// ===== ÜKSUSED =====

// Toote üksus

@Entity

@Table(name = "products")

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String description;

private BigDecimal price;

private Integer stock;

@ManyToOne

private Category category;

// getterid, setterid

}

// Kategooria üksus

@Entity

@Table(name = "categories")

public class Category {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@OneToMany(mappedBy = "category")

private List<Product> products;

// getterid, setterid

}

// Kasutaja üksus

@Entity

@Table(name = "users")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String username;

private String email;

private String password; // krüpteeritud

@OneToMany(mappedBy = "user")

private List<Order> orders;

// getterid, setterid

}

// Tellimuse üksus

@Entity

@Table(name = "orders")

public class Order {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private LocalDateTime orderDate;

private String status;

private BigDecimal totalAmount;

@ManyToOne

private User user;

@OneToMany(mappedBy = "order")

private List<OrderItem> items;

// getterid, setterid

}

// Tellimuse ühiku üksus

@Entity

@Table(name = "order\_items")

public class OrderItem {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private Integer quantity;

private BigDecimal price;

@ManyToOne

private Order order;

@ManyToOne

private Product product;

// getterid, setterid

}

// ===== HOIDLAD =====

// Toote hoidla

public interface ProductRepository extends JpaRepository<Product, Long> {

List<Product> findByCategoryId(Long categoryId);

List<Product> findByNameContainingIgnoreCase(String keyword);

}

// Kategooria hoidla

public interface CategoryRepository extends JpaRepository<Category, Long> {

}

// Kasutaja hoidla

public interface UserRepository extends JpaRepository<User, Long> {

Optional<User> findByUsername(String username);

Optional<User> findByEmail(String email);

}

// Tellimuse hoidla

public interface OrderRepository extends JpaRepository<Order, Long> {

List<Order> findByUserId(Long userId);

}

// ===== TEENUSED =====

// Toote teenus

@Service

public class ProductService {

private final ProductRepository productRepository;

public ProductService(ProductRepository productRepository) {

this.productRepository = productRepository;

}

public List<ProductDTO> getAllProducts() {

return productRepository.findAll().stream()

.map(this::mapToDTO)

.collect(Collectors.toList());

}

public ProductDTO getProductById(Long id) {

return productRepository.findById(id)

.map(this::mapToDTO)

.orElseThrow(() -> new ResourceNotFoundException("Product not found"));

}

public List<ProductDTO> getProductsByCategory(Long categoryId) {

return productRepository.findByCategoryId(categoryId).stream()

.map(this::mapToDTO)

.collect(Collectors.toList());

}

public List<ProductDTO> searchProducts(String keyword) {

return productRepository.findByNameContainingIgnoreCase(keyword).stream()

.map(this::mapToDTO)

.collect(Collectors.toList());

}

// Ülejäänud CRUD operatsioonid

private ProductDTO mapToDTO(Product product) {

// mappimine ProductDTO-ks

}

}

// Tellimuse teenus

@Service

public class OrderService {

private final OrderRepository orderRepository;

private final ProductRepository productRepository;

private final UserRepository userRepository;

// konstruktor

@Transactional

public OrderDTO createOrder(OrderCreateDTO orderCreateDTO) {

User user = userRepository.findById(orderCreateDTO.getUserId())

.orElseThrow(() -> new ResourceNotFoundException("User not found"));

Order order = new Order();

order.setUser(user);

order.setOrderDate(LocalDateTime.now());

order.setStatus("PENDING");

List<OrderItem> orderItems = new ArrayList<>();

BigDecimal totalAmount = BigDecimal.ZERO;

for (OrderItemDTO itemDTO : orderCreateDTO.getItems()) {

Product product = productRepository.findById(itemDTO.getProductId())

.orElseThrow(() -> new ResourceNotFoundException("Product not found"));

if (product.getStock() < itemDTO.getQuantity()) {

throw new InsufficientStockException("Not enough stock for product: " + product.getName());

}

OrderItem orderItem = new OrderItem();

orderItem.setOrder(order);

orderItem.setProduct(product);

orderItem.setQuantity(itemDTO.getQuantity());

orderItem.setPrice(product.getPrice());

orderItems.add(orderItem);

// Uuenda toote laovarude seisu

product.setStock(product.getStock() - itemDTO.getQuantity());

productRepository.save(product);

// Arvuta ühiku kogusumma ja lisa see tellimuse kogusummale

BigDecimal itemTotal = product.getPrice().multiply(BigDecimal.valueOf(itemDTO.getQuantity()));

totalAmount = totalAmount.add(itemTotal);

}

order.setItems(orderItems);

order.setTotalAmount(totalAmount);

Order savedOrder = orderRepository.save(order);

return mapToDTO(savedOrder);

}

// Teised tellimuse operatsioonid

}

// ===== KONTROLLERID =====

// Toote kontroller

@RestController

@RequestMapping("/api/products")

public class ProductController {

private final ProductService productService;

public ProductController(ProductService productService) {

this.productService = productService;

}

@GetMapping

public ResponseEntity<List<ProductDTO>> getAllProducts() {

return ResponseEntity.ok(productService.getAllProducts());

}

@GetMapping("/{id}")

public ResponseEntity<ProductDTO> getProductById(@PathVariable Long id) {

return ResponseEntity.ok(productService.getProductById(id));

}

@GetMapping("/category/{categoryId}")

public ResponseEntity<List<ProductDTO>> getProductsByCategory(@PathVariable Long categoryId) {

return ResponseEntity.ok(productService.getProductsByCategory(categoryId));

}

@GetMapping("/search")

public ResponseEntity<List<ProductDTO>> searchProducts(@RequestParam String keyword) {

return ResponseEntity.ok(productService.searchProducts(keyword));

}

// Ülejäänud CRUD lõpp-punktid

}

// Tellimuse kontroller

@RestController

@RequestMapping("/api/orders")

public class OrderController {

private final OrderService orderService;

// konstruktor

@PostMapping

public ResponseEntity<OrderDTO> createOrder(@RequestBody @Valid OrderCreateDTO orderCreateDTO) {

return ResponseEntity.status(HttpStatus.CREATED).body(orderService.createOrder(orderCreateDTO));

}

@GetMapping("/user/{userId}")

public ResponseEntity<List<OrderDTO>> getUserOrders(@PathVariable Long userId) {

return ResponseEntity.ok(orderService.getUserOrders(userId));

}

// Teised tellimuse lõpp-punktid

}

// ===== KONFIGURATSIOONI KLASS =====

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private UserDetailsServiceImpl userDetailsService;

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.cors().and().csrf().disable()

.authorizeRequests()

.antMatchers("/api/auth/\*\*").permitAll()

.antMatchers(HttpMethod.GET, "/api/products/\*\*").permitAll()

.antMatchers(HttpMethod.GET, "/api/categories/\*\*").permitAll()

.anyRequest().authenticated()

.and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)

.and()

.addFilterBefore(jwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);

}

// JWT filter, password encoder jne

}

// ===== RAKENDUSKLASS =====

@SpringBootApplication

public class EShopApplication {

public static void main(String[] args) {

SpringApplication.run(EShopApplication.class, args);

}

}