with the following layers:

* **Controller Layer:** Handles HTTP requests, maps them to service methods, and returns HTTP responses.
* **Service Layer:** Contains business logic for processing transactions and calculating rewards.
* **Repository Layer:** Interacts with the database to perform CRUD operations.

**Components:**

* **API Layer (Controller):** Exposes RESTful endpoints for external clients.
* **Service Layer:** Handles the core functionality, such as calculating reward points and saving transactions.
* **Persistence Layer (JPA Repositories):** Interfaces with the database to store and retrieve data.

**3. Components and Modules**

* **TransactionController:**
  + Handles HTTP requests related to transactions (e.g., create, retrieve, and delete transactions).
  + API Endpoints:
    - GET /api/transactions: Retrieves all transactions.
    - GET /api/transactions/customer/rewards/{customerId}: Retrieves rewards for a specific customer.
    - GET /api/transactions/customerTransaction/{customerId}: Retrieves transactions for a specific customer.
    - POST /api/transactions: Creates a new transaction and calculates rewards.
    - DELETE /api/transactions/delete/{id}: Deletes a transaction (Admin only).
* **TransactionService:**
  + Business logic related to transactions, such as saving transactions and retrieving them based on customer ID.
* **RewardService:**
  + Responsible for calculating and saving reward points based on customer transactions.
* **Security and Authentication:**
  + Uses Spring Security to secure the endpoints.
  + Admin-only access is required for deleting transactions and update.
* **Persistence Layer (Repositories):**
  + **CustomerTransactionRepository:** Interacts with the database for saving and retrieving transaction data.
  + **RewardPointsRepository:** Interacts with the database for storing and retrieving reward points.

**4. Database Design**

**1. Customer\_Tab Table:**

This table will store information about customers.

**Columns:**

* **id (NUMBER(19))**: The unique identifier for each customer (Primary Key). This is auto-generated.
* **name (VARCHAR2(255))**: The name of the customer.
* **email (VARCHAR2(255))**: The email address of the customer. It should be unique for each customer.

**2. Customer\_Transaction\_Tab Table:**

This table will store transaction information for each customer.

**Columns:**

* **id (NUMBER(19))**: The unique identifier for each transaction (Primary Key). This is auto-generated.
* **customer\_id (NUMBER(19))**: A foreign key referencing the id column of the Customer\_Tab table, linking each transaction to a specific customer.
* **amount (NUMBER(10, 2))**: The amount for the transaction. It must be a positive value.
* **transactionDate (DATE)**: The date of the transaction.
* **points (NUMBER(10))**: The reward points earned from the transaction.

**3. Reward\_Points\_Tab Table:**

This table will store the reward points accumulated by customers.

**Columns:**

* **id (NUMBER(19))**: The unique identifier for each reward points entry (Primary Key). This is auto-generated.
* **customer\_id (NUMBER(19))**: A foreign key referencing the id column of the Customer\_Tab table, linking reward points to a specific customer.
* **month (NUMBER(2))**: The month in which the reward points were accumulated.
* **year (NUMBER(4))**: The year in which the reward points were accumulated.
* **points (NUMBER(10))**: The reward points accumulated for a specific month and year.

**5. API Design**

* **GET /api/transactions**
  + Description: Fetches all transactions.
  + Response: A list of all transactions.
* **GET /api/transactions/customer/rewards/{customerId}**
  + Description: Fetches reward points for a given customer.
  + Response: List of rewards points for the specified customer.
* **GET /api/transactions/customerTransaction/{customerId}**
  + Description: Fetches all transactions for a given customer.
  + Response: A list of transactions for the specified customer.
* **POST /api/transactions**
  + Description: Creates a new transaction and calculates reward points.
  + Request Body: JSON object representing the transaction.
  + Response: Reward points generated for the customer.
* **DELETE /api/transactions/delete/{id}**
  + Description: Deletes a transaction by ID (Admin only).
  + Response: Success or failure message.

**6. Security and Authorization**

* **Role-based Access Control (RBAC):**
  + Only users with the ADMIN role can delete transactions (@PreAuthorize("hasRole('ADMIN')")).
  + Other endpoints are accessible by authenticated users.

**Authorization:**

* + Using Spring Security annotations to define who can access which resources.

**7. Error Handling and Logging**

* **Error Handling:**
  + Standardized error response format with HTTP status codes.
  + Common errors handled include transaction not found, unauthorized access, and invalid input.
* **Logging:**
  + SLF4J Logger is used for logging critical information, such as API calls and errors.

**8. Testing**

* **Unit Tests:**
  + Test individual service methods for calculating rewards and saving transactions.
* **Security Tests:**
  + Ensure that unauthorized users cannot access sensitive endpoints.

**9. Deployment**

* The system is designed to be deployed in a cloud environment (e.g., AWS, Azure, or Google Cloud).