Structure of the order and order item

This documentation provides a detailed overview of the structure and relationships of the Order and Order Item entities in a Spring Boot application. It explains the theory behind their implementation and provides code snippets as hints to guide you in creating these entities.

Order Entity

The Order entity represents an order placed by a customer and contains information such as order ID, customer details, order items, and total price. Here's an example of the Order entity:

```
@Entity
@Table(name = "orders")
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "customer_id")
    private Customer customer;

    @OneToMany(mappedBy = "order", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<OrderItem> orderItems;

    @Column(name = "total_price")
    private BigDecimal totalPrice;

    // Constructors, getters, and setters
}
```

In the above code snippet, the Order entity is annotated with @Entity to map it to a database table named *"orders"*. The @Id and @GeneratedValue annotations specify the primary key ID and its generation strategy.

The customer attribute represents the association between the Order and Customer entities. It is annotated with *@ManyToOne* to indicate a many-to-one relationship with the Customer entity. The @JoinColumn annotation specifies the foreign key column in the *"orders"* table that references the *"customer_id"* column in the *"customers"* table.

The *orderItems* attribute represents the association between the Order and OrderItem entities. It is annotated with @OneToMany to indicate a one-to-many relationship with the OrderItem entity. The mappedBy attribute specifies the corresponding attribute in the OrderItem entity that maps this relationship.
The *cascade* attribute ensures that operations performed on the Order entity are cascaded to its associated OrderItem entities. The orphanRemoval attribute enables the removal of orphaned OrderItem entities when they are no longer associated with an Order. The "*totalPrice"* attribute represents the total price of the order.

Order Item Entity

The Order Item entity represents an individual item within an order and contains information such as the associated product, quantity, and price. Here's an example of the OrderItem entity:

```
@Entity
@Table(name = "order_items")
public class OrderItem {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "order_id")
    private Order order;

    @ManyToOne
    @JoinColumn(name = "product_id")
    private Product product;

    private int quantity;
    private BigDecimal price;

    // Constructors, getters, and setters
}
```

In the above code snippet, the OrderItem entity is annotated with @Entity to map it to a database table named "*order_items*". A few of the other key components associated with the code above for the ***OrderItem*** entity are as follows:

- The @Id and @GeneratedValue annotations specify the primary key ID and its generation strategy.
- The order attribute represents the association between the OrderItem and Order entities. It is annotated with @ManyToOne to indicate a many-to-one relationship with the Order entity.
- The @JoinColumn annotation specifies the foreign key column in the *"order_items"* table that references the "order_id" column in the "orders" table.

- The *product* attribute represents the association between the OrderItem and Product entities. It is annotated with @ManyToOne to indicate a many-to-one relationship with the Product entity.
- The @JoinColumn annotation specifies the foreign key column in the *"order_items"* table that references the "product_id" column in the "products" table.
- The *quantity* attribute represents the quantity of the product in the order item.
- The *price* attribute represents the price of a single product unit in the order item.

Dtos

Order Item Request

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class OrderItemRequest {

    private Long userId;
    private String address;
    private BigDecimal payment;
}
```

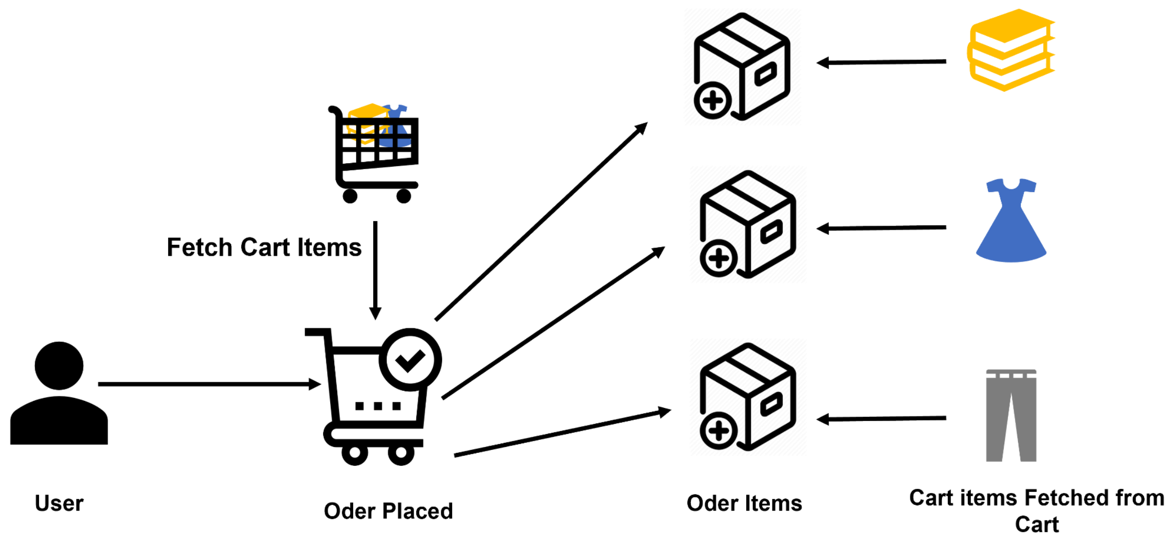
Order Item Response

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class OrderItemResponse {

    private Long id;
    private String createdDate;
    private String address;
    private String payment;
    private BigDecimal totalPrice;
}
```

Structure

Each user can place an order, and each Order contains a list of order Items, which are cart items fetched from the existing cart.
Once an order has been placed, it's logical to make the cart empty, and all the cart items now become oderItem, allowing the user to delete, update, and add a particular item from the order.



Usage

- With the Order and Order Item entities defined, you can use them in your Spring Boot application to manage orders and order items. This includes creating, retrieving, updating, and deleting orders and their associated items.
- You can customise and expand these code snippets based on your requirements and business logic. Additionally, you must create repositories, services, and controllers to implement the necessary CRUD operations for the Order and Order Item entities.
- Ensure you have properly configured and deployed your Spring Boot application to manage orders and order items.