**Exercise 8: Online Bookstore - Implementing CRUD Operations**

**Business Scenario:**

Implement Create, Read, Update, and Delete operations for the Book and Customer entities.

**1. CRUD Endpoints**

**BookController.java and CustomerController.java:**

These classes define RESTful endpoints for managing Book and Customer entities.

CRUD operations are implemented using HTTP methods:

* + **GET**: Retrieve a list of all books/customers or a specific entity by ID.
  + **POST**: Create a new book/customer.
  + **PUT**: Update an existing book/customer by ID.
  + **DELETE**: Remove a book/customer by ID.

The endpoints use @RequestMapping to define base URIs .

* **BookService.java and CustomerService.java:**

These services encapsulate business logic for CRUD operations.

Methods such as getAllBooks(), createBook(), updateBook(), and deleteBook() interact with the repository layer to perform database operations.

* The services leverage DTOs (Data Transfer Objects) to transfer data between the client and server, and they use mappers to convert between entities and DTOs.

**2. Validating Input Data**

**Validation in DTOs:**

* The BookDTO and CustomerDTO classes utilize validation annotations to ensure data integrity:
  + @NotBlank: Ensures that fields such as title, author, name, and email are not null or empty.
  + @Size: Restricts the size of fields like title and name to a specified range.
  + @Min: Ensures that numeric fields like price are above a certain minimum value.
  + @Email: Validates the format of the email field in CustomerDTO.

**Controller Level Validation:**

* In the controllers, the @Valid annotation is used to enforce validation when receiving request bodies. Invalid data will result in a 400 Bad Request response.

**3. Optimistic Locking**

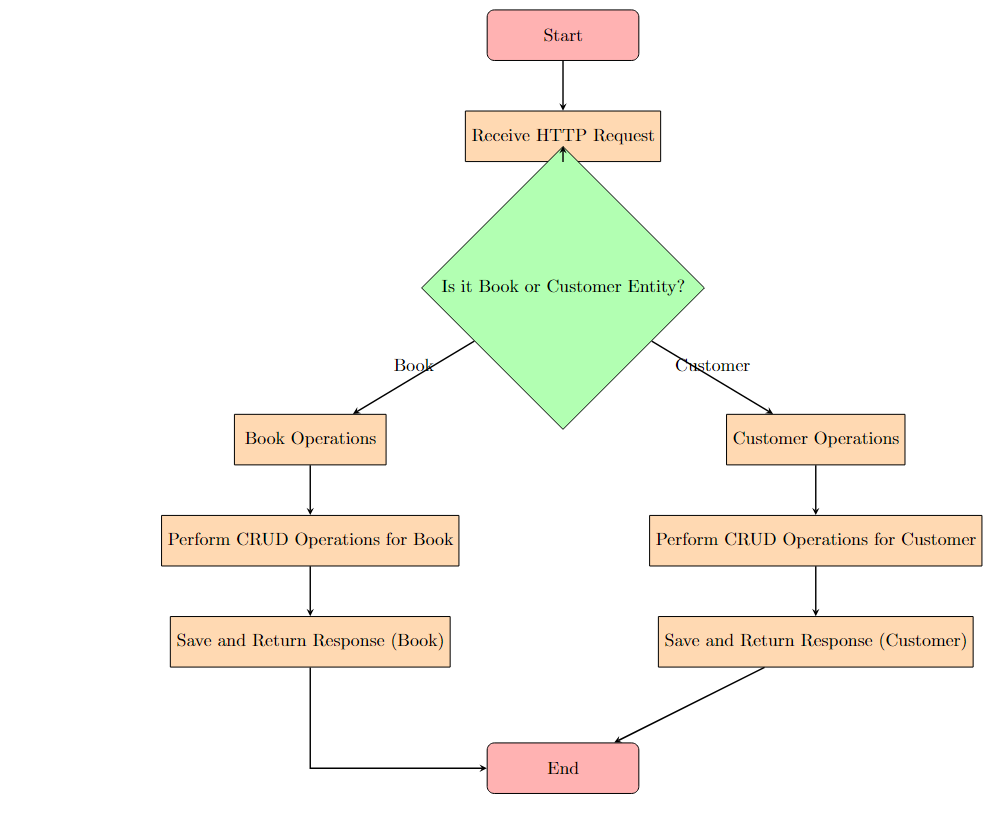
**Versioning in Entities:**

* The Book and Customer entity classes use the @Version annotation on a version field to implement optimistic locking.
* Optimistic locking prevents concurrent updates from overriding each other. When a record is updated, the version number is checked to ensure that no other update has occurred since the record was retrieved.

**Handling Optimistic Locking Exceptions:**

* If a conflict is detected (i.e., the version number has changed), a OptimisticLockException is thrown. This can be handled globally using @ControllerAdvice and custom exception handlers, ensuring that the client is informed of the conflict.

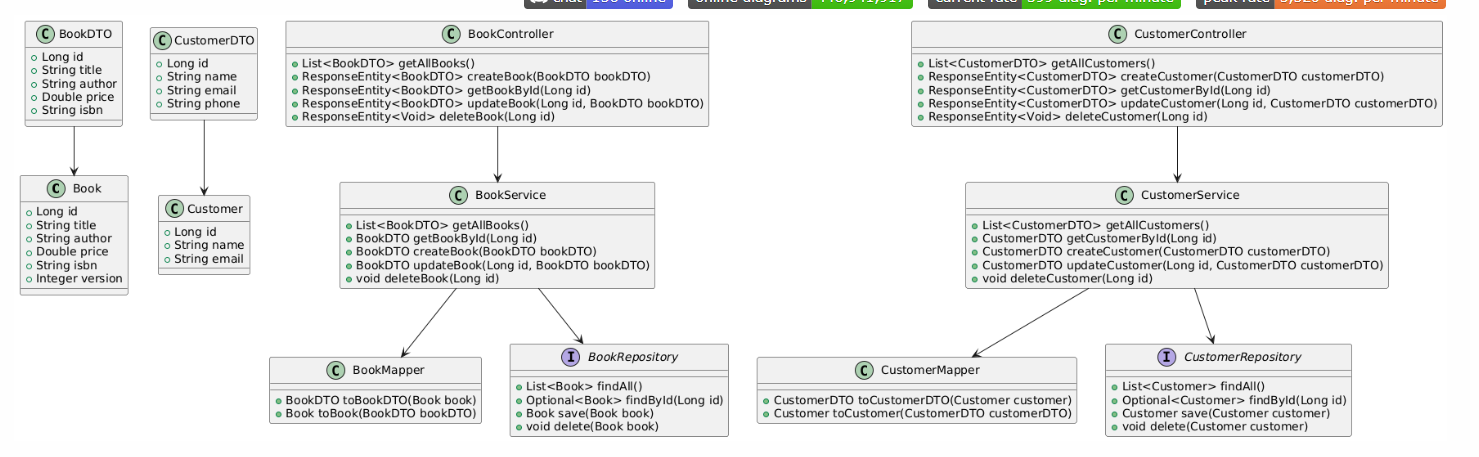
**FLOWCHART :**



**Explanation:**

1. **Start**: The flowchart begins with the "Start" node.
2. **Receive HTTP Request**: The system receives an HTTP request from the client.
3. **Decision: Is it Book or Customer Entity?**: A decision is made to determine whether the operation pertains to a Book or a Customer.
4. **Book Operations / Customer Operations**: Depending on the decision, the flow either handles Book operations or Customer operations.
5. **Perform CRUD Operations**: For the selected entity (Book or Customer), the system performs the necessary CRUD operations (Create, Read, Update, Delete).
6. **Save and Return Response**: After performing the CRUD operation, the system saves the result and returns an appropriate HTTP response.
7. **End**: The flowchart ends with the "End" node, signifying the completion of the operation.

**CLASS DIAGRAM :**



**Explanation:**

1. **Entities**:
   * Book and Customer are the core entities, representing the database models with fields like id, title, author, etc.
2. **DTOs**:
   * BookDTO and CustomerDTO are Data Transfer Objects, used to transfer data between the client and server without exposing the entity structure directly.
3. **Controllers**:
   * BookController and CustomerController manage HTTP requests for the Book and Customer entities, offering methods for CRUD operations.
4. **Services**:
   * BookService and CustomerService contain business logic and interact with repositories to perform CRUD operations.
5. **Mappers**:
   * BookMapper and CustomerMapper convert between DTOs and entities using methods like toBookDTO and toCustomerDTO.
6. **Repositories**:
   * BookRepository and CustomerRepository interface with the database using JPA methods like findAll, findById, save, and delete.
7. **Relationships**:
   * DTO classes depend on their respective entity classes.
   * Controllers depend on services, which in turn depend on repositories and mappers.