**Exercise 7: Online Bookstore - Introduction to Data Transfer Objects (DTOs)**

**Business Scenario:**

The goal of this exercise is to implement Data Transfer Objects (DTOs) to efficiently transfer data between the client and server in the context of an online bookstore. This approach will help in decoupling the domain entities from the data exposed by APIs, making the application more robust and flexible.

**Instructions:**

1. **Create DTOs:**
   * Define BookDTO and CustomerDTO classes to represent the data structure that will be transferred between the client and server.
   * The BookDTO class will include fields such as id, title, author, price, and isbn.
   * The CustomerDTO class will include fields such as name and emailid.
2. **Mapping Entities to DTOs:**
   * Use a mapping library such as **ModelMapper** to automatically map entity objects to DTOs and vice versa.
   * This simplifies the conversion between the entity objects and the data that will be sent over the network, ensuring consistency and reducing manual errors.
3. **Custom Serialization/Deserialization:**
   * Customize the JSON serialization and deserialization process using **Jackson** annotations.
   * In this exercise, the BookDTO class has a custom serialization for the price field to ensure it is formatted as a string with two decimal places.
   * Custom deserialization is also implemented for the price field to parse it from a string back to a double.

**Implementation:**

1. **DTO Classes:**
   * **BookDTO:**
     + Contains fields for id, title, author, price, and isbn.
     + Custom JSON serialization/deserialization is implemented for the price field to handle specific formatting requirements.
   * **CustomerDTO:**
     + Contains fields for name and emailid.
     + Uses basic getters and setters with no additional custom behavior.
2. **Mapping Configuration:**
   * **ModelMapper Bean:**
     + A ModelMapper bean is defined in the BookstoreApiApplication class to facilitate the mapping between entities and DTOs.
     + This bean is injected into services where mapping is required.
3. **Serialization Configuration:**
   * **SerializationConfig Class:**
     + The SerializationConfig class is configured to use custom serializers and deserializers for BookDTO using the ObjectMapper and SimpleModule from Jackson.
     + This ensures that the price field in BookDTO is serialized and deserialized according to the specified format.
4. **Controller Layer:**
   * **BookController and CustomerController:**
     + These controllers handle HTTP requests related to books and customers.
     + They use the service layer to interact with the database and the DTOs to communicate data with clients.
     + Custom headers and response statuses are managed to enhance the API's reliability and client interaction.

**Benefits of Using DTOs:**

* **Decoupling Entities from API Layer:** DTOs abstract the internal structure of the entities, providing a simplified and more stable interface for external clients.
* **Security:** Sensitive fields in entities can be omitted in DTOs, reducing the risk of exposing unnecessary data.
* **Performance Optimization:** DTOs can be tailored to contain only the necessary fields required by the client, reducing the amount of data transferred over the network.

**Conclusion:**

Implementing DTOs in the online bookstore application improves the separation of concerns, enhances security, and provides flexibility in data handling. The approach used in this exercise, including mapping with ModelMapper and custom serialization/deserialization, sets a strong foundation for maintaining a scalable and efficient API.