Building REST API with Spring Boot - code snippets:

Building a REST API in Spring Boot typically involves the **Controller, Service, and Repository** layers to ensure separation of concerns. Within this document we take the example of an auction platform where we bid and place order. The following snippets illustrate the core components for both an ordering and an auction platform, focusing on the key actions of *placing an order* and *placing a bid*.

Here are examples of **Spring Boot code snippets** for the Controller, Service, and Repository layers, demonstrating the core logic for an **ordering and auction platform**. The examples follow the standard Controller-Service-Repository pattern for clear separation of concerns.

1. Ordering Platform Snippets: Placing an Order

This example demonstrates how a user might place a new order using a POST request to /api/orders.

Model (Entity) - Order.java

```
package com.example.ordering.model;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

import java.math.BigDecimal;

@Entity

@Table(name = "orders")

public class Order {

@Id

   @GeneratedValue(strategy = GenerationType.IDENTITY)

   private Long id;

   private String orderNumber;

   private BigDecimal totalAmount;

   private String status;

   private String productSku;

   private int quantity;

   // Getters, setters, and constructors (can use Lombok to reduce boilerplate)
```

```
    // Example getters/setters:
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    // ... other fields
}
```

Note: Use dependencies like Spring Data JPA and a database driver (e.g., MySQL, PostgreSQL) for data persistence.

Order Repository

This interface handles data access operations. Spring Data JPA provides basic CRUD functionality automatically.

```
package com.example.ordering.repository;
import com.example.ordering.model.Order;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
@Repository
public interface OrderRepository extends JpaRepository<Order, Long> {
    // Custom query methods can be added here if needed, e.g.,
    // Optional<Order> findByOrderNumber(String orderNumber);
}
```

Order Service

This layer contains the business logic, processing data and interacting with the repository.

```
package com.example.ordering.service;
import com.example.ordering.model.Order;
import com.example.ordering.repository.OrderRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
@Service
public class OrderService {
private final OrderRepository orderRepository;
@Autowired // Constructor injection is preferred
public OrderService(OrderRepository orderRepository) {
```

```
        this.orderRepository = orderRepository;
    }
    public Order placeOrder(Order order) {
        // Add business logic here, e.g.,
        // - Validate inventory
        // - Calculate total amount
        order.setStatus("CREATED");
        return orderRepository.save(order);
    }
    public Order getOrderDetails(Long orderId) {
        return orderRepository.findById(orderId)
                .orElseThrow(() -> new RuntimeException("Order not found")); // Use specific exceptions
      }
}
```

Order Controller

This layer handles HTTP requests and responses, interacting with the service layer.

```
package com.example.ordering.controller;
import com.example.ordering.model.Order;
import com.example.ordering.service.OrderService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
@RestController
@RequestMapping("/api/orders")
public class OrderController {
   private final OrderService orderService;
   @Autowired
   public OrderController(OrderService orderService) {
```

```
        this.orderService = orderService;
    }
    @PostMapping // Handles POST /api/orders requests
    public ResponseEntity<Order> createOrder(@RequestBody Order orderDetails) {
        Order createdOrder = orderService.placeOrder(orderDetails);
        return new ResponseEntity<>(createdOrder, HttpStatus.CREATED);
    }
    @GetMapping("/{id}") // Handles GET /api/orders/{id} requests
    public ResponseEntity<Order> getOrder(@PathVariable Long id) {
        Order order = orderService.getOrderDetails(id);
        return ResponseEntity.ok(order);
    }
}
```

2. Auction Platform Snippets

This example demonstrates creating an auction item and placing a bid. The core structure is similar to the ordering system but with specific auction logic in the service layer.

Bid Service

The service layer manages the specific auction business logic (e.g., verifying bids are higher than the current price).

```
package com.example.auction.service;
import com.example.auction.model.Bid;
import com.example.auction.repository.BidRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.math.BigDecimal;
@Service
public class BidService {
    private final BidRepository bidRepository;
    // Assume an AuctionRepository or similar service to check current auction status
    @Autowired
```

```
    public BidService(BidRepository bidRepository) {
        this.bidRepository = bidRepository;
    }
    public Bid placeBid(Long auctionId, BigDecimal bidAmount, Long userId) {
        // Business logic for placing a bid:
        // 1. Check if auction is active and not ended
        // 2. Check if bidAmount is higher than current highest bid/starting price
        // 3. Associate bid with auction and user
        // 4. Update the auction's current highest price

        Bid newBid = new Bid(/* set properties */);
        // ... logic implementation ...
        return bidRepository.save(newBid);
    }
}
```

Bid Controller

The controller exposes the endpoint for placing a bid on a specific auction item.

```
package com.example.auction.controller;
import com.example.auction.model.Bid;
import com.example.auction.service.BidService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
@RestController
@RequestMapping("/api/auctions/{auctionId}/bids")
public class BidController {
    private final BidService bidService;
```

```
    @Autowired
    public BidController(BidService bidService) {
        this.bidService = bidService;
    }
    @PostMapping
    // Example of handling a bid placement request
    public ResponseEntity<Bid> placeBid(@PathVariable Long auctionId, @RequestBody BidRequest bidRequest) {
        // In a real application, you'd get the userId from the authenticated user context
        Long currentUserId = 1L; // Placeholder
        Bid placedBid = bidService.placeBid(auctionId, bidRequest.getBidPrice(), currentUserId);
        return new ResponseEntity<>(placedBid, HttpStatus.CREATED);
    }
}
// A simple DTO (Data Transfer Object) for incoming bid requests
class BidRequest {
    private BigDecimal bidPrice;
    // Getter/Setter
    public BigDecimal getBidPrice() { return bidPrice; }
    public void setBidPrice(BigDecimal bidPrice) { this.bidPrice = bidPrice; }
}
```