**Exercise 9: Online Bookstore - Understanding HATEOAS**

Business Scenario:

Enhance your REST API to follow HATEOAS principles for navigation through resources.

**Key Points and Code Explanation:**

1. **Spring HATEOAS Integration**:
   * **HATEOAS** (Hypermedia As The Engine Of Application State) enhances RESTful APIs by including hypermedia links in the responses. This allows clients to dynamically discover available actions.
   * The controllers (BookController and CustomerController) are modified to return RepresentationModel objects, which can include links.
2. **Adding Links to Resources**:

Each DTO (Data Transfer Object), such as BookDTO and CustomerDTO, extends RepresentationModel<T>, which is part of Spring HATEOAS. This enables the inclusion of hypermedia links within these objects.

1. **Hypermedia-Driven APIs**:

The idea behind hypermedia-driven APIs is that clients do not need to hard-code URLs for subsequent operations; instead, they can follow links provided in the API responses.

1. **DTO Classes**:

The DTO classes (BookDTO, CustomerDTO) extend RepresentationModel<T>. This makes it possible to add links directly to these objects, making them HATEOAS-compliant.

1. **Service Layer**:

The service layer (BookService, CustomerService) remains largely unchanged but is responsible for the core business logic. The integration of HATEOAS happens at the controller level, so the service layer focuses on data retrieval and manipulation.

1. **Global Exception Handling**:

A GlobalExceptionHandler is implemented to handle exceptions uniformly across the application. It catches generic exceptions and returns a 500 INTERNAL\_SERVER\_ERROR status with the exception message.

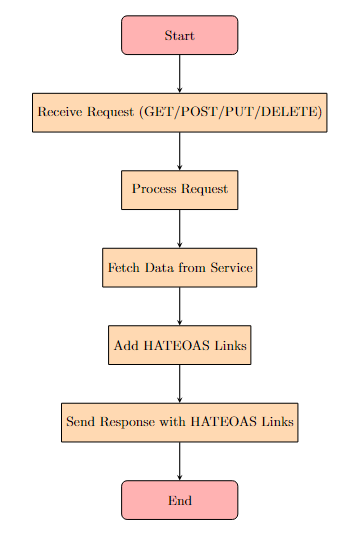
1. **RESTful Controllers**:

The BookController and CustomerController handle the main CRUD operations for books and customers. With the integration of HATEOAS, each resource (book or customer) now includes links to relevant actions, such as retrieving a single resource, updating it, or deleting it.

**Benefits of HATEOAS:**

* **Self-Discoverability**: Clients can navigate the API dynamically without needing prior knowledge of all the endpoints.
* **Reduced Coupling**: Clients depend on the links provided by the server, reducing the need for hardcoded URLs in client applications.
* **Enhanced API Usability**: With clear links and actions provided in the responses, the API becomes more intuitive and easier to consume.

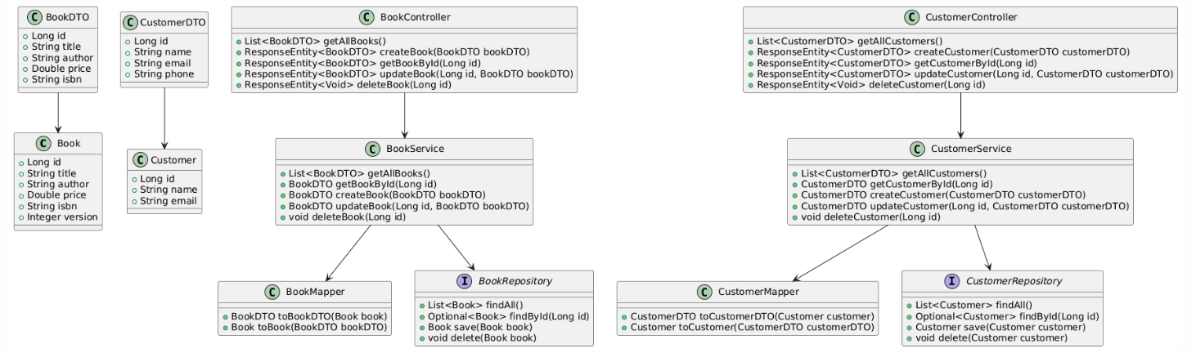
**FLOWCHART:**



**Explanation:**

1. **Start:** The process begins when the server starts listening for incoming HTTP requests on specific endpoints (e.g., /books or /customers).
2. **Receive Request (GET/POST/PUT/DELETE):** Depending on the request type (GET, POST, PUT, DELETE), the appropriate method in the controller (e.g., getAllBooks, createBook) is invoked.
3. **Process Request:** The request parameters or body are processed, validated, and prepared for further actions.
4. **Fetch Data from Service:** The controller interacts with the service layer (e.g., BookService, CustomerService) to fetch or manipulate data from the database.
5. **Add HATEOAS Links:** HATEOAS links are added to the response using WebMvcLinkBuilder. These links allow clients to navigate through related resources .
6. **Send Response with HATEOAS Links:** The response, now enriched with HATEOAS links, is sent back to the client.
7. **End:** The process ends after the response is sent, waiting for the next incoming request.

**CLASS DIAGRAM :**



**Explanation:**

1. **Controller Layer:**
   * **BookController** and **CustomerController** manage HTTP requests related to books and customers, respectively. They define methods like getAllBooks(), getBookById(), etc.
   * These controllers use the services (BookService, CustomerService) to handle business logic.
2. **Service Layer:**
   * **BookService** and **CustomerService** handle the core business logic for books and customers. They interact with the database via repositories (not shown here but assumed).
   * Methods like createBook() and updateCustomer() encapsulate the logic needed to manage these entities.
3. **Model Layer:**
   * **Book** and **Customer** represent the domain models/entities. These classes have attributes like id, title, author, price for Book and id, name, email, address for Customer.
4. **HATEOAS:**
   * **Link** represents a single link with a rel (relation) and href (URL).
   * **Resource<T>** is a generic class that wraps a model (Book, Customer) and provides a list of Link objects, enabling HATEOAS functionality.
5. **Relationships:**
   * **Controllers** use their respective services to fetch or manipulate data.
   * **Services** interact with the model layer to handle the data.
   * **Controllers** return Resource<T> objects containing the entity along with HATEOAS Link objects.