High-Level Documentation Design for Reward Tracker Assignment

1. Introduction

The **Reward Tracker** is a Spring Boot–based application designed to track customer transactions and calculate reward points based on business rules. The application exposes RESTful APIs for managing customers, transactions, and reward points, along with features for authentication and error handling. This document outlines the overall architecture, key components, data model, and design considerations of the project.

2. Architecture Overview

The application is structured using a layered architecture to separate concerns and improve maintainability. The main layers are:

- **Presentation Layer (Controller):**
- Handles HTTP requests and responses. It uses Spring Boot REST controllers to expose endpoints (e.g., CustomerController, CustomerTransactionController).
- **Service Layer:**

Contains business logic such as calculating reward points, processing transactions, and handling customer-related operations. This layer communicates with the repository layer.

- **Data Access Layer (Repository):**

Uses Spring Data JPA to interact with the database (MySQL). It abstracts database operations with repository interfaces (e.g., CustomerRepo, CustomerTransactionRepo, RewardPointsRepo).

- **Domain Model:**

Defines the key entities of the system (Customer, CustomerTransaction, RewardPoints) along with their relationships.

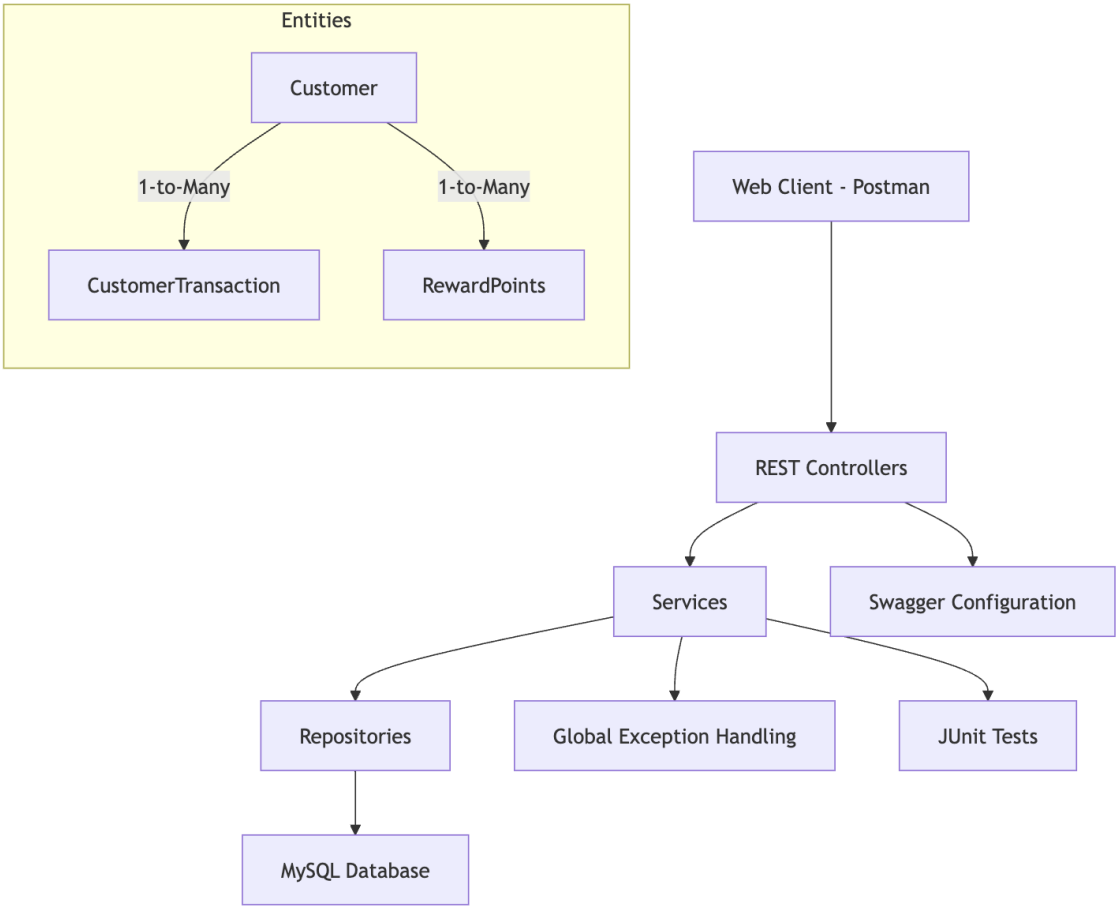
- **Global Exception Handling:**

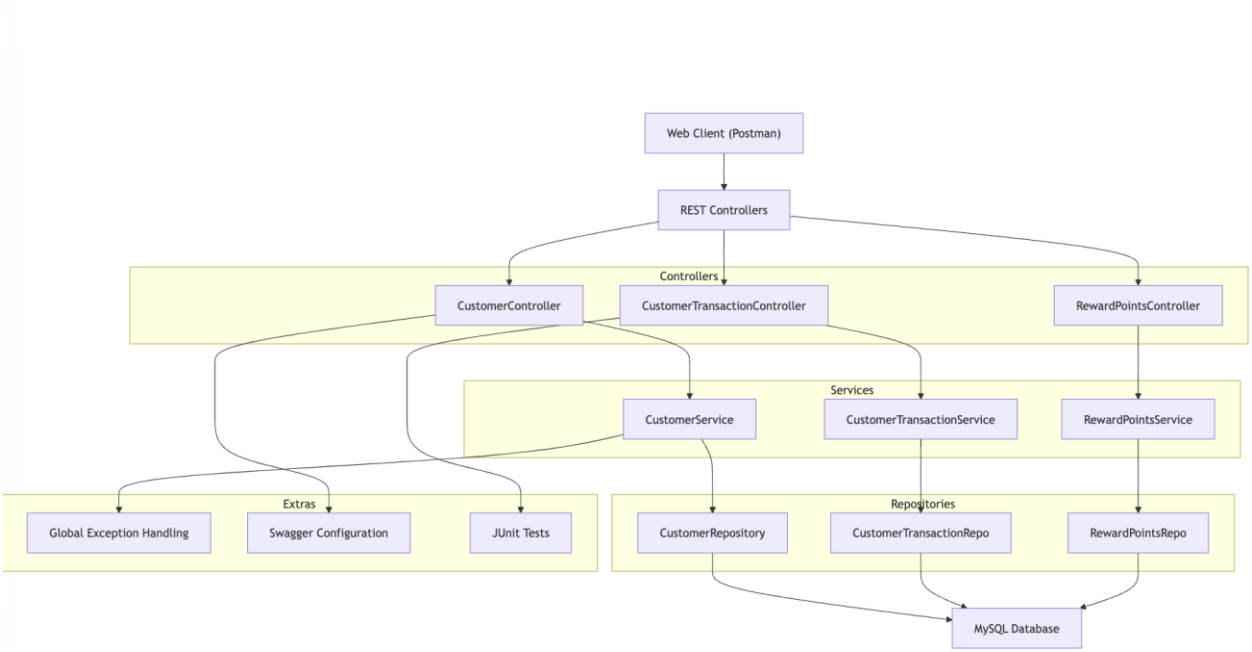
A centralized mechanism (using @RestControllerAdvice) to catch and process exceptions thrown in any layer, ensuring a consistent error response.

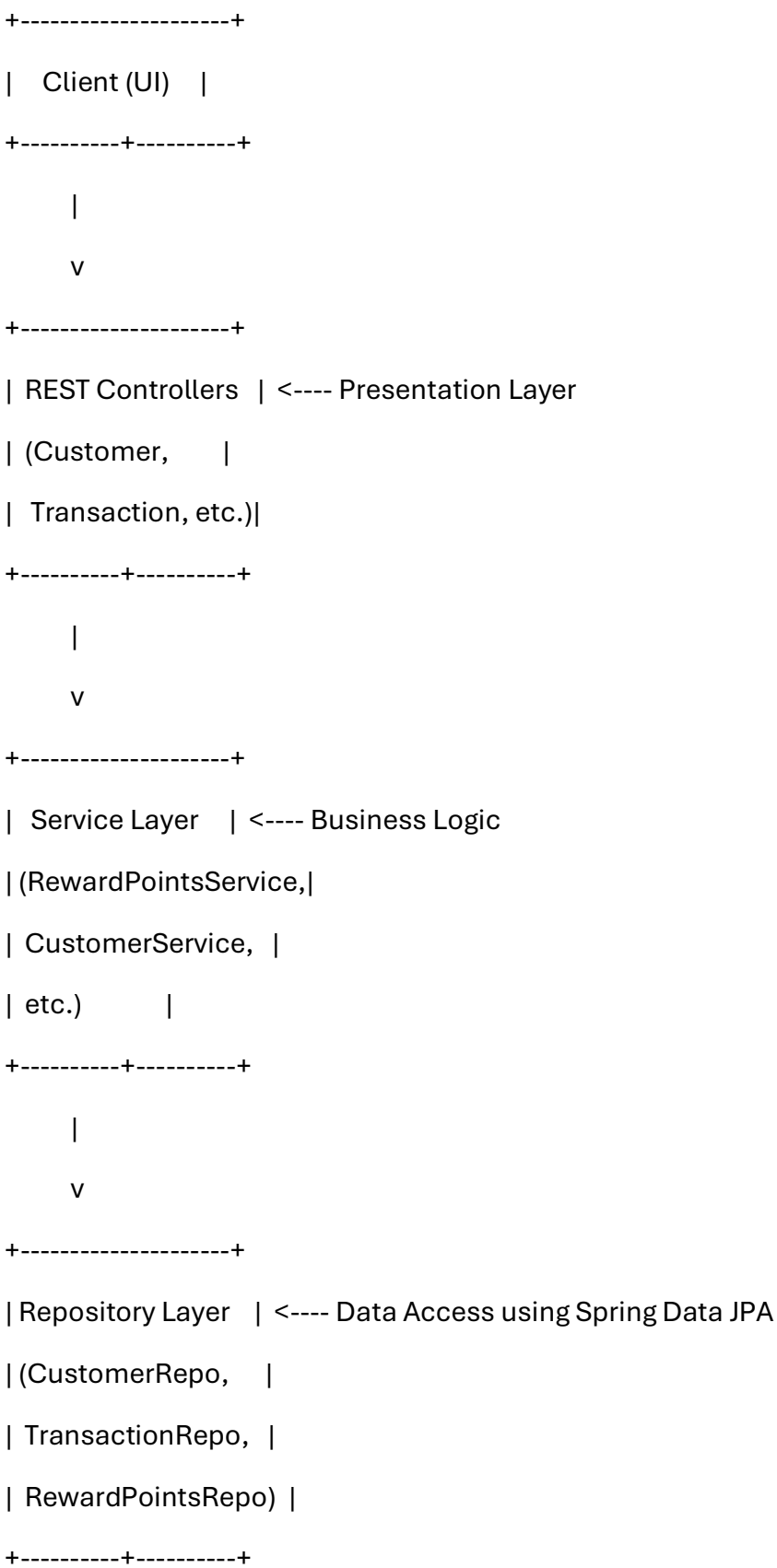
- **Testing Layer:**

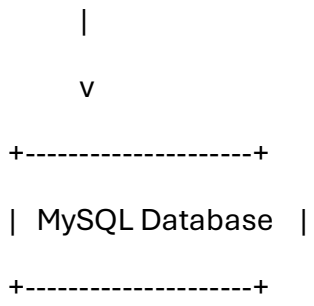
Unit tests and integration tests are written using JUnit 5 and Mockito to validate the behavior of controllers, services, and repositories.

High-Level Component Diagram






Client (UI)
REST Controllers
(Customer,
Transaction, etc.)
Presentation Layer
Service Layer
(RewardPointsService,
CustomerService,
etc.)
Business Logic
Repository Layer
(CustomerRepo,
TransactionRepo,
RewardPointsRepo)
Data Access using Spring Data JPA



3. Technology Stack

- **Backend:**
  - Java (version 11 above)
  - Spring Boot
  - Spring Data JPA
  - Spring Web (REST APIs)
- **Database:**
  - MySQL
- **Documentation & Testing:**
  - Swagger/OpenAPI for API documentation
  - JUnit 5 and Mockito for unit and integration testing
  - Maven for build and dependency management

4. Data Model

Entities:

1. **Customer**
   a. **Fields:**
   b. id, name, email, password
   c. **Relationships:**

One-to-many with **CustomerTransaction**
One-to-many with **RewardPoints**

2. **CustomerTransaction**
   a. **Fields:**

id, amount, date, spentDetails

b. **Relationships:**

Many-to-one with **Customer**

3. **RewardPoints**
    a. **Fields:**

id, rewardMonth, rewardYear, points, createdAt

b. **Relationships:**

Many-to-one with **Customer**

5. Key Functional Flows

5.1 Reward Points Retrieval Flow

1. **API Call:**
2. Client sends a GET request to /reward/customer/{customerId}.
3. **Controller:**

The controller invokes the service method getRewardPointsByCustomerId(customerId).

4. **Service:**

The service retrieves all reward points for the given customer, sorts them by year and month, sums the total points, and formats the monthly breakdown.

5. **Repository:**

The service calls the RewardPointsRepo.findByCustomerId(customerId) to fetch data.

6. **Response:**

The result is sent back to the client as a JSON response.

5.2 Reward Points Calculation and Save Flow

1. **API Call:**

Client (or a scheduled job) triggers an operation to calculate reward points from transactions.

2. **Controller/Service:**

The service method calculateAndSaveRewardPoints(customerId) retrieves transactions, groups them by month, calculates points (using a helper method), and either updates or creates new reward points records.

3. **Repository:**

The repository methods are used to fetch transactions and update/create reward points.

4. **Response:**

A summary of the reward points calculation is returned as a JSON response.

6. Global Exception Handling

- **Purpose:**
- To provide a uniform error response structure for any exception thrown in the application.
- **Implementation:**

Create a class annotated with @RestControllerAdvice that defines methods annotated with @ExceptionHandler for different types of exceptions (e.g., ResourceNotFoundException, MethodArgumentNotValidException, and a generic Exception handler).

- **Error Response Structure:**

A typical error response contains a timestamp, error message, and details of the request.

7. Testing Strategy

Unit Testing:

- **Focus:**

Testing individual methods in service and controller layers using JUnit 5 and Mockito.

- **Example:**

Test RewardPointsService.getRewardPointsByCustomerId by mocking RewardPointsRepo.

- **Tools:**

Spring Boot's test framework, MockMvc for Restcontroller tests.

API Documentation Testing:

- **Swagger/OpenAPI:**

Use Swagger UI to manually test and document your REST endpoints.

8. Security Considerations

- **Authentication and Authorization:**

integrate Spring Security to protect sensitive endpoints (e.g., login, logout, customer data).

- **Data Validation:**

Ensuring request data is validated using annotations (e.g., @Valid,min and max if needed).

9. Conclusion

To conclude, this High-Level Design document gives us a clear understanding of the **Reward Tracker System**. We have covered the **system architecture, data model, key workflows, and testing approach**. This document acts as a blueprint for developers, architects, and stakeholders to align on the overall system structure and design choices.

As we move forward, we can refine this document to include **deployment strategies, monitoring solutions, scalability improvements, and enhanced security measures**. This will help ensure that our system remains **efficient, secure, and scalable** as per the growth and requirements."