**Entity:**

**Category:**

package com.Project.Inventory.and.Sales.Management.System.Entity;  
  
import jakarta.validation.constraints.NotNull;  
  
public class Category {  
  
 private Integer id;  
  
 @NotNull  
 private String name;  
  
 private String description;  
  
  
 public Category() {}  
  
 public Category(Integer id, String name, String description) {  
 this.id = id;  
 this.name = name;  
 this.description = description;  
 }  
  
  
 public Integer getId() {  
 return id;  
 }  
  
 public void setId(Integer id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getDescription() {  
 return description;  
 }  
  
 public void setDescription(String description) {  
 this.description = description;  
 }  
}  
  
  
**Discount:**

package com.Project.Inventory.and.Sales.Management.System.Entity;  
  
import jakarta.validation.constraints.NotNull;  
  
import java.time.LocalDate;  
  
public class Discount {  
  
 private Integer id;  
  
 @NotNull  
 private Double percentage;  
  
 @NotNull  
 private LocalDate startDate;  
  
 @NotNull  
 private LocalDate endDate;  
  
 private String description;  
  
 public Discount() {}  
  
 public Discount(Integer id, Double percentage, LocalDate startDate, LocalDate endDate, String description) {  
 this.id = id;  
 this.percentage = percentage;  
 this.startDate = startDate;  
 this.endDate = endDate;  
 this.description = description;  
 }  
  
 public Integer getId() {  
 return id;  
 }  
  
 public void setId(Integer id) {  
 this.id = id;  
 }  
  
 public Double getPercentage() {  
 return percentage;  
 }  
  
 public void setPercentage(Double percentage) {  
 this.percentage = percentage;  
 }  
  
 public LocalDate getStartDate() {  
 return startDate;  
 }  
  
 public void setStartDate(LocalDate startDate) {  
 this.startDate = startDate;  
 }  
  
 public LocalDate getEndDate() {  
 return endDate;  
 }  
  
 public void setEndDate(LocalDate endDate) {  
 this.endDate = endDate;  
 }  
  
 public String getDescription() {  
 return description;  
 }  
  
 public void setDescription(String description) {  
 this.description = description;  
 }  
}

**InvoiceEmailLog:**

package com.Project.Inventory.and.Sales.Management.System.Entity;  
  
import java.time.LocalDateTime;  
  
public class InvoiceEmailLog {  
 private Integer id;  
 private Integer saleId;  
 private String emailTo;  
 private LocalDateTime sentAt;  
  
  
 public InvoiceEmailLog() {  
 }  
  
 public InvoiceEmailLog(Integer id, Integer saleId, String emailTo, LocalDateTime sentAt) {  
 this.id = id;  
 this.saleId = saleId;  
 this.emailTo = emailTo;  
 this.sentAt = sentAt;  
 }  
  
 public Integer getId() {  
 return id;  
 }  
  
 public void setId(Integer id) {  
 this.id = id;  
 }  
  
 public Integer getSaleId() {  
 return saleId;  
 }  
  
 public void setSaleId(Integer saleId) {  
 this.saleId = saleId;  
 }  
  
 public String getEmailTo() {  
 return emailTo;  
 }  
  
 public void setEmailTo(String emailTo) {  
 this.emailTo = emailTo;  
 }  
  
 public LocalDateTime getSentAt() {  
 return sentAt;  
 }  
  
 public void setSentAt(LocalDateTime sentAt) {  
 this.sentAt = sentAt;  
 }  
}

**Product:**

package com.Project.Inventory.and.Sales.Management.System.Entity;  
  
import jakarta.validation.constraints.NotNull;  
  
import java.time.LocalDate;  
  
public class Product {  
  
 private Integer id;  
  
 @NotNull  
 private String name;  
  
 @NotNull  
 private Integer categoryId;  
  
 @NotNull  
 private Double price;  
  
 @NotNull  
 private Integer quantity;  
  
 private LocalDate expiryDate;  
  
 private Boolean isAvailable;  
  
 public Product() {}  
  
  
 public Product(Integer id, String name, Integer categoryId, Double price, Integer quantity, LocalDate expiryDate, Boolean isAvailable) {  
 this.id = id;  
 this.name = name;  
 this.categoryId = categoryId;  
 this.price = price;  
 this.quantity = quantity;  
 this.expiryDate = expiryDate;  
 this.isAvailable = isAvailable;  
 }  
  
 public Integer getId() {  
 return id;  
 }  
  
 public void setId(Integer id) {  
 this.id = id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public Integer getCategoryId() {  
 return categoryId;  
 }  
  
 public void setCategoryId(Integer categoryId) {  
 this.categoryId = categoryId;  
 }  
  
 public Double getPrice() {  
 return price;  
 }  
  
 public void setPrice(Double price) {  
 this.price = price;  
 }  
  
 public Integer getQuantity() {  
 return quantity;  
 }  
  
 public void setQuantity(Integer quantity) {  
 this.quantity = quantity;  
 }  
  
 public LocalDate getExpiryDate() {  
 return expiryDate;  
 }  
  
 public void setExpiryDate(LocalDate expiryDate) {  
 this.expiryDate = expiryDate;  
 }  
  
 public Boolean getAvailable() {  
 return isAvailable;  
 }  
  
 public void setAvailable(Boolean available) {  
 isAvailable = available;  
 }  
}

**enum Role:**

package com.Project.Inventory.and.Sales.Management.System.Entity;  
  
public enum Role {  
  
 *Admin*,  
 *User*}

**Sale:**

package com.Project.Inventory.and.Sales.Management.System.Entity;  
  
import java.time.LocalDateTime;  
  
public class Sale {  
 private Integer id;  
 private Integer customerId;  
 private Integer discountId; // Optional  
 private Double totalAmount;  
 private LocalDateTime saleDate;  
  
 public Sale() {  
 }  
  
 public Sale(Integer id, Integer customerId, Integer discountId, Double totalAmount, LocalDateTime saleDate) {  
 this.id = id;  
 this.customerId = customerId;  
 this.discountId = discountId;  
 this.totalAmount = totalAmount;  
 this.saleDate = saleDate;  
 }  
  
 public Integer getId() {  
 return id;  
 }  
  
 public void setId(Integer id) {  
 this.id = id;  
 }  
  
 public Integer getCustomerId() {  
 return customerId;  
 }  
  
 public void setCustomerId(Integer customerId) {  
 this.customerId = customerId;  
 }  
  
 public Integer getDiscountId() {  
 return discountId;  
 }  
  
 public void setDiscountId(Integer discountId) {  
 this.discountId = discountId;  
 }  
  
 public Double getTotalAmount() {  
 return totalAmount;  
 }  
  
 public void setTotalAmount(Double totalAmount) {  
 this.totalAmount = totalAmount;  
 }  
  
 public LocalDateTime getSaleDate() {  
 return saleDate;  
 }  
  
 public void setSaleDate(LocalDateTime saleDate) {  
 this.saleDate = saleDate;  
 }  
}

**SaleItem:**

package com.Project.Inventory.and.Sales.Management.System.Entity;  
  
import jakarta.validation.constraints.NotNull;  
  
public class SaleItem {  
  
 private Integer id;  
  
 @NotNull  
 private Integer saleId;  
  
 @NotNull  
 private Integer productId;  
  
 @NotNull  
 private Integer quantity;  
  
 @NotNull  
 private Double price;  
  
  
 public SaleItem() {}  
  
 public SaleItem(Integer id, Integer saleId, Integer productId, Integer quantity, Double price) {  
 this.id = id;  
 this.saleId = saleId;  
 this.productId = productId;  
 this.quantity = quantity;  
 this.price = price;  
 }  
  
 public Integer getId() {  
 return id;  
 }  
  
 public void setId(Integer id) {  
 this.id = id;  
 }  
  
 public Integer getSaleId() {  
 return saleId;  
 }  
  
 public void setSaleId(Integer saleId) {  
 this.saleId = saleId;  
 }  
  
 public Integer getProductId() {  
 return productId;  
 }  
  
 public void setProductId(Integer productId) {  
 this.productId = productId;  
 }  
  
 public Integer getQuantity() {  
 return quantity;  
 }  
  
 public void setQuantity(Integer quantity) {  
 this.quantity = quantity;  
 }  
  
 public Double getPrice() {  
 return price;  
 }  
  
 public void setPrice(Double price) {  
 this.price = price;  
 }  
}

**Customer:**

package com.Project.Inventory.and.Sales.Management.System.Entity;  
  
import jakarta.validation.constraints.Email;  
import jakarta.validation.constraints.NotNull;  
  
public class Customer {  
  
 private Integer id;  
  
 @NotNull  
 private String username;  
  
 @NotNull  
 private String password;  
  
 @NotNull  
 @Email  
 private String email;  
  
 @NotNull  
 private Role role;  
  
 public Customer() {}  
  
 public Customer(String email, Integer id, String username, String password, Role role) {  
 this.email = email;  
 this.id = id;  
 this.username = username;  
 this.password = password;  
 this.role = role;  
 }  
  
 public Integer getId() {  
 return id;  
 }  
  
 public void setId(Integer id) {  
 this.id = id;  
 }  
  
 public String getUsername() {  
 return username;  
 }  
  
 public void setUsername(String username) {  
 this.username = username;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
  
 public void setEmail(String email) {  
 this.email = email;  
 }  
  
 public Role getRole() {  
 return role;  
 }  
  
 public void setRole(Role role) {  
 this.role = role;  
 }  
}

**Repository:**

**CategoryRepository:**

package com.Project.Inventory.and.Sales.Management.System.Repository;  
  
import com.Project.Inventory.and.Sales.Management.System.Entity.Category;  
import org.springframework.jdbc.core.BeanPropertyRowMapper;  
import org.springframework.jdbc.core.JdbcTemplate;  
import org.springframework.stereotype.Repository;  
  
import javax.sql.DataSource;  
import java.util.List;  
  
@Repository  
public class CategoryRepository {  
 private final JdbcTemplate jdbcTemplate;  
  
 public CategoryRepository(DataSource dataSource) {  
 this.jdbcTemplate = new JdbcTemplate(dataSource);  
 }  
  
 public List<Category> getAll() {  
 return jdbcTemplate.query("SELECT \* FROM categories", new BeanPropertyRowMapper<>(Category.class));  
 }  
  
 public Category getById(int id) {  
 return jdbcTemplate.queryForObject("SELECT \* FROM categories WHERE id = ?", new Object[]{id}, new BeanPropertyRowMapper<>(Category.class));  
 }  
  
 public void save(Category category) {  
 jdbcTemplate.update("INSERT INTO categories (name, description) VALUES (?, ?)", category.getName(), category.getDescription());  
 }  
  
 public void update(Category category) {  
 jdbcTemplate.update("UPDATE categories SET name = ?, description = ? WHERE id = ?",  
 category.getName(), category.getDescription(), category.getId());  
 }  
  
 public void delete(int id) {  
 jdbcTemplate.update("DELETE FROM categories WHERE id = ?", id);  
 }  
}

**DiscountRepository:**

package com.Project.Inventory.and.Sales.Management.System.Repository;  
  
import com.Project.Inventory.and.Sales.Management.System.Entity.Discount;  
import org.springframework.jdbc.core.BeanPropertyRowMapper;  
import org.springframework.jdbc.core.JdbcTemplate;  
import org.springframework.stereotype.Repository;  
  
import javax.sql.DataSource;  
import java.util.List;  
  
@Repository  
public class DiscountRepository {  
 private final JdbcTemplate jdbcTemplate;  
  
 public DiscountRepository(DataSource dataSource) {  
 this.jdbcTemplate = new JdbcTemplate(dataSource);  
 }  
  
 public List<Discount> getActiveDiscounts() {  
 return jdbcTemplate.query("SELECT \* FROM discounts WHERE CURDATE() BETWEEN start\_date AND end\_date", new BeanPropertyRowMapper<>(Discount.class));  
 }  
}

**InvoiceEmailLogRepository:**

package com.Project.Inventory.and.Sales.Management.System.Repository;  
  
import com.Project.Inventory.and.Sales.Management.System.Entity.InvoiceEmailLog;  
import org.springframework.jdbc.core.BeanPropertyRowMapper;  
import org.springframework.jdbc.core.JdbcTemplate;  
import org.springframework.stereotype.Repository;  
  
import javax.sql.DataSource;  
import java.util.List;  
  
@Repository  
public class InvoiceEmailLogRepository {  
 private final JdbcTemplate jdbcTemplate;  
  
 public InvoiceEmailLogRepository(DataSource dataSource) {  
 this.jdbcTemplate = new JdbcTemplate(dataSource);  
 }  
  
 public void save(InvoiceEmailLog log) {  
 jdbcTemplate.update("INSERT INTO invoice\_email\_log (sale\_id, email\_to, sent\_at) VALUES (?, ?, ?)",  
 log.getSaleId(), log.getEmailTo(), log.getSentAt());  
 }  
  
 public List<InvoiceEmailLog> getBySaleId(int saleId) {  
 return jdbcTemplate.query("SELECT \* FROM invoice\_email\_log WHERE sale\_id = ?", new Object[]{saleId}, new BeanPropertyRowMapper<>(InvoiceEmailLog.class));  
 }  
}

**ProductRepository:**

package com.Project.Inventory.and.Sales.Management.System.Repository;  
  
import com.Project.Inventory.and.Sales.Management.System.Entity.Product;  
import org.springframework.jdbc.core.BeanPropertyRowMapper;  
import org.springframework.jdbc.core.JdbcTemplate;  
import org.springframework.stereotype.Repository;  
  
import javax.sql.DataSource;  
import java.util.List;  
  
@Repository  
public class ProductRepository {  
 private final JdbcTemplate jdbcTemplate;  
  
 public ProductRepository(DataSource dataSource) {  
 this.jdbcTemplate = new JdbcTemplate(dataSource);  
 }  
  
 public List<Product> getAll() {  
 return jdbcTemplate.query("SELECT \* FROM products", new BeanPropertyRowMapper<>(Product.class));  
 }  
  
 public Product getById(int id) {  
 return jdbcTemplate.queryForObject("SELECT \* FROM products WHERE id = ?", new Object[]{id}, new BeanPropertyRowMapper<>(Product.class));  
 }  
  
 public void save(Product product) {  
 jdbcTemplate.update("INSERT INTO products (name, category\_id, price, quantity, expiry\_date, is\_available) VALUES (?, ?, ?, ?, ?, ?)",  
 product.getName(), product.getCategoryId(), product.getPrice(), product.getQuantity(), product.getExpiryDate(), product.getAvailable());  
 }  
  
 public void update(Product product) {  
 jdbcTemplate.update("UPDATE products SET name = ?, category\_id = ?, price = ?, quantity = ?, expiry\_date = ?, is\_available = ? WHERE id = ?",  
 product.getName(), product.getCategoryId(), product.getPrice(), product.getQuantity(), product.getExpiryDate(), product.getAvailable(), product.getId());  
 }  
  
 public void delete(int id) {  
 jdbcTemplate.update("DELETE FROM products WHERE id = ?", id);  
 }  
  
 public List<Product> getLowStock(int threshold) {  
 return jdbcTemplate.query("SELECT \* FROM products WHERE quantity < ?", new Object[]{threshold}, new BeanPropertyRowMapper<>(Product.class));  
 }  
}

**SaleItemRepository:**

package com.Project.Inventory.and.Sales.Management.System.Repository;  
  
  
import com.Project.Inventory.and.Sales.Management.System.Entity.SaleItem;  
import org.springframework.jdbc.core.BeanPropertyRowMapper;  
import org.springframework.jdbc.core.JdbcTemplate;  
import org.springframework.stereotype.Repository;  
  
import javax.sql.DataSource;  
import java.util.List;  
  
@Repository  
public class SaleItemRepository {  
 private final JdbcTemplate jdbcTemplate;  
  
 public SaleItemRepository(DataSource dataSource) {  
 this.jdbcTemplate = new JdbcTemplate(dataSource);  
 }  
  
 public List<SaleItem> getBySaleId(int saleId) {  
 return jdbcTemplate.query("SELECT \* FROM sale\_items WHERE sale\_id = ?", new Object[]{saleId}, new BeanPropertyRowMapper<>(SaleItem.class));  
 }  
  
 public void save(SaleItem item) {  
 jdbcTemplate.update("INSERT INTO sale\_items (sale\_id, product\_id, quantity, price) VALUES (?, ?, ?, ?)",  
 item.getSaleId(), item.getProductId(), item.getQuantity(), item.getPrice());  
 }  
}

**SaleRepository:**

package com.Project.Inventory.and.Sales.Management.System.Repository;  
  
import com.Project.Inventory.and.Sales.Management.System.Entity.Sale;  
import org.springframework.jdbc.core.BeanPropertyRowMapper;  
import org.springframework.jdbc.core.JdbcTemplate;  
import org.springframework.stereotype.Repository;  
  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.List;  
  
@Repository  
public class SaleRepository {  
  
 private final JdbcTemplate jdbcTemplate;  
  
 public SaleRepository(JdbcTemplate jdbcTemplate) {  
 this.jdbcTemplate = jdbcTemplate;  
 }  
  
 // Save a new Sale  
 public void save(Sale sale) {  
 String sql = "INSERT INTO sale (customer\_id, discount\_id, total\_amount, sale\_date) VALUES (?, ?, ?, ?)";  
 jdbcTemplate.update(sql,  
 sale.getCustomerId(),  
 sale.getDiscountId(), // can be null  
 sale.getTotalAmount(),  
 sale.getSaleDate());  
 }  
  
 // Get the last inserted ID (for MySQL, PostgreSQL, etc.)  
 public int getLastInsertedId() {  
 return jdbcTemplate.queryForObject("SELECT LAST\_INSERT\_ID()", Integer.class);  
 }  
  
 // Get sale by ID  
 public Sale getById(int id) {  
 String sql = "SELECT \* FROM sale WHERE id = ?";  
 List<Sale> results = jdbcTemplate.query(sql, new Object[]{id}, this::mapRowToSale);  
 return results.isEmpty() ? null : results.get(0);  
 }  
  
 // Get all sales  
 public List<Sale> getAll() {  
 String sql = "SELECT \* FROM sale ORDER BY sale\_date DESC";  
 return jdbcTemplate.query(sql, this::mapRowToSale);  
 }  
  
 // Delete a sale by ID  
 public void delete(int id) {  
 String sql = "DELETE FROM sale WHERE id = ?";  
 jdbcTemplate.update(sql, id);  
 }  
  
 // Custom RowMapper to map manually  
 private Sale mapRowToSale(ResultSet rs, int rowNum) throws SQLException {  
 Sale sale = new Sale();  
 sale.setId(rs.getInt("id"));  
 sale.setCustomerId(rs.getInt("customer\_id"));  
 sale.setDiscountId(rs.getObject("discount\_id") != null ? rs.getInt("discount\_id") : null);  
 sale.setTotalAmount(rs.getDouble("total\_amount"));  
 sale.setSaleDate(rs.getTimestamp("sale\_date").toLocalDateTime());  
 return sale;  
 }  
}

**CustomerRepository:**

package com.Project.Inventory.and.Sales.Management.System.Repository;  
  
import com.Project.Inventory.and.Sales.Management.System.Entity.User;  
import org.springframework.jdbc.core.BeanPropertyRowMapper;  
import org.springframework.jdbc.core.JdbcTemplate;  
import org.springframework.stereotype.Repository;  
  
import javax.sql.DataSource;  
import java.util.List;  
  
@Repository  
public class CustomerRepository {  
 private final JdbcTemplate jdbcTemplate;  
  
 public UserRepository(DataSource dataSource) {  
 this.jdbcTemplate = new JdbcTemplate(dataSource);  
 }  
  
 public List<User> getAll() {  
 return jdbcTemplate.query("SELECT \* FROM users", new BeanPropertyRowMapper<>(User.class));  
 }  
  
 public User getById(int id) {  
 return jdbcTemplate.queryForObject("SELECT \* FROM users WHERE id = ?", new Object[]{id}, new BeanPropertyRowMapper<>(User.class));  
 }  
  
 public void save(CustomerRepository user) {  
 jdbcTemplate.update("INSERT INTO users (username, email, password, role) VALUES (?, ?, ?, ?)",  
 user.getUsername(), user.getEmail(), user.getPassword(), user.getRole());  
 }  
  
 public void update(CustomerRepository user) {  
 jdbcTemplate.update("UPDATE users SET username = ?, email = ?, password = ?, role = ? WHERE id = ?",  
 user.getUsername(), user.getEmail(), user.getPassword(), user.getRole(), user.getId());  
 }  
  
 public void delete(int id) {  
 jdbcTemplate.update("DELETE FROM users WHERE id = ?", id);  
 }  
}

**Service:  
CategoryService:**

package com.Project.Inventory.and.Sales.Management.System.Service;  
  
import com.Project.Inventory.and.Sales.Management.System.Entity.Category;  
import com.Project.Inventory.and.Sales.Management.System.Repository.CategoryRepository;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.List;  
  
@Service  
@Transactional  
public class CategoryService {  
  
 private final CategoryRepository categoryRepository;  
  
 public CategoryService(CategoryRepository categoryRepository) {  
 this.categoryRepository = categoryRepository;  
 }  
  
 public List<Category> getAllCategories() {  
 return categoryRepository.getAll();  
 }  
  
 public Category getCategoryById(int id) {  
 return categoryRepository.getById(id);  
 }  
  
 public void createCategory(Category category) {  
 categoryRepository.save(category);  
 }  
  
 public void updateCategory(Category category) {  
 categoryRepository.update(category);  
 }  
  
 public void deleteCategory(int id) {  
 categoryRepository.delete(id);  
 }  
}

**ProductService:**

package com.Project.Inventory.and.Sales.Management.System.Service;  
  
import com.Project.Inventory.and.Sales.Management.System.Entity.Product;  
import com.Project.Inventory.and.Sales.Management.System.Repository.ProductRepository;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.List;  
  
@Service  
@Transactional  
public class ProductService {  
  
 private final ProductRepository productRepository;  
  
 public ProductService(ProductRepository productRepository) {  
 this.productRepository = productRepository;  
 }  
  
 // Get all products  
 public List<Product> getAllProducts() {  
 return productRepository.getAll();  
 }  
  
 // Get product by id  
 public Product getProductById(int id) {  
 return productRepository.getById(id);  
 }  
  
 // Create new product with validation and availability logic  
 public void createProduct(Product product) {  
 validateProduct(product);  
 updateAvailability(product);  
 productRepository.save(product);  
 }  
  
 // Update product with validation and availability logic  
 public void updateProduct(Product product) {  
 validateProduct(product);  
 updateAvailability(product);  
 productRepository.update(product);  
 }  
  
 // Delete product by id  
 public void deleteProduct(int id) {  
 productRepository.delete(id);  
 }  
  
 // Get products with low stock threshold  
 public List<Product> getLowStockProducts(int threshold) {  
 return productRepository.getLowStock(threshold);  
 }  
  
 // Private helper to validate product fields  
 private void validateProduct(Product product) {  
 if (product.getName() == null || product.getName().isBlank()) {  
 throw new IllegalArgumentException("Product name cannot be null or empty.");  
 }  
 if (product.getCategoryId() == null) {  
 throw new IllegalArgumentException("Product must have a category.");  
 }  
 if (product.getPrice() == null || product.getPrice() < 0) {  
 throw new IllegalArgumentException("Product price must be non-negative.");  
 }  
 if (product.getQuantity() == null || product.getQuantity() < 0) {  
 throw new IllegalArgumentException("Product quantity must be non-negative.");  
 }  
 // Optionally, check expiryDate if set  
 if (product.getExpiryDate() != null && product.getExpiryDate().isBefore(java.time.LocalDate.*now*())) {  
 throw new IllegalArgumentException("Product expiry date cannot be in the past.");  
 }  
 }  
  
 // Private helper to update availability based on quantity  
 private void updateAvailability(Product product) {  
 product.setAvailable(product.getQuantity() > 0);  
 }  
}

**CustomerService:**package com.Project.Inventory.and.Sales.Management.System.Service;  
  
import com.Project.Inventory.and.Sales.Management.System.Entity.Customer;  
import com.Project.Inventory.and.Sales.Management.System.Repository.CustomerRepository;  
import org.springframework.jdbc.core.JdbcTemplate;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.authority.SimpleGrantedAuthority;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.Collection;  
import java.util.List;  
  
@Service  
@Transactional  
public class CustomerService {  
  
 private final CustomerRepository customerRepository;  
  
  
 public CustomerService(CustomerRepository customerRepository, JdbcTemplate jdbcTemplate) {  
 this.customerRepository = customerRepository;  
  
 }  
  
  
  
 private Collection<? extends GrantedAuthority> getAuthorities(String role) {  
 return List.*of*(new SimpleGrantedAuthority("ROLE\_" + role));  
 }  
  
 public List<Customer> getAllUsers() {  
 return customerRepository.getAll();  
 }  
  
 public Customer getUserById(int id) {  
 return customerRepository.getById(id);  
 }  
  
 public void createUser(Customer user) {  
 customerRepository.save(user);  
 }  
  
 public void updateUser(Customer user) {  
 customerRepository.update(user);  
 }  
  
 public void deleteUser(int id) {  
 customerRepository.delete(id);  
 }  
}

**DiscountService:**

package com.Project.Inventory.and.Sales.Management.System.Service;  
  
import com.Project.Inventory.and.Sales.Management.System.Entity.Discount;  
import com.Project.Inventory.and.Sales.Management.System.Repository.DiscountRepository;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.List;  
  
@Service  
@Transactional  
public class DiscountService {  
  
 private final DiscountRepository discountRepository;  
  
 public DiscountService(DiscountRepository discountRepository) {  
 this.discountRepository = discountRepository;  
 }  
  
 // Retrieve all discounts (admin)  
 public List<Discount> getAllDiscounts() {  
 return discountRepository.getAll();  
 }  
  
 // Retrieve active discounts (public)  
 public List<Discount> getActiveDiscounts() {  
 return discountRepository.getActiveDiscounts();  
 }  
  
 // Get discount by ID  
 public Discount getDiscountById(int id) {  
 return discountRepository.getById(id);  
 }  
  
 // Create new discount (admin)  
 public void createDiscount(Discount discount) {  
 validateDiscount(discount);  
 discountRepository.save(discount);  
 }  
  
 // Update existing discount (admin)  
 public void updateDiscount(Discount discount) {  
 validateDiscount(discount);  
 discountRepository.update(discount);  
 }  
  
 // Delete discount by ID (admin)  
 public void deleteDiscount(int id) {  
 discountRepository.delete(id);  
 }  
  
 // Validation method to check discount data  
 private void validateDiscount(Discount discount) {  
 if (discount.getPercentage() == null || discount.getPercentage() <= 0 || discount.getPercentage() > 100) {  
 throw new IllegalArgumentException("Discount percentage must be between 1 and 100.");  
 }  
 if (discount.getStartDate() == null || discount.getEndDate() == null) {  
 throw new IllegalArgumentException("Start date and end date cannot be null.");  
 }  
 if (discount.getStartDate().isAfter(discount.getEndDate())) {  
 throw new IllegalArgumentException("Start date cannot be after end date.");  
 }  
 }  
}

**InvoiceEmailLogService:**

package com.Project.Inventory.and.Sales.Management.System.Service;  
  
import com.Project.Inventory.and.Sales.Management.System.Entity.InvoiceEmailLog;  
import com.Project.Inventory.and.Sales.Management.System.Repository.InvoiceEmailLogRepository;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.List;  
  
@Service  
@Transactional  
public class InvoiceEmailLogService {  
  
 private final InvoiceEmailLogRepository invoiceEmailLogRepository;  
  
 public InvoiceEmailLogService(InvoiceEmailLogRepository invoiceEmailLogRepository) {  
 this.invoiceEmailLogRepository = invoiceEmailLogRepository;  
 }  
  
 */\*\*  
 \* Save a new email log entry after an invoice email is sent.  
 \*  
 \* @param log the InvoiceEmailLog entity to save  
 \*/* public void saveEmailLog(InvoiceEmailLog log) {  
 if (log == null) {  
 throw new IllegalArgumentException("InvoiceEmailLog cannot be null");  
 }  
 if (log.getSaleId() == null) {  
 throw new IllegalArgumentException("Sale ID must be provided");  
 }  
 if (log.getEmailTo() == null || log.getEmailTo().isBlank()) {  
 throw new IllegalArgumentException("Recipient email must be provided");  
 }  
 if (log.getSentAt() == null) {  
 throw new IllegalArgumentException("Sent timestamp must be provided");  
 }  
 invoiceEmailLogRepository.save(log);  
 }  
  
 */\*\*  
 \* Retrieve all invoice email logs related to a specific sale.  
 \*  
 \* @param saleId the ID of the sale  
 \* @return list of InvoiceEmailLog entries for the sale  
 \*/* public List<InvoiceEmailLog> getEmailLogsBySaleId(int saleId) {  
 return invoiceEmailLogRepository.getBySaleId(saleId);  
 }  
}

**SalesItemService:**

package com.Project.Inventory.and.Sales.Management.System.Service;  
  
import com.Project.Inventory.and.Sales.Management.System.Entity.SaleItem;  
import com.Project.Inventory.and.Sales.Management.System.Repository.SaleItemRepository;  
import com.Project.Inventory.and.Sales.Management.System.Repository.ProductRepository;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.List;  
  
@Service  
@Transactional  
public class SaleItemService {  
  
 private final SaleItemRepository saleItemRepository;  
 private final ProductRepository productRepository;  
  
 public SaleItemService(SaleItemRepository saleItemRepository, ProductRepository productRepository) {  
 this.saleItemRepository = saleItemRepository;  
 this.productRepository = productRepository;  
 }  
  
 // Get all sale items by saleId  
 public List<SaleItem> getItemsBySaleId(int saleId) {  
 return saleItemRepository.getBySaleId(saleId);  
 }  
  
 // Add a sale item with validation and stock update  
 public void addSaleItem(SaleItem saleItem) {  
 // Check if product exists  
 var product = productRepository.getById(saleItem.getProductId());  
 if (product == null) {  
 throw new IllegalArgumentException("Product not found with id: " + saleItem.getProductId());  
 }  
  
 // Check if enough stock is available  
 if (product.getQuantity() < saleItem.getQuantity()) {  
 throw new IllegalArgumentException("Insufficient stock for product: " + product.getName());  
 }  
  
 // Update product stock  
 product.setQuantity(product.getQuantity() - saleItem.getQuantity());  
 product.setAvailable(product.getQuantity() > 0);  
 productRepository.update(product);  
  
 // Save sale item  
 saleItemRepository.save(saleItem);  
 }  
  
 // Update sale item quantity and price, adjust stock accordingly  
 public void updateSaleItem(SaleItem saleItem) {  
 SaleItem existingItem = saleItemRepository.getBySaleId(saleItem.getSaleId())  
 .stream()  
 .filter(item -> item.getId().equals(saleItem.getId()))  
 .findFirst()  
 .orElseThrow(() -> new IllegalArgumentException("SaleItem not found with id: " + saleItem.getId()));  
  
 var product = productRepository.getById(saleItem.getProductId());  
 if (product == null) {  
 throw new IllegalArgumentException("Product not found with id: " + saleItem.getProductId());  
 }  
  
 int quantityDiff = saleItem.getQuantity() - existingItem.getQuantity();  
  
 if (quantityDiff > 0 && product.getQuantity() < quantityDiff) {  
 throw new IllegalArgumentException("Insufficient stock to increase quantity for product: " + product.getName());  
 }  
  
 // Update product stock based on quantity difference  
 product.setQuantity(product.getQuantity() - quantityDiff);  
 product.setAvailable(product.getQuantity() > 0);  
 productRepository.update(product);  
  
 // Update sale item  
 saleItemRepository.update(saleItem);  
 }  
  
 // Delete sale item and restore product stock  
 public void deleteSaleItem(int saleItemId) {  
 // Fetch the sale item to get productId and quantity  
 SaleItem saleItem = saleItemRepository.getAll()  
 .stream()  
 .filter(item -> item.getId().equals(saleItemId))  
 .findFirst()  
 .orElseThrow(() -> new IllegalArgumentException("SaleItem not found with id: " + saleItemId));  
  
 var product = productRepository.getById(saleItem.getProductId());  
 if (product != null) {  
 product.setQuantity(product.getQuantity() + saleItem.getQuantity());  
 product.setAvailable(product.getQuantity() > 0);  
 productRepository.update(product);  
 }  
  
 saleItemRepository.delete(saleItemId);  
 }  
}

**SaleService:**

package com.Project.Inventory.and.Sales.Management.System.Service;  
  
import com.Project.Inventory.and.Sales.Management.System.DTO.SaleRequestDTO;  
import com.Project.Inventory.and.Sales.Management.System.Entity.\*;  
import com.Project.Inventory.and.Sales.Management.System.Repository.\*;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.time.LocalDateTime;  
import java.time.chrono.ChronoLocalDate;  
import java.util.List;  
  
@Service  
@Transactional  
public class SaleService {  
  
 private final SaleRepository saleRepository;  
 private final SaleItemService saleItemService;  
 private final ProductRepository productRepository;  
 private final DiscountRepository discountRepository;  
 private final InvoiceEmailLogService invoiceEmailLogService;  
  
 public SaleService(SaleRepository saleRepository,  
 SaleItemService saleItemService,  
 ProductRepository productRepository,  
 DiscountRepository discountRepository,  
 InvoiceEmailLogService invoiceEmailLogService) {  
 this.saleRepository = saleRepository;  
 this.saleItemService = saleItemService;  
 this.productRepository = productRepository;  
 this.discountRepository = discountRepository;  
 this.invoiceEmailLogService = invoiceEmailLogService;  
 }  
  
 public void createSale(SaleRequestDTO request) {  
 if (request.getItems() == null || request.getItems().isEmpty()) {  
 throw new IllegalArgumentException("Sale must contain at least one item.");  
 }  
  
 // Calculate total  
 double total = request.getItems().stream()  
 .mapToDouble(item -> item.getPrice() \* item.getQuantity())  
 .sum();  
  
 // Apply discount if present  
 if (request.getDiscountId() != null) {  
 Discount discount = discountRepository.getById(request.getDiscountId());  
 if (discount != null && discount.getStartDate().isBefore(ChronoLocalDate.*from*(LocalDateTime.*now*()))  
 && discount.getEndDate().isAfter(ChronoLocalDate.*from*(LocalDateTime.*now*()))) {  
 total = total - (total \* discount.getPercentage() / 100.0);  
 }  
 }  
  
 // Create and save Sale  
 Sale sale = new Sale();  
 sale.setCustomerId(request.getCustomerId());  
 sale.setDiscountId(request.getDiscountId());  
 sale.setTotalAmount(total);  
 sale.setSaleDate(request.getSaleDate() != null ? request.getSaleDate() : LocalDateTime.*now*());  
  
 saleRepository.save(sale);  
  
  
  
 // Save each sale item  
 for (SaleItem item : request.getItems()) {  
 item.setSaleId(sale.getId());  
 saleItemService.addSaleItem(item); // handles product stock update  
 }  
  
  
 InvoiceEmailLog log = new InvoiceEmailLog();  
 log.setSaleId(sale.getId());  
 log.setEmailTo("nasifsadnanc1@gmail.com");  
 log.setSentAt(LocalDateTime.*now*());  
 invoiceEmailLogService.saveEmailLog(log);  
 }  
  
 public List<Sale> getAllSales() {  
 return saleRepository.getAll();  
 }  
  
 public Sale getSaleById(int id) {  
 return saleRepository.getById(id);  
 }  
  
 public void deleteSale(int id) {  
 // 1. Get all sale items for this sale  
 List<SaleItem> saleItems = saleItemService.getItemsBySaleId(id);  
  
 // 2. Restore stock for each product  
 for (SaleItem item : saleItems) {  
 var product = productRepository.getById(item.getProductId());  
 if (product != null) {  
 product.setQuantity(product.getQuantity() + item.getQuantity());  
 product.setAvailable(product.getQuantity() > 0);  
 productRepository.update(product);  
 }  
 }  
  
 // 3. Delete sale items  
 for (SaleItem item : saleItems) {  
 saleItemService.deleteSaleItem(item.getId());  
 }  
  
 // 4. Finally delete the sale  
 saleRepository.delete(id);  
 }  
}

**API:**

**CategoryApi:**

package com.Project.Inventory.and.Sales.Management.System.API;  
  
import com.Project.Inventory.and.Sales.Management.System.Entity.Category;  
import com.Project.Inventory.and.Sales.Management.System.Service.CategoryService;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/categories")  
public class CategoryApi {  
  
 private final CategoryService categoryService;  
  
 public CategoryApi(CategoryService categoryService) {  
 this.categoryService = categoryService;  
 }  
  
 @GetMapping  
 public ResponseEntity<List<Category>> getAllCategories() {  
 return ResponseEntity.*ok*(categoryService.getAllCategories());  
 }  
  
 @GetMapping("/{id}")  
 public ResponseEntity<Category> getCategoryById(@PathVariable int id) {  
 Category category = categoryService.getCategoryById(id);  
 if (category == null) {  
 return ResponseEntity.*notFound*().build();  
 }  
 return ResponseEntity.*ok*(category);  
 }  
  
 @PostMapping  
 public ResponseEntity<String> createCategory(@RequestBody Category category) {  
 categoryService.createCategory(category);  
 return ResponseEntity.*ok*("Category created successfully.");  
 }  
  
 @PutMapping("/{id}")  
 public ResponseEntity<String> updateCategory(@PathVariable int id, @RequestBody Category category) {  
 category.setId(id);  
 categoryService.updateCategory(category);  
 return ResponseEntity.*ok*("Category updated successfully.");  
 }  
  
 @DeleteMapping("/{id}")  
 public ResponseEntity<String> deleteCategory(@PathVariable int id) {  
 categoryService.deleteCategory(id);  
 return ResponseEntity.*ok*("Category deleted successfully.");  
 }  
}

**CustomerApi:**

package com.Project.Inventory.and.Sales.Management.System.API;  
  
  
import com.Project.Inventory.and.Sales.Management.System.Entity.Customer;  
import com.Project.Inventory.and.Sales.Management.System.Service.CustomerService;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/customers")  
public class CustomerApi {  
  
 private final CustomerService customerService;  
  
 public CustomerApi(CustomerService customerService) {  
 this.customerService = customerService;  
 }  
  
 // GET all customers  
 @GetMapping  
 public ResponseEntity<List<Customer>> getAllCustomers() {  
 return ResponseEntity.*ok*(customerService.getAllUsers());  
 }  
  
 // GET customer by ID  
 @GetMapping("/{id}")  
 public ResponseEntity<Customer> getCustomerById(@PathVariable int id) {  
 Customer user = customerService.getUserById(id);  
 if (user == null) {  
 return ResponseEntity.*notFound*().build();  
 }  
 return ResponseEntity.*ok*(user);  
 }  
  
 // POST create customer  
 @PostMapping  
 public ResponseEntity<String> createCustomer(@RequestBody Customer customer) {  
 try {  
 customerService.createUser(customer);  
 return ResponseEntity.*ok*("Customer created successfully.");  
 } catch (Exception e) {  
 return ResponseEntity.*badRequest*().body("Failed to create customer: " + e.getMessage());  
 }  
 }  
  
 // PUT update customer  
 @PutMapping("/{id}")  
 public ResponseEntity<String> updateCustomer(@PathVariable int id, @RequestBody Customer customer) {  
 customer.setId(id);  
 try {  
 customerService.updateUser(customer);  
 return ResponseEntity.*ok*("Customer updated successfully.");  
 } catch (Exception e) {  
 return ResponseEntity.*badRequest*().body("Failed to update customer: " + e.getMessage());  
 }  
 }  
  
 // DELETE customer  
 @DeleteMapping("/{id}")  
 public ResponseEntity<String> deleteCustomer(@PathVariable int id) {  
 customerService.deleteUser(id);  
 return ResponseEntity.*ok*("Customer deleted successfully.");  
 }  
}

**DiscountApi:**

package com.Project.Inventory.and.Sales.Management.System.API;  
  
  
import com.Project.Inventory.and.Sales.Management.System.Entity.Discount;  
import com.Project.Inventory.and.Sales.Management.System.Service.DiscountService;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/discounts")  
public class DiscountApi {  
  
 private final DiscountService discountService;  
  
 public DiscountApi(DiscountService discountService) {  
 this.discountService = discountService;  
 }  
  
 // GET all discounts (Admin)  
 @GetMapping  
 public ResponseEntity<List<Discount>> getAllDiscounts() {  
 return ResponseEntity.*ok*(discountService.getAllDiscounts());  
 }  
  
 // GET active discounts (Public)  
 @GetMapping("/active")  
 public ResponseEntity<List<Discount>> getActiveDiscounts() {  
 return ResponseEntity.*ok*(discountService.getActiveDiscounts());  
 }  
  
 // GET discount by ID  
 @GetMapping("/{id}")  
 public ResponseEntity<Discount> getDiscountById(@PathVariable int id) {  
 Discount discount = discountService.getDiscountById(id);  
 if (discount == null) {  
 return ResponseEntity.*notFound*().build();  
 }  
 return ResponseEntity.*ok*(discount);  
 }  
  
 // POST create discount (Admin)  
 @PostMapping  
 public ResponseEntity<String> createDiscount(@RequestBody Discount discount) {  
 try {  
 discountService.createDiscount(discount);  
 return ResponseEntity.*ok*("Discount created successfully.");  
 } catch (IllegalArgumentException e) {  
 return ResponseEntity.*badRequest*().body(e.getMessage());  
 }  
 }  
  
 // PUT update discount (Admin)  
 @PutMapping("/{id}")  
 public ResponseEntity<String> updateDiscount(@PathVariable int id, @RequestBody Discount discount) {  
 discount.setId(id);  
 try {  
 discountService.updateDiscount(discount);  
 return ResponseEntity.*ok*("Discount updated successfully.");  
 } catch (IllegalArgumentException e) {  
 return ResponseEntity.*badRequest*().body(e.getMessage());  
 }  
 }  
  
 // DELETE discount (Admin)  
 @DeleteMapping("/{id}")  
 public ResponseEntity<String> deleteDiscount(@PathVariable int id) {  
 discountService.deleteDiscount(id);  
 return ResponseEntity.*ok*("Discount deleted successfully.");  
 }  
}

**InvoiceEmailLogApi:**

package com.Project.Inventory.and.Sales.Management.System.API;  
  
import com.Project.Inventory.and.Sales.Management.System.Entity.InvoiceEmailLog;  
import com.Project.Inventory.and.Sales.Management.System.Service.InvoiceEmailLogService;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/invoice-email-logs")  
public class InvoiceEmailLogApi{  
  
 private final InvoiceEmailLogService invoiceEmailLogService;  
  
 public InvoiceEmailLogApi(InvoiceEmailLogService invoiceEmailLogService) {  
 this.invoiceEmailLogService = invoiceEmailLogService;  
 }  
  
 // GET all email logs for a specific sale  
 @GetMapping("/by-sale/{saleId}")  
 public ResponseEntity<List<InvoiceEmailLog>> getLogsBySaleId(@PathVariable int saleId) {  
 return ResponseEntity.*ok*(invoiceEmailLogService.getEmailLogsBySaleId(saleId));  
 }  
  
 // POST log a new invoice email (optional admin/manual logging)  
 @PostMapping  
 public ResponseEntity<String> logInvoiceEmail(@RequestBody InvoiceEmailLog log) {  
 try {  
 invoiceEmailLogService.saveEmailLog(log);  
 return ResponseEntity.*ok*("Invoice email log saved successfully.");  
 } catch (IllegalArgumentException e) {  
 return ResponseEntity.*badRequest*().body("Failed to save log: " + e.getMessage());  
 }  
 }  
}

**ProductApi:**

package com.Project.Inventory.and.Sales.Management.System.API;  
  
  
import com.Project.Inventory.and.Sales.Management.System.Entity.Product;  
import com.Project.Inventory.and.Sales.Management.System.Service.ProductService;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/products")  
public class ProductApi {  
  
 private final ProductService productService;  
  
 public ProductApi(ProductService productService) {  
 this.productService = productService;  
 }  
  
 // GET all products  
 @GetMapping  
 public ResponseEntity<List<Product>> getAllProducts() {  
 return ResponseEntity.*ok*(productService.getAllProducts());  
 }  
  
 // GET product by ID  
 @GetMapping("/{id}")  
 public ResponseEntity<Product> getProductById(@PathVariable int id) {  
 Product product = productService.getProductById(id);  
 if (product == null) {  
 return ResponseEntity.*notFound*().build();  
 }  
 return ResponseEntity.*ok*(product);  
 }  
  
 // GET products with low stock  
 @GetMapping("/low-stock")  
 public ResponseEntity<List<Product>> getLowStockProducts(@RequestParam(defaultValue = "10") int threshold) {  
 return ResponseEntity.*ok*(productService.getLowStockProducts(threshold));  
 }  
  
 // POST create product  
 @PostMapping  
 public ResponseEntity<String> createProduct(@RequestBody Product product) {  
 try {  
 productService.createProduct(product);  
 return ResponseEntity.*ok*("Product created successfully.");  
 } catch (IllegalArgumentException e) {  
 return ResponseEntity.*badRequest*().body(e.getMessage());  
 }  
 }  
  
 // PUT update product  
 @PutMapping("/{id}")  
 public ResponseEntity<String> updateProduct(@PathVariable int id, @RequestBody Product product) {  
 try {  
 product.setId(id);  
 productService.updateProduct(product);  
 return ResponseEntity.*ok*("Product updated successfully.");  
 } catch (IllegalArgumentException e) {  
 return ResponseEntity.*badRequest*().body(e.getMessage());  
 }  
 }  
  
 // DELETE product  
 @DeleteMapping("/{id}")  
 public ResponseEntity<String> deleteProduct(@PathVariable int id) {  
 productService.deleteProduct(id);  
 return ResponseEntity.*ok*("Product deleted successfully.");  
 }  
}

**SaleApi:**

package com.Project.Inventory.and.Sales.Management.System.API;  
  
  
import com.Project.Inventory.and.Sales.Management.System.DTO.SaleRequestDTO;  
import com.Project.Inventory.and.Sales.Management.System.Entity.Sale;  
import com.Project.Inventory.and.Sales.Management.System.Service.SaleService;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/sales")  
public class SaleApi {  
  
 private final SaleService saleService;  
  
 public SaleApi(SaleService saleService) {  
 this.saleService = saleService;  
 }  
  
 // GET all sales  
 @GetMapping  
 public ResponseEntity<List<Sale>> getAllSales() {  
 return ResponseEntity.*ok*(saleService.getAllSales());  
 }  
  
 // GET sale by ID  
 @GetMapping("/{id}")  
 public ResponseEntity<Sale> getSaleById(@PathVariable int id) {  
 Sale sale = saleService.getSaleById(id);  
 if (sale == null) {  
 return ResponseEntity.*notFound*().build();  
 }  
 return ResponseEntity.*ok*(sale);  
 }  
  
 // POST create a new sale (includes sale items, optional discount)  
 @PostMapping  
 public ResponseEntity<String> createSale(@RequestBody SaleRequestDTO saleRequestDTO) {  
 try {  
 saleService.createSale(saleRequestDTO);  
 return ResponseEntity.*ok*("Sale created successfully with invoice email logged.");  
 } catch (IllegalArgumentException e) {  
 return ResponseEntity.*badRequest*().body("Sale creation failed: " + e.getMessage());  
 }  
 }  
  
 // DELETE sale (restores product stock and removes items)  
 @DeleteMapping("/{id}")  
 public ResponseEntity<String> deleteSale(@PathVariable int id) {  
 saleService.deleteSale(id);  
 return ResponseEntity.*ok*("Sale deleted and stock restored successfully.");  
 }  
}

**SaleItemApi:**

package com.Project.Inventory.and.Sales.Management.System.API;  
  
  
import com.Project.Inventory.and.Sales.Management.System.Entity.SaleItem;  
import com.Project.Inventory.and.Sales.Management.System.Service.SaleItemService;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/sale-items")  
public class SaleItemApi {  
  
 private final SaleItemService saleItemService;  
  
 public SaleItemApi(SaleItemService saleItemService) {  
 this.saleItemService = saleItemService;  
 }  
  
 // GET all items for a specific sale  
 @GetMapping("/by-sale/{saleId}")  
 public ResponseEntity<List<SaleItem>> getItemsBySaleId(@PathVariable int saleId) {  
 return ResponseEntity.*ok*(saleItemService.getItemsBySaleId(saleId));  
 }  
  
 // POST add a new sale item  
 @PostMapping  
 public ResponseEntity<String> addSaleItem(@RequestBody SaleItem saleItem) {  
 try {  
 saleItemService.addSaleItem(saleItem);  
 return ResponseEntity.*ok*("Sale item added successfully.");  
 } catch (IllegalArgumentException e) {  
 return ResponseEntity.*badRequest*().body(e.getMessage());  
 }  
 }  
  
 // PUT update an existing sale item  
 @PutMapping("/{id}")  
 public ResponseEntity<String> updateSaleItem(@PathVariable int id, @RequestBody SaleItem saleItem) {  
 try {  
 saleItem.setId(id);  
 saleItemService.updateSaleItem(saleItem);  
 return ResponseEntity.*ok*("Sale item updated successfully.");  
 } catch (IllegalArgumentException e) {  
 return ResponseEntity.*badRequest*().body(e.getMessage());  
 }  
 }  
  
 // DELETE a sale item  
 @DeleteMapping("/{id}")  
 public ResponseEntity<String> deleteSaleItem(@PathVariable int id) {  
 try {  
 saleItemService.deleteSaleItem(id);  
 return ResponseEntity.*ok*("Sale item deleted successfully.");  
 } catch (IllegalArgumentException e) {  
 return ResponseEntity.*badRequest*().body(e.getMessage());  
 }  
 }  
}