
--- MAIN

```
package com.feast.server_main;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.boot.autoconfigure.security.servlet.SecurityAutoConfiguration;

@SpringBootApplication
public class ServerMainApplication {

	public static void main(String[] args) {
		SpringApplication.run(ServerMainApplication.class, args);
	}

}
```

---------- MAIN CONFIG

```
package com.feast.server_main.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
```

```
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

import java.util.Arrays;

@Configuration
public class SecurityConfig implements WebMvcConfigurer {

  @Bean
  public BCryptPasswordEncoder bCryptPasswordEncoder() {
    return new BCryptPasswordEncoder();
  }

  @Override
  public void addCorsMappings(CorsRegistry registry) {
    registry.addMapping("/**")
        .allowedOrigins("http://127.0.0.1:5500")
        .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
```

```
            .allowedHeaders("*")
            .allowCredentials(true)
            .maxAge(3600);
    }
}
```

----- AUTHCONTROLLER

```
package com.feast.server_main.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.feast.server_main.dto.UserDTO;
import com.feast.server_main.model.User;
import com.feast.server_main.service.AuthService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.webjars.NotFoundException;
```

```
@RestController
@CrossOrigin("http://127.0.0.1:5500")
public class AuthController {

  private static final Logger logger = LoggerFactory.getLogger(AuthController.class);

  @Autowired
  private AuthService authService;

  @Autowired
  private BCryptPasswordEncoder bCryptPasswordEncoder;

  public AuthController(AuthService authService, BCryptPasswordEncoder
bCryptPasswordEncoder) {
    this.authService = authService;
    this.bCryptPasswordEncoder = bCryptPasswordEncoder;
  }

  @PostMapping("/login")
  public ResponseEntity<UserDTO> login(@RequestBody User user) {
    logger.info("Received login request for email: {}", user.getEmail());
    try {
      UserDTO loggedInUser = authService.login(user);
      logger.info("Login successful for user: {}", user.getEmail());
      return ResponseEntity.ok(loggedInUser);
    } catch (NotFoundException e) {
```

```
            logger.warn("Login failed: User not found for email: {}", user.getEmail());
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(null); // Or .build()
        }
    }

    @PostMapping("/signup")
    public ResponseEntity<User> signup(@RequestBody User user) {
        try {
            User signedUpUser = authService.signup(user);
            logger.info("Signup successful for user: {}", signedUpUser.getEmail());
            return ResponseEntity.status(HttpStatus.CREATED).body(signedUpUser); // Use 201 Created
        } catch (IllegalArgumentException e) {
            logger.error("Signup failed: {}", e.getMessage());
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null); // Or .build()
        }
    }
}
```

---------- RESTAURANT CONTROLLER

```
package com.feast.server_main.controller;

import java.util.List;
import java.util.Map;
import java.util.Optional;
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.server.ResponseStatusException;

import com.feast.server_main.dto.FoodItemDTO;
import com.feast.server_main.dto.ResFoodItemDTO;
import com.feast.server_main.dto.RestaurantOrderStatusDTO;
import com.feast.server_main.dto.UpdateOrderStatusRequestDTO;
import com.feast.server_main.dto.UserDTO;
import com.feast.server_main.model.FoodItem;
import com.feast.server_main.model.Restaurant;
import com.feast.server_main.model.RestaurantOrderStatus;
import com.feast.server_main.model.User;
```

```
import com.feast.server_main.service.*;
import com.feast.server_main.response.StandardResponse;

@RestController
@RequestMapping("/restaurant")
@CrossOrigin("http://127.0.0.1:5500")
public class RestaurantController {

	@Autowired
	private RestaurantService restaurantService;

	private static final Logger logger =
LoggerFactory.getLogger(RestaurantController.class);

	@PostMapping("/create")
	public ResponseEntity<StandardResponse<UserDTO>> createRestaurant(
			@RequestParam("userId") Integer userId,
			@RequestParam("restaurantName") String restaurantName,
			@RequestParam("restaurantAddress") String restaurantAddress,
			@RequestParam("cuisine") String cuisine,
			@RequestParam("ownerName") String ownerName) {
		logger.info("Creating restaurant for userId: {}", userId);
		try {
			UserDTO user =
restaurantService.createRestaurantForExistingUser(userId, restaurantName,
restaurantAddress,
cuisine, ownerName);
```

```
                logger.info("Restaurant created successfully for userId: {}", userId);
                return new ResponseEntity<>(
                                new StandardResponse<>(HttpStatus.CREATED.value(), "Restaurant created successfully",user),
                                HttpStatus.CREATED);
            } catch (IllegalArgumentException e) {
                logger.error("Error creating restaurant: {}", e.getMessage());
                throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage());
            } catch (IllegalStateException e) {
                logger.error("Error creating restaurant: {}", e.getMessage());
                throw new ResponseStatusException(HttpStatus.CONFLICT, e.getMessage());
            } catch (Exception e) {
                logger.error("Error creating restaurant: {}", e.getMessage());
                throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "Failed to create restaurant", e);
            }
        }

        @GetMapping("/check")
        public ResponseEntity<StandardResponse<HasRestaurantResponse>> hasRestaurant(@RequestParam("userId") Integer userId) {
            logger.info("Checking if user has restaurant. userId: {}", userId);
            try {
                boolean hasRestaurant = restaurantService.hasRestaurant(userId);
```

```
                HasRestaurantResponse response = new
HasRestaurantResponse(hasRestaurant);

                logger.info("User has restaurant: {}, userId: {}", hasRestaurant, userId);

                return new ResponseEntity<>(

                                new StandardResponse<>(HttpStatus.OK.value(),
"Restaurant status checked", response), HttpStatus.OK);

        } catch (Exception e) {

                logger.error("Error checking restaurant status: {}", e.getMessage());

                throw new
ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "Failed to check
restaurant status", e);

        }

    }


    // Inner class to represent the response

    static class HasRestaurantResponse {

        private boolean hasRestaurant;


        public HasRestaurantResponse(boolean hasRestaurant) {

            this.hasRestaurant = hasRestaurant;

        }


        public boolean isHasRestaurant() {

            return hasRestaurant;

        }

    }
```

```
    @GetMapping("/restaurantId")
    public ResponseEntity<StandardResponse<Integer>>
getRestaurantId(@RequestParam("userId") Integer userId) {
        logger.info("Getting restaurant ID for userId: {}", userId);
        try {
            Optional<Integer> restaurantId =
restaurantService.getRestaurantIdByUserId(userId);
            if (restaurantId.isPresent()) {
                logger.info("Restaurant ID found: {} for userId: {}",
restaurantId.get(), userId);
                return new ResponseEntity<>(
                        new
StandardResponse<>(HttpStatus.OK.value(), "Restaurant ID retrieved successfully",
restaurantId.get()),
                        HttpStatus.OK);
            } else {
                logger.warn("Restaurant ID not found for userId: {}", userId);
                throw new
ResponseStatusException(HttpStatus.NOT_FOUND, "Restaurant ID not found");
            }
        } catch (Exception e) {
            logger.error("Error getting restaurant ID: {}", e.getMessage());
            throw new
ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "Failed to get
restaurant ID", e);
        }
    }
```

```
    @PostMapping("/addItem")
    public ResponseEntity<StandardResponse<FoodItemDTO>>
addFoodItem(@RequestBody FoodItem foodItem) {
        logger.info("Adding food item: {}", foodItem.getFoodName());
        try {
            FoodItemDTO item = restaurantService.addFoodItem(foodItem);
            logger.info("Food item added successfully: {}", item.getFoodName());
            return new ResponseEntity<>(
                    new
StandardResponse<>(HttpStatus.CREATED.value(), "Food item added successfully", item),
                    HttpStatus.CREATED);
        } catch (Exception e) {
            logger.error("Error adding food item: {}", e.getMessage());
            throw new
ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "Failed to add food
item", e);
        }
    }

    @GetMapping("/items")
    public ResponseEntity<StandardResponse<List<FoodItemDTO>>> getAllItems() {
        logger.info("Getting all food items");
        try {
            List<FoodItemDTO> items = restaurantService.getAllItems();
            logger.info("Retrieved {} food items", items.size());
            return new ResponseEntity<>(new
StandardResponse<>(HttpStatus.OK.value(), "All food items retrieved", items),
```

```
                HttpStatus.OK);
        } catch (Exception e) {
            logger.error("Error getting all food items: {}", e.getMessage());
            throw new
ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "Failed to retrieve food
items", e);
        }

    }

    @GetMapping("/{restaurantId}/food-items")
    public ResponseEntity<StandardResponse<List<ResFoodItemDTO>>>
getFoodItemsByRestaurant(
            @PathVariable Integer restaurantId) {
        logger.info("Getting food items for restaurantId: {}", restaurantId);
        try {
            List<ResFoodItemDTO> foodItem =
restaurantService.getFoodItemsByRestaurant(restaurantId);
            logger.info("Retrieved {} food items for restaurantId: {}",
foodItem.size(), restaurantId);
            return new ResponseEntity<>(
                    new StandardResponse<>(HttpStatus.OK.value(), "Food
items retrieved successfully", foodItem),
                    HttpStatus.OK);
        } catch (Exception e) {
            logger.error("Error getting food items by restaurant: {}",
e.getMessage());
```

```
                throw new
ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
                        "Failed to retrieve food items by restaurant", e);
            }
        }

        @PutMapping("/food-item/{foodItemId}")
    public ResponseEntity<StandardResponse<ResFoodItemDTO>> updateFoodItem(
        @PathVariable Integer foodItemId,
        @RequestBody FoodItem foodItem) {
      logger.info("Updating food item with ID: {}", foodItemId);
      try {
        FoodItemDTO updatedFoodItem = restaurantService.updateFoodItem(foodItemId,
foodItem);
        ResFoodItemDTO foodItemDTO =
restaurantService.mapToResFoodDto(updatedFoodItem);
        logger.info("Food item updated successfully: {}", updatedFoodItem.getFoodName());
        return new ResponseEntity<>(
            new StandardResponse<>(HttpStatus.OK.value(), "Food item updated",
foodItemDTO),
            HttpStatus.OK);
      } catch (IllegalArgumentException e) {
        logger.warn("Food item not found for update. ID: {}", foodItemId);
        throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Food item not
found", e);
      }
       catch (Exception e) {
```

```
        logger.error("Error updating food item: {}", e.getMessage());
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
"Failed to update food item", e);
    }
  }

  @DeleteMapping("/food-item/{foodItemId}")
  public ResponseEntity<StandardResponse<Void>> deleteFoodItem(@PathVariable
Integer foodItemId) {
    logger.info("Deleting food item with ID: {}", foodItemId);
    try {
        restaurantService.deleteFoodItem(foodItemId);
        logger.info("Food item deleted successfully. ID: {}", foodItemId);
        return new ResponseEntity<>(
            new StandardResponse<>(HttpStatus.NO_CONTENT.value(), "Food item deleted
successfully", null),
            HttpStatus.NO_CONTENT);
    } catch (IllegalArgumentException e) {
        logger.warn("Food item not found for deletion. ID: {}", foodItemId);
        throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Food item not
found", e);
    }catch (Exception e) {
        logger.error("Error deleting food item: {}", e.getMessage());
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
"Failed to delete food item", e);
    }
  }
```

```
    // Get order status by order ID
    @GetMapping("/order/{orderId}/status")
    public ResponseEntity<StandardResponse<RestaurantOrderStatusDTO>>
getOrderStatusByOrderId(
            @PathVariable Integer orderId) {
        logger.info("Getting order status for order ID: {}", orderId);
        try {
            RestaurantOrderStatusDTO orderStatus =
restaurantService.getOrderStatusByOrderId(orderId);
            if (orderStatus != null) {
                logger.info("Order status found for order ID {}: {}", orderId,
orderStatus.getStatus());
                return new ResponseEntity<>(
                        new
StandardResponse<>(HttpStatus.OK.value(), "Order status retrieved", orderStatus),
HttpStatus.OK);
            } else {
                logger.warn("Order status not found for order ID: {}", orderId);
                throw new
ResponseStatusException(HttpStatus.NOT_FOUND, "Order status not found");
            }
        } catch (Exception e) {
            logger.error("Error getting order status: {}", e.getMessage());
            throw new
ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "Failed to retrieve order
status", e);
        }
```

```
    }

    @GetMapping("/order/{restaurantId}/statuses")
    public ResponseEntity<StandardResponse<List<RestaurantOrderStatusDTO>>>
getAllOrderStatusesByRestaurantId(
            @PathVariable Integer restaurantId) {
        logger.info("Getting all order statuses for restaurant ID: {}", restaurantId);
        try {
            List<RestaurantOrderStatusDTO> orderStatuses = restaurantService
                    .getAllOrderStatusesByRestaurantId(restaurantId);
            logger.info("Retrieved {} order statuses for restaurant ID: {}",
orderStatuses.size(), restaurantId);
            return new ResponseEntity<>(
                    new StandardResponse<>(HttpStatus.OK.value(),
"Order statuses retrieved", orderStatuses), HttpStatus.OK);
        } catch (Exception e) {
            logger.error("Error getting order statuses: {}", e.getMessage());
            throw new
ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
                    "Failed to retrieve order statuses", e);
        }
    }

    @PutMapping("/order")
    public ResponseEntity<StandardResponse<RestaurantOrderStatusDTO>>
updateOrderStatus(
        @RequestBody UpdateOrderStatusRequestDTO requestBody) {
```

```
        logger.info("Updating order status with request body: {}", requestBody);
        try {

            Integer orderId = null;
            if (requestBody.getOrder() != null) {
                orderId = requestBody.getOrder().getOrderId();
            }

            String status = requestBody.getStatus();

            Integer restaurantId = null;
            if (requestBody.getRestaurant() != null) {
                restaurantId = requestBody.getRestaurant().getRestaurantId();
            }

            if (orderId == null || status == null || restaurantId == null) {
                logger.error("Missing or invalid required parameters in the request body.");
                throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Missing or invalid orderId, status, or restaurantId.");
            }

            RestaurantOrderStatusDTO updatedOrderStatus = restaurantService.updateOrderStatus(orderId, status, restaurantId);
            logger.info("Order status updated successfully for order ID: {}", orderId);
            return new ResponseEntity<>(
                    new StandardResponse<>(HttpStatus.OK.value(), "Order status updated", updatedOrderStatus), HttpStatus.OK);
```

```
        } catch (IllegalArgumentException e) {
            logger.error("Error updating order status: {}", e.getMessage());
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST,
e.getMessage(), e);
        } catch (Exception e) {
            logger.error("Error updating order status: {}", e.getMessage());
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
"Failed to update order status", e);
        }
    }


    // Get all order status
    @GetMapping("/order/status/all")
    public ResponseEntity<StandardResponse<List<RestaurantOrderStatusDTO>>>
getAllOrderStatus() {
        logger.info("Getting all order statuses");
        try {
            List<RestaurantOrderStatusDTO> allOrderStatus =
restaurantService.getAllOrderStatus();
            logger.info("Retrieved {} order statuses in total.", allOrderStatus.size());
            return new ResponseEntity<>(
                    new StandardResponse<>(HttpStatus.OK.value(), "All
order statuses retrieved", allOrderStatus),
                    HttpStatus.OK);
        } catch (Exception e) {
            logger.error("Error getting all order statuses: {}", e.getMessage());
```

```
                    throw new
ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "Failed to retrieve all
order statuses",
                                        e);
                }
        }
}
```

---------- USER CONTROLLER

```
package com.feast.server_main.controller;


import java.util.List;


import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
```

```
import org.springframework.web.server.ResponseStatusException;

import com.feast.server_main.dto.CusOrderDTO;
import com.feast.server_main.dto.FoodItemDTO;
import com.feast.server_main.dto.OrderDTO;
import com.feast.server_main.dto.ResFoodItemDTO;
import com.feast.server_main.dto.RestaurantDTO;
import com.feast.server_main.dto.RestaurantOrderStatusDTO;
import com.feast.server_main.dto.UserDTO;
import com.feast.server_main.model.Order;
import com.feast.server_main.model.RestaurantOrderStatus;
import com.feast.server_main.response.StandardResponse;
import com.feast.server_main.service.UserService;

@RestController
@RequestMapping("/customer")
@CrossOrigin("http://127.0.0.1:5500")
public class UserController {

   @Autowired
   private UserService userService;
   private static final Logger logger = LoggerFactory.getLogger(UserController.class);

   @GetMapping("/food-items")
   public ResponseEntity<StandardResponse<List<ResFoodItemDTO>>> getAllFoodItems()
{
```

```
        logger.info("Getting all food items for customer.");
        try {
            List<ResFoodItemDTO> foodItems = userService.getAllFoodItems();
            logger.info("Retrieved {} food items.", foodItems.size());
            return new ResponseEntity<>(new StandardResponse<>(HttpStatus.OK.value(),
"Success", foodItems), HttpStatus.OK);
        } catch (Exception e) {
            logger.error("Error retrieving food items: {}", e.getMessage());
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
"Failed to retrieve food items", e);
        }
    }

    @GetMapping("/food-items/{foodItemId}/restaurant")
    public ResponseEntity<StandardResponse<Integer>>
getRestaurantIdByFoodItemId(@PathVariable Integer foodItemId) {
        logger.info("Getting restaurant ID for food item ID: {}", foodItemId);
        try {
            Integer restaurantId = userService.getRestaurantIdByFoodItemId(foodItemId);
            logger.info("Retrieved restaurant ID: {} for food item ID: {}", restaurantId, foodItemId);
            return new ResponseEntity<>(new StandardResponse<>(HttpStatus.OK.value(),
"Success", restaurantId), HttpStatus.OK);
        } catch (IllegalArgumentException e) {
            logger.warn("Food item with ID {} not found.", foodItemId);
            return new ResponseEntity<>(new
StandardResponse<>(HttpStatus.NOT_FOUND.value(), "Food item not found", null),
HttpStatus.NOT_FOUND);
        } catch (Exception e) {
```

```
        logger.error("Error retrieving restaurant ID for food item ID {}: {}", foodItemId,
e.getMessage());
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
"Failed to retrieve restaurant ID", e);
    }
  }

  @GetMapping("/restaurants")
  public ResponseEntity<StandardResponse<List<RestaurantDTO>>> getAllRestaurants() {
    logger.info("Getting all restaurants for customer.");
    try {
      List<RestaurantDTO> restaurants = userService.getAllRestaurants();
      logger.info("Retrieved {} restaurants.", restaurants.size());
      return new ResponseEntity<>(new StandardResponse<>(HttpStatus.OK.value(),
"Success", restaurants), HttpStatus.OK);
    } catch (Exception e) {
      logger.error("Error retrieving restaurants: {}", e.getMessage());
      throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
"Failed to retrieve restaurants", e);
    }
  }

  @GetMapping("/restaurants/{restaurantId}/food-items")
  public ResponseEntity<StandardResponse<List<ResFoodItemDTO>>>
getFoodItemsByRestaurant(
      @PathVariable Integer restaurantId) {
    logger.info("Getting food items for restaurantId: {}", restaurantId);
```

```
    try {
        List<ResFoodItemDTO> foodItems =
userService.getFoodItemsByRestaurant(restaurantId);
        logger.info("Retrieved {} food items for restaurantId: {}", foodItems.size(),
restaurantId);
        return new ResponseEntity<>(new StandardResponse<>(HttpStatus.OK.value(),
"Success", foodItems), HttpStatus.OK);
    } catch (IllegalArgumentException e) {
        logger.error("Invalid argument for restaurantId: {}", restaurantId, e);
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage());
    } catch (Exception e) {
        logger.error("Error retrieving food items for restaurantId {}: {}", restaurantId,
e.getMessage());
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
"Failed to retrieve food items by restaurant", e);
    }
  }

  @PostMapping("/order")
  public ResponseEntity<StandardResponse<CusOrderDTO>> placeOrder(@RequestBody
OrderDTO orderDTO) {
    logger.info("Placing order: {}", orderDTO);
    try {
        CusOrderDTO placedOrder = userService.placeOrder(orderDTO);
        logger.info("Order placed successfully. Order ID: {}", placedOrder.getOrderId());
        return new ResponseEntity<>(
            new StandardResponse<>(HttpStatus.CREATED.value(), "Order placed
successfully", placedOrder),
```

```
            HttpStatus.CREATED);
    } catch (IllegalArgumentException e) {
      logger.error("Invalid order details: {}", orderDTO, e);
      throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage());
    } catch (Exception e) {
      logger.error("Error placing order: {}", e.getMessage());
      throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
"Failed to place order", e);
    }
  }

  @GetMapping("/cart")
  public ResponseEntity<StandardResponse<List<OrderDTO>>>
getCartByOrderId(@RequestParam("userId") Integer userId) {
    logger.info("Getting cart for userId: {}", userId);
    try {
      List<OrderDTO> cart = userService.getCartByUserId(userId);
      logger.info("Retrieved cart for userId {}. Cart size: {}", userId, cart.size());
      return new ResponseEntity<>(new StandardResponse<>(HttpStatus.OK.value(),
"Success", cart), HttpStatus.OK);
    } catch (IllegalArgumentException e) {
      logger.error("Invalid userId: {}", userId, e);
      throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage());
    } catch (Exception e) {
      logger.error("Error retrieving cart for userId {}: {}", userId, e.getMessage());
      throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
"Failed to retrieve cart", e);
```

```
    }
  }

  @PutMapping("/cart/{orderId}")
  public ResponseEntity<StandardResponse<OrderDTO>> updateOrder(@PathVariable Integer orderId,
      @RequestParam("userId") Integer userId, @RequestBody OrderDTO orderDTO) {
    logger.info("Updating orderId: {} for userId: {} with orderDTO: {}", orderId, userId, orderDTO);
    try {
      OrderDTO orderUpdated = userService.updateOrderQuantity(orderId, orderDTO.getQuantity(), userId);
      logger.info("Order updated successfully. Order ID: {}", orderId);
      return new ResponseEntity<>(new StandardResponse<>(HttpStatus.OK.value(), "Successfully updated", orderUpdated),
          HttpStatus.OK);
    } catch (IllegalArgumentException e) {
      logger.error("Invalid order update request. orderId: {}, userId: {}, orderDTO: {}", orderId, userId, orderDTO, e);
      throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage());
    } catch (Exception e) {
      logger.error("Error updating order. orderId: {}, userId: {}, orderDTO: {}", orderId, userId, orderDTO, e.getMessage());
      throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "Failed to update order", e);
    }
  }
```

```
  @DeleteMapping("/cart/clear")
  public ResponseEntity<StandardResponse<Void>> clearCart(@RequestParam("userId") Integer userId) {
    logger.info("Clearing cart for userId: {}", userId);
    try {
      userService.removeOrder(userId);
      logger.info("Cart cleared successfully for userId: {}", userId);
      return new ResponseEntity<>(
          new StandardResponse<>(HttpStatus.NO_CONTENT.value(), "Order deleted successfully", null),
          HttpStatus.NO_CONTENT);
    } catch (IllegalArgumentException e) {
      logger.error("Invalid userId: {}", userId, e);
      throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage());
    } catch (Exception e) {
      logger.error("Error clearing cart for userId: {}", userId, e.getMessage());
      throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "Failed to clear cart", e);
    }
  }


  @DeleteMapping("/item/clear/{orderId}")
  public ResponseEntity<StandardResponse<Void>> removeOrderItem(@PathVariable Integer orderId, @RequestParam("userId") Integer userId) {
    logger.info("Removing order item with orderId: {} for userId: {}", orderId, userId);
    try {
```

```
            userService.removeOrderItem(orderId, userId);
            logger.info("Order item removed successfully. Order ID: {}", orderId);
            return new ResponseEntity<>(new StandardResponse<>(HttpStatus.OK.value(),
"Successfully removed item", null), HttpStatus.OK);
        } catch (IllegalArgumentException e) {
            logger.error("Invalid request to remove order item. orderId: {}, userId: {}", orderId,
userId, e);
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage());
        } catch (Exception e) {
            logger.error("Error removing order item. orderId: {}, userId: {}", orderId, userId, e);
            throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
"Failed to remove order item", e);
        }
    }

    @GetMapping("/profile/{userId}")
    public ResponseEntity<StandardResponse<List<UserDTO>>>
getUserProfileByUserId(@PathVariable Integer userId) {
        logger.info("Getting user profile for userId: {}", userId);
        try {
            List<UserDTO> userProfile = userService.getUserProfileByUserId(userId);
            logger.info("User profile retrieved successfully for userId: {}", userId);
            return new ResponseEntity<>(new StandardResponse<>(HttpStatus.OK.value(),
"User profile retrieved successfully", userProfile), HttpStatus.OK);
        } catch (IllegalArgumentException e) {
            logger.error("Invalid userId: {}", userId, e);
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage());
```

```
    } catch (Exception e) {
        logger.error("Error retrieving user profile for userId {}: {}", userId, e.getMessage());
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
"Failed to get user profile", e);
    }
  }

  @PutMapping("/profile/edit/{userId}")
  public ResponseEntity<StandardResponse<UserDTO>>
updateUserProfile(@PathVariable Integer userId, @RequestBody UserDTO userDTO) {
    logger.info("Updating user profile for userId: {} with userDTO: {}", userId, userDTO);
    try {
        UserDTO updatedUser = userService.updateUserProfile(userId, userDTO);
        logger.info("User profile updated successfully for userId: {}", userId);
        return new ResponseEntity<>(new StandardResponse<>(HttpStatus.OK.value(),
"User profile updated", updatedUser), HttpStatus.OK);
    } catch (IllegalArgumentException e) {
        logger.error("Invalid user profile update request. userId: {}, userDTO: {}", userId,
userDTO, e);
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage());
    } catch (Exception e) {
        logger.error("Error updating user profile for userId {}: {}", userId, e.getMessage());
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
"Failed to update user profile", e);
    }
  }
```

```
   @PostMapping("/order/status")
   public ResponseEntity<StandardResponse<RestaurantOrderStatusDTO>>
updateOrderStatus(@RequestBody RestaurantOrderStatus orderStatusDTO) {
      logger.info("Updating order status: {}", orderStatusDTO);
      try {
         RestaurantOrderStatusDTO savedOrderStatus =
userService.updateOrderStatus(orderStatusDTO);
         logger.info("Order status updated successfully. Order ID: {}",
savedOrderStatus.getOrderDTO().getOrderId());
         return new ResponseEntity<>(new StandardResponse<>(HttpStatus.OK.value(),
"Order status updated", savedOrderStatus), HttpStatus.OK);
      } catch (IllegalArgumentException e) {
         logger.error("Invalid order status update request: {}", orderStatusDTO, e);
         throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage());
      } catch (Exception e) {
         logger.error("Error updating order status: {}", orderStatusDTO, e.getMessage());
         throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
"Failed to update order status", e);
      }
   }
}
```

-----CUSORDERDTO

```
package com.feast.server_main.dto;

import java.time.LocalDateTime;
```

```
public class CusOrderDTO {


   private Integer orderId;

   private ResFoodItemDTO resFoodItemTO;

   private Double totalPrice;

   private Integer quantity;

   private LocalDateTime date;

        public CusOrderDTO(Integer orderId, ResFoodItemDTO resFoodItemTO, Double
totalPrice, Integer quantity,

                        LocalDateTime date) {

                super();

                this.orderId = orderId;

                this.resFoodItemTO = resFoodItemTO;

                this.totalPrice = totalPrice;

                this.quantity = quantity;

                this.date = date;

        }

        public Integer getOrderId() {

                return orderId;

        }

        public void setOrderId(Integer orderId) {

                this.orderId = orderId;

        }

        public ResFoodItemDTO getResFoodItemTO() {

                return resFoodItemTO;
```

```
}
public void setResFoodItemTO(ResFoodItemDTO resFoodItemTO) {
	this.resFoodItemTO = resFoodItemTO;
}
public Double getTotalPrice() {
	return totalPrice;
}
public void setTotalPrice(Double totalPrice) {
	this.totalPrice = totalPrice;
}
public Integer getQuantity() {
	return quantity;
}
public void setQuantity(Integer quantity) {
	this.quantity = quantity;
}
public LocalDateTime getDate() {
	return date;
}
public void setDate(LocalDateTime date) {
	this.date = date;
}
```

}

----------FOODITEMDTO

```
package com.feast.server_main.dto;


public class FoodItemDTO {


   private Integer foodId;

   private String foodName;

   private String foodType;

   private String description;

   private Float price;

   private String imageURL;

   private Double rating;

   private RestaurantDTO restaurantDto;


        public FoodItemDTO(Integer foodId, String foodName, String foodType, String
description, Float price,

                        String imageURL, Double rating, RestaurantDTO restaurantDto) {

                super();

                this.foodId = foodId;

                this.foodName = foodName;

                this.foodType = foodType;

                this.description = description;

                this.price = price;

                this.imageURL = imageURL;
```

```
        this.rating = rating;
        this.restaurantDto = restaurantDto;
    }
    public Integer getFoodId() {
        return foodId;
    }
    public void setFoodId(Integer foodId) {
        this.foodId = foodId;
    }
    public String getFoodName() {
        return foodName;
    }
    public void setFoodName(String foodName) {
        this.foodName = foodName;
    }
    public String getFoodType() {
        return foodType;
    }
    public void setFoodType(String foodType) {
        this.foodType = foodType;
    }
    public String getDescription() {
        return description;
    }
    public void setDescription(String description) {
        this.description = description;
```

```
}
public Float getPrice() {
        return price;
}
public void setPrice(Float price) {
        this.price = price;
}
public String getImageURL() {
        return imageURL;
}
public void setImageURL(String imageURL) {
        this.imageURL = imageURL;
}
public Double getRating() {
        return rating;
}
public void setRating(Double rating) {
        this.rating = rating;
}
public RestaurantDTO getRestaurant() {
        return restaurantDto;
}
public void setRestaurant(RestaurantDTO restaurantDto) {
        this.restaurantDto = restaurantDto;
}
```

```
}
```

----------ORDERDTO

```
package com.feast.server_main.dto;


import java.time.LocalDateTime;


import com.feast.server_main.model.FoodItem;
import com.feast.server_main.model.User;



public class OrderDTO {


   private Integer orderId;
   private UserDTO user;
   private FoodItemDTO foodItem;
   private Double totalPrice;
   private Integer quantity;
   private LocalDateTime date;


        public OrderDTO(Integer orderId, UserDTO user, FoodItemDTO foodItem, Double
totalPrice, Integer quantity,
                        LocalDateTime date) {
                super();
                this.orderId = orderId;
                this.user = user;
                this.foodItem = foodItem;
```

```
        this.totalPrice = totalPrice;
        this.quantity = quantity;
        this.date = date;
    }
    public Integer getOrderId() {
        return orderId;
    }
    public void setOrderId(Integer orderId) {
        this.orderId = orderId;
    }
    public UserDTO getUser() {
        return user;
    }
    public void setUser(UserDTO user) {
        this.user = user;
    }
    public FoodItemDTO getFoodItem() {
        return foodItem;
    }
    public void setFoodItem(FoodItemDTO foodItem) {
        this.foodItem = foodItem;
    }
    public Double getTotalPrice() {
        return totalPrice;
    }
    public void setTotalPrice(Double totalPrice) {
```

```
            this.totalPrice = totalPrice;
        }
        public Integer getQuantity() {
            return quantity;
        }
        public void setQuantity(Integer quantity) {
            this.quantity = quantity;
        }
        public LocalDateTime getDate() {
            return date;
        }
        public void setDate(LocalDateTime date) {
            this.date = date;
        }


}
----------RESFOODITEMDTO
package com.feast.server_main.dto;

public class ResFoodItemDTO {

   private Integer foodId;
   private String foodName;
   private String foodType;
   private String description;
```

```
    private Float price;
    private String imageURL;
    private Double rating;

    public ResFoodItemDTO(Integer foodId, String foodName, String foodType, String
description, Float price,
                    String imageURL, Double rating) {
        super();
        this.foodId = foodId;
        this.foodName = foodName;
        this.foodType = foodType;
        this.description = description;
        this.price = price;
        this.imageURL = imageURL;
        this.rating = rating;
    }

    public Integer getFoodId() {
        return foodId;
    }

    public void setFoodId(Integer foodId) {
        this.foodId = foodId;
    }

    public String getFoodName() {
```

```
        return foodName;
}

public void setFoodName(String foodName) {
        this.foodName = foodName;
}

public String getFoodType() {
        return foodType;
}

public void setFoodType(String foodType) {
        this.foodType = foodType;
}

public String getDescription() {
        return description;
}

public void setDescription(String description) {
        this.description = description;
}

public Float getPrice() {
        return price;
}
```

```
    public void setPrice(Float price) {
        this.price = price;
    }

    public String getImageURL() {
        return imageURL;
    }

    public void setImageURL(String imageURL) {
        this.imageURL = imageURL;
    }

    public Double getRating() {
        return rating;
    }

    public void setRating(Double rating) {
        this.rating = rating;
    }

}
```

----------RESTAURANT DTO

```
package com.feast.server_main.dto;
```

```
public class RestaurantDTO {

   private Integer restaurantId;
   private String restaurantName;
   private String address;
   private String cuisine;
   private String ownerName;

   public RestaurantDTO() {

   }
        public RestaurantDTO(Integer restaurantId, String restaurantName, String address,
String cuisine, String ownerName) {
                super();
                this.restaurantId = restaurantId;
                this.restaurantName = restaurantName;
                this.address = address;
                this.cuisine = cuisine;
                this.ownerName = ownerName;
        }

        public Integer getRestaurantId() {
                return restaurantId;
        }
```

```
public void setRestaurantId(Integer restaurantId) {
        this.restaurantId = restaurantId;
}

public String getRestaurantName() {
        return restaurantName;
}

public void setRestaurantName(String restaurantName) {
        this.restaurantName = restaurantName;
}

public String getAddress() {
        return address;
}

public void setAddress(String address) {
        this.address = address;
}

public String getCuisine() {
        return cuisine;
}

public void setCuisine(String cuisine) {
```

```
            this.cuisine = cuisine;
        }

        public String getOwnerName() {
            return ownerName;
        }

        public void setOwnerName(String ownerName) {
            this.ownerName = ownerName;
        }


}
```

----------RES ORDER STATUS DTO

```
package com.feast.server_main.dto;

import java.time.LocalDateTime;

import com.feast.server_main.model.Order;
import com.feast.server_main.model.Restaurant;

public class RestaurantOrderStatusDTO {

        private Integer restaurantOrderStatusId;
   private RestaurantDTO restaurantDTO;
   private OrderDTO orderDTO;
```

```
    private String status;
    private LocalDateTime orderedAt;

    public RestaurantOrderStatusDTO() {

    }

        public RestaurantOrderStatusDTO(Integer restaurantOrderStatusId, RestaurantDTO
restaurantDTO, OrderDTO orderDTO,
                        String status, LocalDateTime orderedAt) {
                super();
                this.restaurantOrderStatusId = restaurantOrderStatusId;
                this.restaurantDTO = restaurantDTO;
                this.orderDTO = orderDTO;
                this.status = status;
                this.orderedAt = orderedAt;
        }

        public Integer getRestaurantOrderStatusId() {
                return restaurantOrderStatusId;
        }

        public void setRestaurantOrderStatusId(Integer restaurantOrderStatusId) {
                this.restaurantOrderStatusId = restaurantOrderStatusId;
        }
```

```
public RestaurantDTO getRestaurantDTO() {
        return restaurantDTO;
}

public void setRestaurantDTO(RestaurantDTO restaurantDTO) {
        this.restaurantDTO = restaurantDTO;
}

public OrderDTO getOrderDTO() {
        return orderDTO;
}

public void setOrderDTO(OrderDTO orderDTO) {
        this.orderDTO = orderDTO;
}

public String getStatus() {
        return status;
}

public void setStatus(String status) {
        this.status = status;
}

public LocalDateTime getOrderedAt() {
        return orderedAt;
```

```
        }

        public void setOrderedAt(LocalDateTime orderedAt) {
                this.orderedAt = orderedAt;
        }

}
```

----------UPDATE ORDER STATUS REQUEST DTO

```
package com.feast.server_main.dto;

public class UpdateOrderStatusRequestDTO {

   private OrderDetails order;
   private String status;
   private RestaurantDetails restaurant;

   public static class OrderDetails {
      private Integer orderId;

      public OrderDetails() {
      }

      public Integer getOrderId() {
         return orderId;
      }
```

```
    public void setOrderId(Integer orderId) {
        this.orderId = orderId;
    }
}

public static class RestaurantDetails {
    private Integer restaurantId;

    public RestaurantDetails() {
    }

    public Integer getRestaurantId() {
        return restaurantId;
    }

    public void setRestaurantId(Integer restaurantId) {
        this.restaurantId = restaurantId;
    }
}

public UpdateOrderStatusRequestDTO() {
}

public OrderDetails getOrder() {
    return order;
}
```

```
    public void setOrder(OrderDetails order) {
        this.order = order;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public RestaurantDetails getRestaurant() {
        return restaurant;
    }

    public void setRestaurant(RestaurantDetails restaurant) {
        this.restaurant = restaurant;
    }
}
```

----------USER DTO

```
package com.feast.server_main.dto;

public class UserDTO {
```

```
    private Integer userId;
    private String userName;
    private String email;
    private Long phoneNumber;
    private String address;
    private String role;

    public UserDTO() {

    }
        public UserDTO(Integer userId, String userName, String email, Long phoneNumber,
String address,
                        String role) {
                super();
                this.userId = userId;
                this.userName = userName;
                this.email = email;
                this.phoneNumber = phoneNumber;
                this.address = address;
                this.role = role;
        }

        public Integer getUserId() {
                return userId;
        }
```

```
public String getUserName() {
        return userName;
}

public void setUserName(String userName) {
        this.userName = userName;
}

public String getEmail() {
        return email;
}

public void setEmail(String email) {
        this.email = email;
}


public Long getPhoneNumber() {
        return phoneNumber;
}

public void setPhoneNumber(Long phoneNumber) {
        this.phoneNumber = phoneNumber;
}

public String getAddress() {
```

```
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getRole() {
        return role;
    }

    public void setRole(String role) {
        this.role = role;
    }



}
```

----------MAIN MODEL----------

---------- FOODITEM

```
package com.feast.server_main.model;

import jakarta.persistence.*;

@Entity
@Table(name = "FoodItems")
```

```
public class FoodItem {

    @Id

    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"fooditems_generator")

    @SequenceGenerator(name = "fooditems_generator", sequenceName =
"fooditems_sequence", allocationSize = 1)

    @Column(name = "FoodId")

    private Integer foodId;


    @Column(name = "FoodName", columnDefinition = "VARCHAR(255)")

    private String foodName;


    @Column(name = "FoodType", columnDefinition = "VARCHAR(255)")

    private String foodType;


    @ManyToOne

    @JoinColumn(name = "RestaurantId")

    private Restaurant restaurant;


    @Column(name = "Description", columnDefinition = "VARCHAR(255)")

    private String description;


    @Column(name = "Price")

    private Float price;


    @Column(name = "ImageURL", columnDefinition = "VARCHAR(255)")

    private String imageURL;
```

```
    @Column(name = "Rating")
    private Double rating;

    public FoodItem() {
    }

    public FoodItem(Integer foodId, String foodName, String foodType, Restaurant
restaurant, String description, Float price, String imageURL, Double rating) {
        this.foodId = foodId;
        this.foodName = foodName;
        this.foodType= foodType;
        this.restaurant = restaurant;
        this.description = description;
        this.price = price;
        this.imageURL = imageURL;
        this.rating = rating;
    }

    public Integer getFoodId() {
        return foodId;
    }

    public void setFoodId(Integer foodId) {
        this.foodId = foodId;
    }
```

```
public String getFoodName() {
   return foodName;
}

public void setFoodName(String foodName) {
   this.foodName = foodName;
}

public String getFoodType() {
   return foodType;
}

public void setFoodType(String foodType) {
   this.foodType = foodType;
}

public Restaurant getRestaurant() {
   return restaurant;
}

public void setRestaurant(Restaurant restaurant) {
   this.restaurant = restaurant;
}

public String getDescription() {
```

```
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

public Float getPrice() {
    return price;
}

public void setPrice(Float price) {
    this.price = price;
}

public String getImageURL() {
    return imageURL;
}

public void setImageURL(String imageURL) {
    this.imageURL = imageURL;
}

public Double getRating() {
    return rating;
}
```

```
    public void setRating(Double rating) {
        this.rating = rating;
    }

    @Override
    public String toString() {
        return "FoodItem [foodId=" + foodId + ", foodName=" + foodName + ", restaurant=" + restaurant + ", description=" + description + ", price=" + price + ", imageURL=" + imageURL + ", rating=" + rating + "]";
    }
}
```

----------ORDER

```
package com.feast.server_main.model;

import jakarta.persistence.*;
import java.time.LocalDateTime;

@Entity
@Table(name = "Orders")
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "orders_generator")
    @SequenceGenerator(name = "orders_generator", sequenceName = "orders_sequence", allocationSize = 1)
    @Column(name = "OrderId")
```

```
    private Integer orderId;

    @ManyToOne
    @JoinColumn(name = "UserId", nullable = false)
    private User user;

    @ManyToOne
    @JoinColumn(name = "FoodId", nullable = false)
    private FoodItem foodItem;

    @Column(name = "TotalPrice")
    private Double totalPrice;

    @Column(name = "Quantity")
    private Integer quantity;

    @Column(name = "OrderDate")
    private LocalDateTime date;

    public Order() {
    }

    public Order(Integer orderId, User user, FoodItem foodItem, Double totalPrice, Integer
quantity, LocalDateTime date) {
        this.orderId = orderId;
        this.user = user;
```

```
    this.foodItem = foodItem;
    this.totalPrice = totalPrice;
    this.quantity = quantity;
    this.date = date;
}

public Integer getOrderId() {
    return orderId;
}

public void setOrderId(Integer orderId) {
    this.orderId = orderId;
}

public User getUser() {
    return user;
}

public void setUser(User user) {
    this.user = user;
}

public FoodItem getFoodItem() {
    return foodItem;
}
```

```
public void setFoodItem(FoodItem foodItem) {
   this.foodItem = foodItem;
}

public Double getTotalPrice() {
   return totalPrice;
}

public void setTotalPrice(Double totalPrice) {
   this.totalPrice = totalPrice;
}

public Integer getQuantity() {
   return quantity;
}

public void setQuantity(Integer quantity) {
   this.quantity = quantity;
}

public LocalDateTime getDate() {
   return date;
}

public void setDate(LocalDateTime date) {
   this.date = date;
```

```
    }


    @Override
    public String toString() {
        return "Order [orderId=" + orderId + ", user=" + user + ", foodItem=" + foodItem + ",
totalPrice=" + totalPrice + ", quantity=" + quantity + ", date=" + date + "]";
    }
}
----------RESTAURANT
package com.feast.server_main.model;


import jakarta.persistence.*;


@Entity
@Table(name = "restaurants")
public class Restaurant {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"restaurants_generator")
    @SequenceGenerator(name = "restaurants_generator", sequenceName =
"restaurants_sequence", allocationSize = 1)
    @Column(name = "RestaurantId")
    private Integer restaurantId;



    @OneToOne
    @JoinColumn(name = "UserId", unique = true)
```

```
    private User user;

    @Column(name = "RestaurantName")
    private String restaurantName;

    @Column(name = "Address")
    private String address;

    @Column(name = "Cuisine")
    private String cuisine;

    @Column(name = "OwnerName")
    private String ownerName;

    public Restaurant() {
    }

    public Restaurant(User user,Integer restaurantId, String restaurantName, String address,
String cuisine, String ownerName) {
            this.user = user;
        this.restaurantId = restaurantId;
        this.restaurantName = restaurantName;
        this.address = address;
        this.cuisine = cuisine;
        this.ownerName = ownerName;
    }
```

```
public Integer getRestaurantId() {
   return restaurantId;
}

public void setRestaurantId(Integer restaurantId) {
   this.restaurantId = restaurantId;
}

public User getUser() {
   return user;
}

public void setUser(User user) {
   this.user = user;
}

public String getRestaurantName() {
   return restaurantName;
}

public void setRestaurantName(String restaurantName) {
   this.restaurantName = restaurantName;
}

public String getAddress() {
```

```
    return address;
}

public void setAddress(String address) {
    this.address = address;
}

public String getCuisine() {
    return cuisine;
}

public void setCuisine(String cuisine) {
    this.cuisine = cuisine;
}

public String getOwnerName() {
    return ownerName;
}

public void setOwnerName(String ownerName) {
    this.ownerName = ownerName;
}

@Override
public String toString() {
```

```
      return "Restaurant [restaurantId=" + restaurantId + ", restaurantName=" +
restaurantName + ", address=" + address + ", cuisine=" + cuisine + ", ownerName=" +
ownerName + "]";
   }
}
----------RESTAURANT STATUS
package com.feast.server_main.model;

import jakarta.persistence.*;
import java.time.LocalDateTime;

@Entity
@Table(name = "RestaurantOrderStatus")
public class RestaurantOrderStatus {
   @Id
   @GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"order_status_generator")
   @SequenceGenerator(name = "order_status_generator", sequenceName =
"order_status_sequence", allocationSize = 1)
   @Column(name = "RestaurantOrderStatusId")
   private Integer restaurantOrderStatusId;

   @ManyToOne
   @JoinColumn(name = "RestaurantId", nullable = false)
   private Restaurant restaurant;

   @ManyToOne
```

```
    @JoinColumn(name = "OrderId", nullable = false)
    private Order order;

    @Column(name = "Status", columnDefinition = "VARCHAR(255)")
    private String status;

    @Column(name = "OrderedAt")
    private LocalDateTime orderedAt;

    public RestaurantOrderStatus() {
    }

    public RestaurantOrderStatus(Integer restaurantOrderStatusId, Restaurant restaurant,
Order order, String status, LocalDateTime orderedAt) {
        this.restaurantOrderStatusId = restaurantOrderStatusId;
        this.restaurant = restaurant;
        this.order = order;
        this.status = status;
        this.orderedAt = orderedAt;
    }

    public Integer getRestaurantOrderStatusId() {
        return restaurantOrderStatusId;
    }

    public void setRestaurantOrderStatusId(Integer restaurantOrderStatusId) {
```

```
    this.restaurantOrderStatusId = restaurantOrderStatusId;
}

public Restaurant getRestaurant() {
    return restaurant;
}

public void setRestaurant(Restaurant restaurant) {
    this.restaurant = restaurant;
}

public Order getOrder() {
    return order;
}

public void setOrder(Order order) {
    this.order = order;
}

public String getStatus() {
    return status;
}

public void setStatus(String status) {
    this.status = status;
}
```

```
    public LocalDateTime getOrderedAt() {
        return orderedAt;
    }

    public void setOrderedAt(LocalDateTime orderedAt) {
        this.orderedAt = orderedAt;
    }

    @Override
    public String toString() {
        return "RestaurantOrderStatus [restaurantOrderStatusId=" + restaurantOrderStatusId +
", restaurant=" + restaurant + ", order=" + order + ", status=" + status + ", orderedAt=" +
orderedAt + "]";
    }
}
```

----------USER

```
package com.feast.server_main.model;

import jakarta.persistence.*;

@Entity
@Table(name = "Users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "user_generator")
```

```
    @SequenceGenerator(name = "user_generator", sequenceName = "users_sequence",
allocationSize = 1)
    @Column(name = "UserId")
    private Integer userId;

    @Column(name = "UserName", columnDefinition = "VARCHAR(255)")
    private String userName;

    @Column(name = "email", columnDefinition = "VARCHAR(255)")
    private String email;

    @Column(name = "phoneNumber")
    private Long phoneNumber;

    @Column(name = "password", columnDefinition = "VARCHAR(255)")
    private String password;

    @Column(name = "address", columnDefinition = "VARCHAR(255)")
    private String address;

    @Column(name = "role", columnDefinition = "VARCHAR(255)")
    private String role;

    @OneToOne(mappedBy = "user")
    private Restaurant restaurant;
```

```
    public User() {
    }

    public User(Integer userId, String userName, String email, Long phoneNumber, String
password, String address, String role) {
        this.userId = userId;
        this.userName = userName;
        this.email = email;
        this.phoneNumber = phoneNumber;
        this.password = password;
        this.address = address;
        this.role = role;
    }

    public Integer getUserId() {
        return userId;
    }

    public void setUserId(Integer userId) {
        this.userId = userId;
    }

    public String getUserName() {
        return userName;
    }
```

```
public void setUserName(String userName) {
   this.userName = userName;
}

public String getEmail() {
   return email;
}

public void setEmail(String email) {
   this.email = email;
}

public Long getPhoneNumber() {
   return phoneNumber;
}

public void setPhoneNumber(Long phoneNumber) {
   this.phoneNumber = phoneNumber;
}

public String getPassword() {
   return password;
}

public void setPassword(String password) {
   this.password = password;
```

```
}

public String getAddress() {
   return address;
}

public void setAddress(String address) {
   this.address = address;
}

public String getRole() {
   return role;
}

public void setRole(String role) {
   this.role = role;
}

public Restaurant getRestaurant() {
   return restaurant;
}

public void setRestaurant(Restaurant restaurant) {
   this.restaurant = restaurant;
}
```

```
  @Override
  public String toString() {
    return "User [userId=" + userId + ", userName=" + userName + ", email=" + email + ",
phoneNumber=" + phoneNumber + ", password=" + password + ", address=" + address + ",
role=" + role + "]";
  }
}
```

----------MAIN SERVICE----------

------AUTHSERVICE

```
package com.feast.server_main.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.webjars.NotFoundException; // Consider using Spring's DataAccessException
import org.springframework.dao.EmptyResultDataAccessException; // Import this
import org.springframework.transaction.annotation.Transactional; // Import this
import org.slf4j.Logger; // Import SLF4J
import org.slf4j.LoggerFactory;

import com.feast.server_main.dto.UserDTO;
import com.feast.server_main.model.User;
import com.feast.server_main.repository.UserRepository;

@Service
public class AuthService {
  private static final Logger logger = LoggerFactory.getLogger(AuthService.class);
```

```
  @Autowired
  private UserRepository userRepository;

  @Autowired
  private BCryptPasswordEncoder bCryptPasswordEncoder;

  public AuthService(UserRepository userRepository, BCryptPasswordEncoder
bCryptPasswordEncoder) {
    this.userRepository = userRepository;
    this.bCryptPasswordEncoder = bCryptPasswordEncoder;
  }

  @Transactional
  public User signup(User user) {
    String email = user.getEmail();
    User existingUser = userRepository.getByEmail(email);

    if (existingUser != null) {
      throw new IllegalArgumentException("Email already exists");
    }

    User newUser = new User();
    newUser.setUserName(user.getUserName());
    newUser.setEmail(user.getEmail());
    newUser.setPhoneNumber(user.getPhoneNumber());
```

```
    newUser.setAddress(user.getAddress());
    newUser.setRole(user.getRole());
    newUser.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));

    User savedUser = userRepository.save(newUser);
    logger.info("User signed up successfully: {}", savedUser.getEmail());
    return savedUser;
}

public UserDTO login(User currUser) {
    String email = currUser.getEmail();
    logger.info("Received login request for email: {}", email);
    User user = null;
    try {
        user = userRepository.getByEmail(email);
    } catch (EmptyResultDataAccessException e) {
        logger.error("User not found for email: {}", email);
        throw new NotFoundException("User not found");
    }
    logger.debug("Encoded password from DB: {}", user.getPassword());
    logger.debug("Raw password from request: {}", currUser.getPassword());
    if (bCryptPasswordEncoder.matches(currUser.getPassword(), user.getPassword())) {
        logger.info("Login successful for user: {}", email);
        return convertToDto(user);
    } else {
        logger.warn("Invalid password for user: {}", email);
```

```
            throw new NotFoundException("Invalid password");
        }
    }

    private UserDTO convertToDto(User user) {
        return new UserDTO(
                user.getUserId(),
                user.getUserName(),
                user.getEmail(),
                user.getPhoneNumber(),
                user.getAddress(),
                user.getRole());
    }
}
```

----------USER DETAILS SERVICE

```
// CustomUserDetailsService.java
package com.feast.server_main.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.springframework.security.core.authority.SimpleGrantedAuthority;

import com.feast.server_main.model.User;
```

```
import com.feast.server_main.repository.UserRepository;

import java.util.Collections;

@Service
public class CustomUserDetailsService implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String email) throws
UsernameNotFoundException {
        User user = userRepository.getByEmail(email);
        if (user == null) {
            throw new UsernameNotFoundException("User not found with email: " + email);
        }
        return new org.springframework.security.core.userdetails.User(
                user.getEmail(),
                user.getPassword(),
                Collections.singletonList(new SimpleGrantedAuthority(user.getRole()))
        );
    }
}
```

----------RESTAURANT SERVICE

```
package com.feast.server_main.service;
```

```
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.feast.server_main.dto.FoodItemDTO;
import com.feast.server_main.dto.OrderDTO;
import com.feast.server_main.dto.ResFoodItemDTO;
import com.feast.server_main.dto.RestaurantDTO;
import com.feast.server_main.dto.RestaurantOrderStatusDTO;
import com.feast.server_main.dto.UserDTO;
import com.feast.server_main.model.FoodItem;
import com.feast.server_main.model.Order;
import com.feast.server_main.model.Restaurant;
import com.feast.server_main.model.RestaurantOrderStatus;
import com.feast.server_main.model.User;
import com.feast.server_main.repository.FoodItemRepository;
import com.feast.server_main.repository.OrderRepository;
import com.feast.server_main.repository.RestaurantOrderStatusRepository;
import com.feast.server_main.repository.RestaurantRepository;
import com.feast.server_main.repository.UserRepository;
```

```
@Service
public class RestaurantService {

    @Autowired
    private FoodItemRepository foodItemRepository;

    @Autowired
    private RestaurantRepository restaurantRepository;

    @Autowired
    private RestaurantOrderStatusRepository restaurantOrderStatusRepository;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private OrderRepository orderRepository;


    @Transactional
    public UserDTO createRestaurantForExistingUser(Integer userId, String
restaurantName, String restaurantAddress,
                    String cuisine, String ownerName) {

        User user = userRepository.findByUserId(userId);
```

```
            if (user.getRestaurant() != null) {
                throw new IllegalStateException("User with ID " + userId + " already has a restaurant.");
            }

            Restaurant restaurant = new Restaurant();
            restaurant.setUser(user);
            restaurant.setRestaurantName(restaurantName);
            restaurant.setAddress(restaurantAddress);
            restaurant.setCuisine(cuisine);
            restaurant.setOwnerName(ownerName);
            restaurantRepository.save(restaurant);

            user.setRestaurant(restaurant);
            userRepository.save(user);

            return convertToUserDTO(user);
        }

        private UserDTO convertToUserDTO(User user) {
            UserDTO userDTO = new UserDTO();
            userDTO.setUserName(user.getUserName());
            userDTO.setEmail(user.getEmail());
            userDTO.setRole(user.getRole());
            userDTO.setPhoneNumber(user.getPhoneNumber());
```

```
            userDTO.setAddress(user.getAddress());

            if (user.getRestaurant() != null) {
                Restaurant restaurant = new Restaurant();
                restaurant.setRestaurantId(user.getRestaurant().getRestaurantId());
        restaurant.setRestaurantName(user.getRestaurant().getRestaurantName());
                restaurant.setAddress(user.getRestaurant().getAddress());
                restaurant.setCuisine(user.getRestaurant().getCuisine());
                restaurant.setOwnerName(user.getRestaurant().getOwnerName());
            }
            return userDTO;
        }

        public boolean hasRestaurant(Integer userId) {
            return restaurantRepository.existsByUserId(userId);
        }

        public Optional<Integer> getRestaurantIdByUserId(Integer userId) {
            Optional<Restaurant> restaurant =
restaurantRepository.findByUser_UserId(userId);
            return restaurant.map(Restaurant::getRestaurantId);
        }

        public List<FoodItemDTO> getAllItems() {
            List<FoodItem> items = foodItemRepository.findAll();
            return items.stream()
```

```
            .map(this::mapFoodItemToDTO)
            .collect(Collectors.toList());
    }

    public List<ResFoodItemDTO> getFoodItemsByRestaurant(Integer restaurantId) {
        List<FoodItem> foodItems =
foodItemRepository.findByRestaurant_RestaurantId(restaurantId);
        return convertToResDto(foodItems);
    }

    @Transactional
    public FoodItemDTO addFoodItem(FoodItem foodItem) {
        FoodItem savedFoodItem = foodItemRepository.save(foodItem);
        return mapFoodItemToDTO(savedFoodItem);
    }

    @Transactional
    public FoodItemDTO updateFoodItem(Integer foodItemId, FoodItem foodItem) {
        FoodItem existingFoodItem = foodItemRepository.findById(foodItemId)
                .orElseThrow(() -> new IllegalArgumentException("Food Item
not found with ID: " + foodItemId));

        existingFoodItem.setFoodName(foodItem.getFoodName());
        existingFoodItem.setFoodType(foodItem.getFoodType());
        existingFoodItem.setDescription(foodItem.getDescription());
        existingFoodItem.setPrice(foodItem.getPrice());
        existingFoodItem.setImageURL(foodItem.getImageURL());
```

```
            existingFoodItem.setRating(foodItem.getRating());


            return convertToDto(foodItemRepository.save(existingFoodItem));
        }


        public FoodItemDTO convertToDto(FoodItem foodItem) {
            return new FoodItemDTO(foodItem.getFoodId(), foodItem.getFoodName(),
foodItem.getFoodType(),
                    foodItem.getDescription(), foodItem.getPrice(),
foodItem.getImageURL(), foodItem.getRating(),
                    mapRestaurantToDTO(foodItem.getRestaurant()));
        }


        public ResFoodItemDTO mapToResFoodDto(FoodItem foodItem) {
            return new ResFoodItemDTO(foodItem.getFoodId(),
foodItem.getFoodName(), foodItem.getFoodType(),
                    foodItem.getDescription(), foodItem.getPrice(),
foodItem.getImageURL(), foodItem.getRating()
                    );
        }
        public ResFoodItemDTO mapToResFoodDto(FoodItemDTO foodItem) {
            return new ResFoodItemDTO(foodItem.getFoodId(),
foodItem.getFoodName(), foodItem.getFoodType(),
                    foodItem.getDescription(), foodItem.getPrice(),
foodItem.getImageURL(), foodItem.getRating()
                    );
        }
```

```
    private RestaurantDTO mapRestaurantToDTO(Restaurant restaurant) {
        return new RestaurantDTO(restaurant.getRestaurantId(),
restaurant.getRestaurantName(), restaurant.getAddress(),
                restaurant.getCuisine(), restaurant.getOwnerName());
    }


    public List<ResFoodItemDTO> convertToResDto(List<FoodItem> foodItems) {
        List<ResFoodItemDTO> resFoodItems = new ArrayList<>();
        if (foodItems != null && !foodItems.isEmpty()) {
            for (FoodItem foodItem : foodItems) {
                ResFoodItemDTO resFoodItem = new ResFoodItemDTO(
                foodItem.getFoodId(),
                foodItem.getFoodName(),
                foodItem.getFoodType(),
                foodItem.getDescription(),
                foodItem.getPrice(),
                foodItem.getImageURL(),
                foodItem.getRating()
                        );
                resFoodItems.add(resFoodItem);
            }
        }
        return resFoodItems;
    }
```

```
	@Transactional
	public void deleteFoodItem(Integer foodItemId) {
		FoodItem foodItem = foodItemRepository.findById(foodItemId)
				.orElseThrow(() -> new IllegalArgumentException("Food Item not found"));
		foodItemRepository.delete(foodItem);
	}

	FoodItem convertFoodDtoToEntity(FoodItemDTO dto, Restaurant restaurant) {
		FoodItem foodItem = new FoodItem(dto.getFoodId(), dto.getFoodName(), dto.getFoodType(), restaurant,
				dto.getDescription(), dto.getPrice(), dto.getImageURL(), dto.getRating());
		return foodItem;
	}

	public RestaurantOrderStatusDTO getOrderStatusByOrderId(Integer orderId) {
		Optional<RestaurantOrderStatus> orderStatus = restaurantOrderStatusRepository.findByOrderOrderId(orderId);
		return mapResOrderToDTO(orderStatus.orElse(null));
	}

	public List<RestaurantOrderStatusDTO> getAllOrderStatusesByRestaurantId(Integer restaurantId) {
		List<RestaurantOrderStatus> restaurantOrderList = restaurantOrderStatusRepository.findByRestaurantRestaurantId(restaurantId);
		return restaurantOrderList.stream()
	 .map(this::mapResOrderToDTO)
```

```
            .collect(Collectors.toList());

        }


        @Transactional
    public RestaurantOrderStatusDTO updateOrderStatus(Integer orderId, String newStatus,
Integer restaurantId) {
        Optional<RestaurantOrderStatus> existingOrderStatusOptional =
restaurantOrderStatusRepository
            .findByOrderOrderId(orderId);
        Restaurant restaurant = restaurantRepository.findById(restaurantId)
            .orElseThrow(() -> new IllegalArgumentException("Restaurant not found with ID: " +
restaurantId));


        Order order = orderRepository.findByOrderId(orderId);


        RestaurantOrderStatus orderStatusToSave;


        if (existingOrderStatusOptional.isPresent()) {
            RestaurantOrderStatus existingOrderStatus = existingOrderStatusOptional.get();
            existingOrderStatus.setRestaurant(restaurant);
            existingOrderStatus.setOrder(order);
            existingOrderStatus.setStatus(newStatus);
            orderStatusToSave = restaurantOrderStatusRepository.save(existingOrderStatus);
        } else {
            RestaurantOrderStatus newOrderStatus = new RestaurantOrderStatus();
            newOrderStatus.setRestaurant(restaurant);
            newOrderStatus.setOrder(order);
```

```
        newOrderStatus.setStatus(newStatus);
        orderStatusToSave = restaurantOrderStatusRepository.save(newOrderStatus);
    }

    return mapResOrderToDTO(orderStatusToSave);
  }

    public UserDTO convertUserToDto(User user) {
        return new UserDTO(user.getUserId(), user.getUserName(), user.getEmail(),
user.getPhoneNumber(),
                user.getAddress(), user.getRole());
    }

    public FoodItemDTO mapFoodItemToDTO(FoodItem foodItem) {
        return new FoodItemDTO(foodItem.getFoodId(), foodItem.getFoodName(),
foodItem.getFoodType(),
                foodItem.getDescription(), foodItem.getPrice(),
foodItem.getImageURL(), foodItem.getRating(),
                mapRestaurantToDTO(foodItem.getRestaurant()));
    }

    private OrderDTO mapOrderToDTO(Order order,RestaurantOrderStatus entity) {
        return new OrderDTO(order.getOrderId(),
convertUserToDto(entity.getOrder().getUser()),
mapFoodItemToDTO(entity.getOrder().getFoodItem()), order.getTotalPrice(),
                order.getQuantity(), order.getDate());
    }
```

```
	private RestaurantOrderStatusDTO mapResOrderToDTO(RestaurantOrderStatus entity) {
		RestaurantOrderStatusDTO orderStatusDTO = new RestaurantOrderStatusDTO();

	orderStatusDTO.setRestaurantOrderStatusId(entity.getRestaurantOrderStatusId());

	orderStatusDTO.setRestaurantDTO(mapRestaurantToDTO(entity.getRestaurant()));
		orderStatusDTO.setOrderDTO(mapOrderToDTO(entity.getOrder(),entity));
		orderStatusDTO.setStatus(entity.getStatus());
		orderStatusDTO.setOrderedAt(entity.getOrderedAt());
		return orderStatusDTO;
	}



	public List<RestaurantOrderStatusDTO> getAllOrderStatus() {
		List<RestaurantOrderStatus> ordersList = restaurantOrderStatusRepository.findAll();
		return ordersList.stream()
	 .map(this::mapResOrderToDTO)
	 .collect(Collectors.toList());
	}

	FoodItem convertDtoToEntity(FoodItemDTO dto, Restaurant restaurant) {
		FoodItem foodItem = new FoodItem();
```

```
            foodItem.setFoodName(dto.getFoodName());
            foodItem.setFoodType(dto.getFoodType());
            foodItem.setRestaurant(restaurant);
            foodItem.setDescription(dto.getDescription());
            foodItem.setPrice(dto.getPrice());
            foodItem.setImageURL(dto.getImageURL());
            foodItem.setRating(dto.getRating());
            return foodItem;
        }


        private FoodItemDTO mapFoodItemToDto(FoodItem foodItem) {
            return new FoodItemDTO(foodItem.getFoodId(), foodItem.getFoodName(),
foodItem.getFoodType(),
                    foodItem.getDescription(), foodItem.getPrice(),
foodItem.getImageURL(), foodItem.getRating(),
                    mapRestaurantToDTO(foodItem.getRestaurant()));
        }

}
```

----------USER SERVICE

```
package com.feast.server_main.service;

import java.time.LocalDateTime;
import java.util.Collections;
import java.util.List;
import java.util.Optional;
```

```
import java.util.stream.Collectors;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.feast.server_main.dto.CusOrderDTO;
import com.feast.server_main.dto.FoodItemDTO;
import com.feast.server_main.dto.OrderDTO;
import com.feast.server_main.dto.ResFoodItemDTO;
import com.feast.server_main.dto.RestaurantDTO;
import com.feast.server_main.dto.RestaurantOrderStatusDTO;
import com.feast.server_main.dto.UserDTO;
import com.feast.server_main.model.FoodItem;
import com.feast.server_main.model.Order;
import com.feast.server_main.model.Restaurant;
import com.feast.server_main.model.RestaurantOrderStatus;
import com.feast.server_main.model.User;
import com.feast.server_main.repository.FoodItemRepository;
import com.feast.server_main.repository.OrderRepository;
import com.feast.server_main.repository.RestaurantOrderStatusRepository;
import com.feast.server_main.repository.RestaurantRepository;
import com.feast.server_main.repository.UserRepository;

@Service
public class UserService {
```

```
    @Autowired
    private RestaurantRepository restaurantRepository;

    @Autowired
    private FoodItemRepository foodItemRepository;

    @Autowired
    private OrderRepository orderRepository;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private RestaurantOrderStatusRepository restaurantOrderStatusRepository;

    @Autowired
    private RestaurantService restaurantService;

    @Transactional(readOnly = true)
    public List<RestaurantDTO> getAllRestaurants() {
        List<Restaurant> restaurants = restaurantRepository.findAll();
        return
restaurants.stream().map(this::mapRestaurantToDTO).collect(Collectors.toList());
    }
```

```
        public Integer getRestaurantIdByFoodItemId(Integer foodItemId) {


    Optional<FoodItem> foodItem = foodItemRepository.findById(foodItemId);
    if (foodItem.isPresent()) {
      return foodItem.get().getRestaurant().getRestaurantId();
    } else {
      return null;
    }
  }
        public List<ResFoodItemDTO> getFoodItemsByRestaurant(Integer restaurantId) {
                List<FoodItem> foodItems =
foodItemRepository.findByRestaurant_RestaurantId(restaurantId);
                return restaurantService.convertToResDto(foodItems);
        }


        @Transactional
        public CusOrderDTO placeOrder(OrderDTO orderDTO) {
                if (orderDTO.getFoodItem() == null || orderDTO.getQuantity() == null ||
orderDTO.getTotalPrice() == null) {
                        throw new IllegalArgumentException("Order details are incomplete.");
                }


                User user;
                if (orderDTO.getUser() != null && orderDTO.getUser().getUserId() != null) {
                        user =
userRepository.findById(orderDTO.getUser().getUserId()).orElseThrow(
```

```
                                        () -> new IllegalArgumentException("User not found for
userId: " + orderDTO.getUser().getUserId()));

                } else if (orderDTO.getUser() != null && orderDTO.getUser().getUserName() !=
null

                                && !orderDTO.getUser().getUserName().isEmpty()) {

                        List<User> users =
userRepository.findByUserName(orderDTO.getUser().getUserName());

                        if (users.isEmpty()) {

                                throw new IllegalArgumentException("User not found for
userName: " + orderDTO.getUser().getUserName());

                        }

                        user = users.get(0);

                } else {

                        throw new IllegalArgumentException("User information (userId or
userName) is required.");

                }

                FoodItem foodItem =
foodItemRepository.findById(orderDTO.getFoodItem().getFoodId())

                                .orElseThrow(() -> new IllegalArgumentException("Food item
not found."));


                Order order = new Order();

                order.setUser(user);

                order.setFoodItem(foodItem);

                order.setTotalPrice(orderDTO.getTotalPrice());

                order.setQuantity(orderDTO.getQuantity());

                order.setDate(LocalDateTime.now());
```

```
                Order savedOrder = orderRepository.save(order);


                return mapOrderToCusDTO(savedOrder);
        }


        public List<OrderDTO> getCartByUserId(Integer userId) {
                User user = userRepository.findByUserId(userId);
                List<Order> cart =
orderRepository.findByUserWithFoodItemAndRestaurant(user);
                return cart.stream().map(this::mapOrderToDTO).collect(Collectors.toList());
        }


        public List<ResFoodItemDTO> getAllFoodItems() {
                List<FoodItem> allFoodItems = foodItemRepository.findAll();


                 return allFoodItems.stream().map(this::mapFoodItemToResDTO)
                 .collect(Collectors.toList());


//                return
allFoodItems.stream().map(this::mapFoodItemToDTO).collect(Collectors.toList());
        }


   @Transactional
   public OrderDTO updateOrderQuantity(Integer orderId, int newQuantity, Integer userId) {
      Order order = orderRepository.findByOrderId(orderId);
      if (order == null) {
         throw new IllegalArgumentException("Order not found");
```

```
    }
    if (!order.getUser().getUserId().equals(userId)) {
       throw new IllegalArgumentException("Order does not belong to this user");
    }
    order.setQuantity(newQuantity);
    order.setTotalPrice(order.getFoodItem().getPrice() * (double) newQuantity);
    Order updatedOrder = orderRepository.save(order);

    return mapOrderToDTO(updatedOrder);
  }

        @Transactional
        public void removeOrder(Integer userId) {
                Optional<User> userOptional = userRepository.findById(userId);
                if (!userOptional.isPresent()) {
                        throw new IllegalArgumentException("User not found.");
                }
                User user = userOptional.get();
                List<Order> orders = orderRepository.findByUser(user);
                if (orders.isEmpty()) {
                        throw new IllegalArgumentException("Cart is already empty for this
user.");
                }
                orderRepository.deleteAll(orders);
        }
```

```
@Transactional(readOnly = true)
public List<UserDTO> getUserProfileByUserId(Integer userId) {
	User user = userRepository.findByUserId(userId);
	if (user != null) {
		return Collections.singletonList(mapUserToDTO(user));
	} else {
		return Collections.emptyList();
	}
}

@Transactional
public UserDTO updateUserProfile(Integer userId, UserDTO userDTO) {
	User user = userRepository.findByUserId(userId);
	User existingUser = user;

	if (userDTO.getUserName() != null) {
		existingUser.setUserName(userDTO.getUserName());
	}
	if (userDTO.getPhoneNumber() != null) {
		existingUser.setPhoneNumber(userDTO.getPhoneNumber());
	}
	if (userDTO.getEmail() != null) {
		existingUser.setEmail(userDTO.getEmail());
	}
	if (userDTO.getAddress() != null) {
		existingUser.setAddress(userDTO.getAddress());
```

```
            }

            User updatedUser = userRepository.save(existingUser);
            UserDTO updatedUserDTO = convertToDto(updatedUser);
            return updatedUserDTO;
        }

        @Transactional
        public RestaurantOrderStatusDTO updateOrderStatus(RestaurantOrderStatus
orderStatusDTO) {
            Order orderDTOOrder =
orderRepository.findByOrderId(orderStatusDTO.getOrder().getOrderId());
            Restaurant restaurant = restaurantRepository

        .findByRestaurantId(orderStatusDTO.getRestaurant().getRestaurantId());
            String status = orderStatusDTO.getStatus();
            if (restaurant == null || status == null || orderDTOOrder == null) {
                throw new IllegalArgumentException("Missing required order
details.");
            }

            RestaurantOrderStatus restaurantOrderStatus = new
RestaurantOrderStatus();
            restaurantOrderStatus.setRestaurant(restaurant);
            restaurantOrderStatus.setOrder(orderDTOOrder);
            restaurantOrderStatus.setStatus(status);
            restaurantOrderStatus.setOrderedAt(LocalDateTime.now());
```

```
        return
mapResOrderToDTO(restaurantOrderStatusRepository.save(restaurantOrderStatus));
    }




    @Transactional
    public void removeOrderItem(Integer orderId, Integer userId) {
        Order order = orderRepository.findByOrderId(orderId);
        if (!order.getUser().getUserId().equals(userId)) {
            throw new IllegalArgumentException("Unauthorized to remove this
order.");
        }
        orderRepository.delete(order);
    }


    private ResFoodItemDTO mapFoodItemToResDTO(FoodItem foodItem) {
        ResFoodItemDTO dto = new ResFoodItemDTO(foodItem.getFoodId(),
foodItem.getFoodName(), foodItem.getFoodType(),
                foodItem.getDescription(), foodItem.getPrice(),
foodItem.getImageURL(), foodItem.getRating());
        return dto;
    }


    private RestaurantDTO mapRestaurantToDTO(Restaurant restaurant) {
```

```
        return new RestaurantDTO(restaurant.getRestaurantId(),
restaurant.getRestaurantName(), restaurant.getAddress(),
                restaurant.getCuisine(), restaurant.getOwnerName());
    }

    private FoodItemDTO mapFoodItemToDTO(FoodItem foodItem) {
        return new FoodItemDTO(foodItem.getFoodId(), foodItem.getFoodName(),
foodItem.getFoodType(),
                foodItem.getDescription(), foodItem.getPrice(),
foodItem.getImageURL(), foodItem.getRating(),
                mapRestaurantToDTO(foodItem.getRestaurant()));
    }

    private OrderDTO mapOrderToDTO(Order order) {
        return new OrderDTO(order.getOrderId(), mapUserToDTO(order.getUser()),
mapFoodItemToDTO(order.getFoodItem()), order.getTotalPrice(),
                order.getQuantity(), order.getDate());
    }

    private CusOrderDTO mapOrderToCusDTO(Order order) {
        return new CusOrderDTO(order.getOrderId(),
mapFoodItemToResDTO(order.getFoodItem()), order.getTotalPrice(),
                order.getQuantity(), order.getDate());
    }
    private RestaurantOrderStatusDTO mapResOrderToDTO(RestaurantOrderStatus
entity) {
```

```
            return new RestaurantOrderStatusDTO(
                    entity.getRestaurantOrderStatusId(),
                    mapRestaurantToDTO(entity.getRestaurant()),
                    mapOrderToDTO(entity.getOrder()),
                    entity.getStatus(),
                    entity.getOrderedAt()
                    );
        }


        private UserDTO mapUserToDTO(User user) {
            return new UserDTO(user.getUserId(), user.getUserName(), user.getEmail(),
user.getPhoneNumber(),
                    user.getAddress(), user.getRole());
        }


        private UserDTO convertToDto(User user) {
            return new UserDTO(user.getUserId(), user.getUserName(), user.getEmail(),
user.getPhoneNumber(),
                    user.getAddress(), user.getRole());
        }
}
```