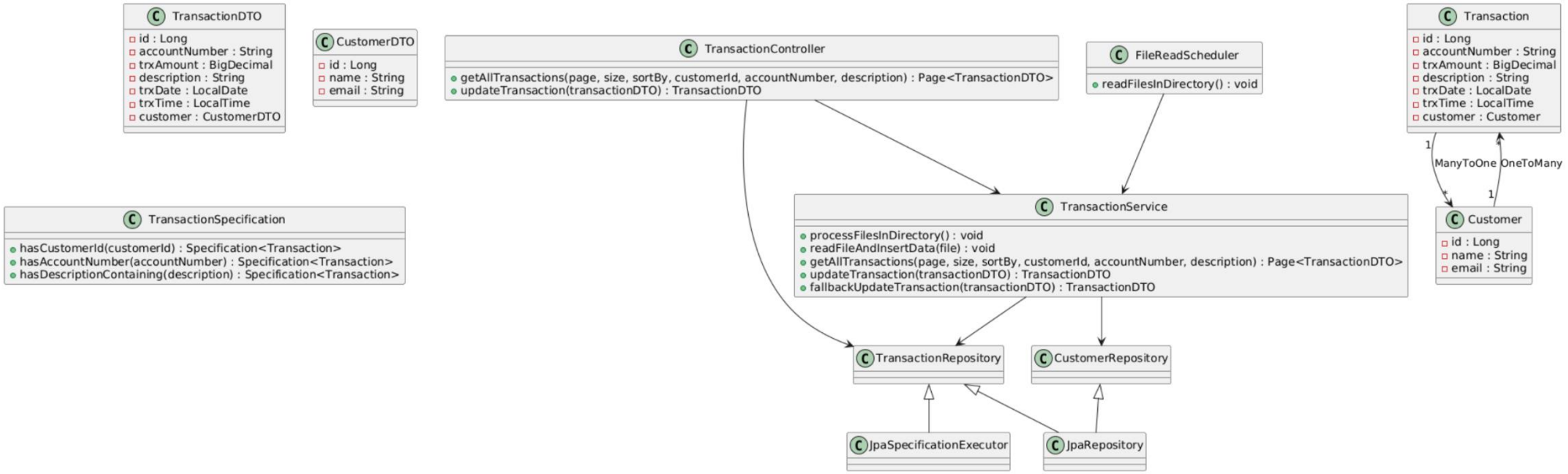
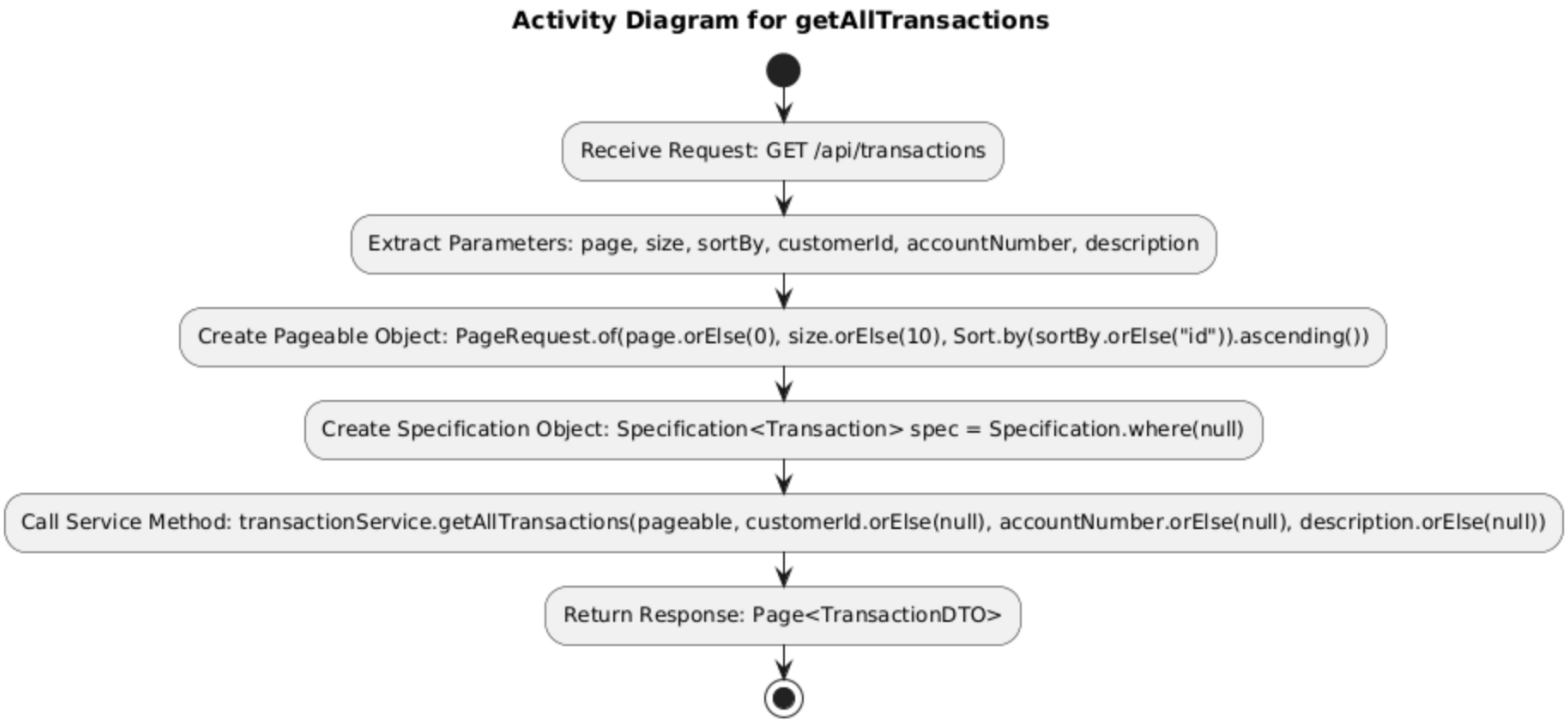
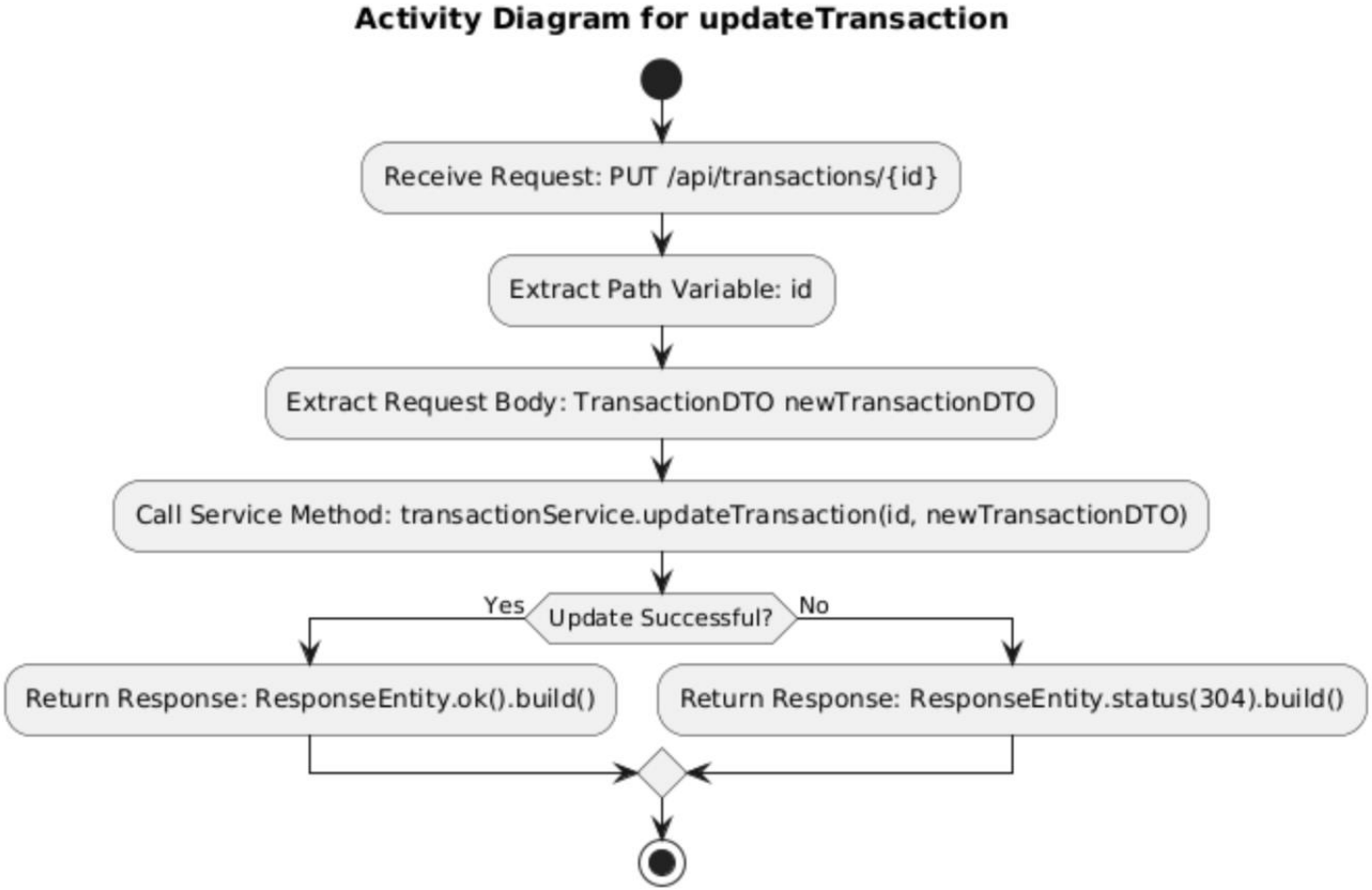
1. Class Diagram

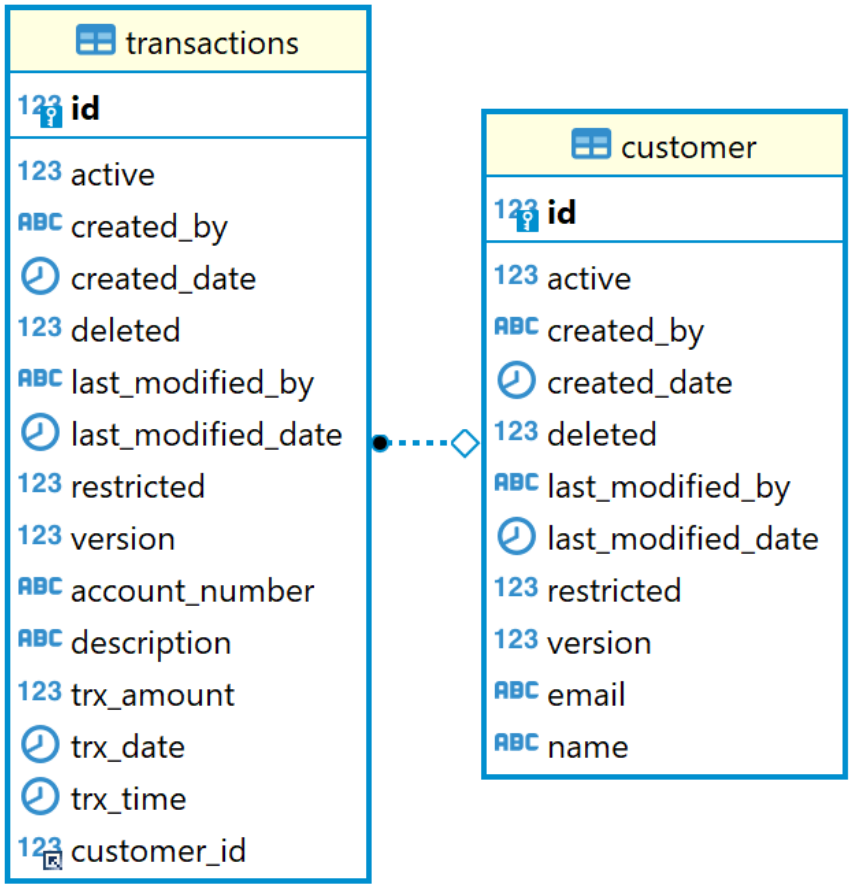


2. Activity Diagram




Activity Diagram for updateTransaction
Receive Request: PUT /api/transactions/{id}
Extract Path Variable: id
Extract Request Body: TransactionDTO newTransactionDTO
Call Service Method: transactionService.updateTransaction(id, newTransactionDTO)
Update Successful?
Yes
No
Return Response: ResponseEntity.ok().build()
Return Response: ResponseEntity.status(304).build()

3. Database diagram



## 4. Design Pattern

1. Repository Pattern:

   The Repository Pattern abstracts the data layer, providing a cleaner way to access and manipulate data. It separates the business logic from data access logic.

   Example: TransactionRepository and CustomerRepository are interfaces extending JpaRepository, providing a way to interact with the database.

2. DTO (Data Transfer Object) Pattern:

   DTOs are used to transfer data between different layers of an application. They help in encapsulating data and reducing the number of method calls.

   Example: TransactionDTO and CustomerDTO are used to transfer data between layers without exposing the entity classes directly.

3. Singleton Pattern:

   The Singleton Pattern ensures that a class has only one instance and provides a global point of access to it. In Spring, this is often managed by the framework itself

   Example: FileReadScheduler is a singleton component managed by Spring, ensuring only one instance is created.

4. Specification Pattern:

   The Specification Pattern is used to create reusable and combinable query criteria. It helps in building dynamic queries based on different conditions.

Example: TransactionSpecification provides a way to create dynamic queries based on different criteria.

5. Retry Pattern:

   The Retry Pattern is used to handle transient failures by retrying an operation a certain number of times before giving up. This is often used in network calls or database operations

   Example: @Retry annotation in TransactionService for updateTransaction method to handle transient failures.

## 5. Database scripts

```
CREATE DATABASE `mbb` /*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci */ /*!80016 DEFAULT ENCRYPTION='N' */;

– mbb.customer definition


CREATE TABLE `customer` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `active` bit(1) NOT NULL,
  `created_by` varchar(255) DEFAULT NULL,
  `created_date` datetime(6) DEFAULT NULL,
  `deleted` bit(1) NOT NULL,
  `last_modified_by` varchar(255) DEFAULT NULL,
```

```
  `last_modified_date` datetime(6) DEFAULT NULL,
  `restricted` bit(1) NOT NULL,
  `version` int NOT NULL,
  `email` varchar(255) DEFAULT NULL,
  `name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=223 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

– mbb.transactions definition

CREATE TABLE `transactions` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `active` bit(1) NOT NULL,
  `created_by` varchar(255) DEFAULT NULL,
  `created_date` datetime(6) DEFAULT NULL,
  `deleted` bit(1) NOT NULL,
  `last_modified_by` varchar(255) DEFAULT NULL,
  `last_modified_date` datetime(6) DEFAULT NULL,
```

```
  `restricted` bit(1) NOT NULL,
  `version` int NOT NULL,
  `account_number` varchar(255) DEFAULT NULL,
  `description` varchar(255) DEFAULT NULL,
  `trx_amount` double DEFAULT NULL,
  `trx_date` date DEFAULT NULL,
  `trx_time` time DEFAULT NULL,
  `customer_id` bigint DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `FK3xa9vicv9jgdsk2dl9qlhikn9` (`customer_id`),
  CONSTRAINT `FK3xa9vicv9jgdsk2dl9qlhikn9` FOREIGN KEY (`customer_id`) REFERENCES `customer` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=48 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```