eShop Website

Archontis E. Kostis

Assignment for Cloud Development

HELLENIC REPUBLIC
UNIVERSITY OF MACEDONIA

Table of contents

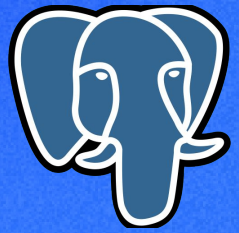
01

Tech Stack & Code Structure

Back End Tech Stack

Spring

Backend Framework



PostgreSQL

Database

Jakarta Validation

Validation Framework

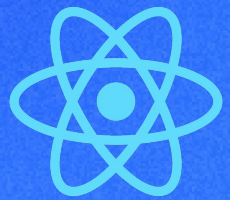
Swagger UI

API Documentation

Front End Tech Stack

TypeScript

Programming Language



React

Component Framework

Material UI

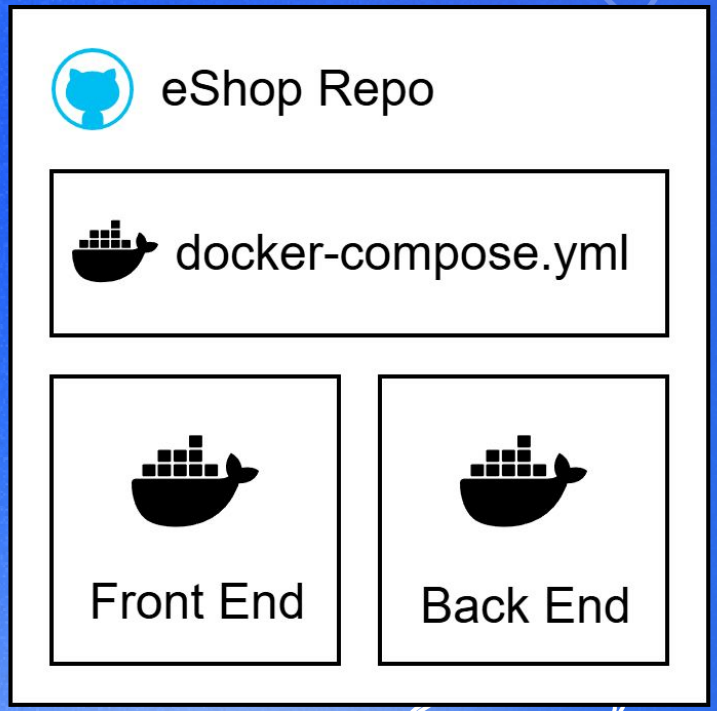
React Component Library

Orval

Type Generation from Backend

Code Structure

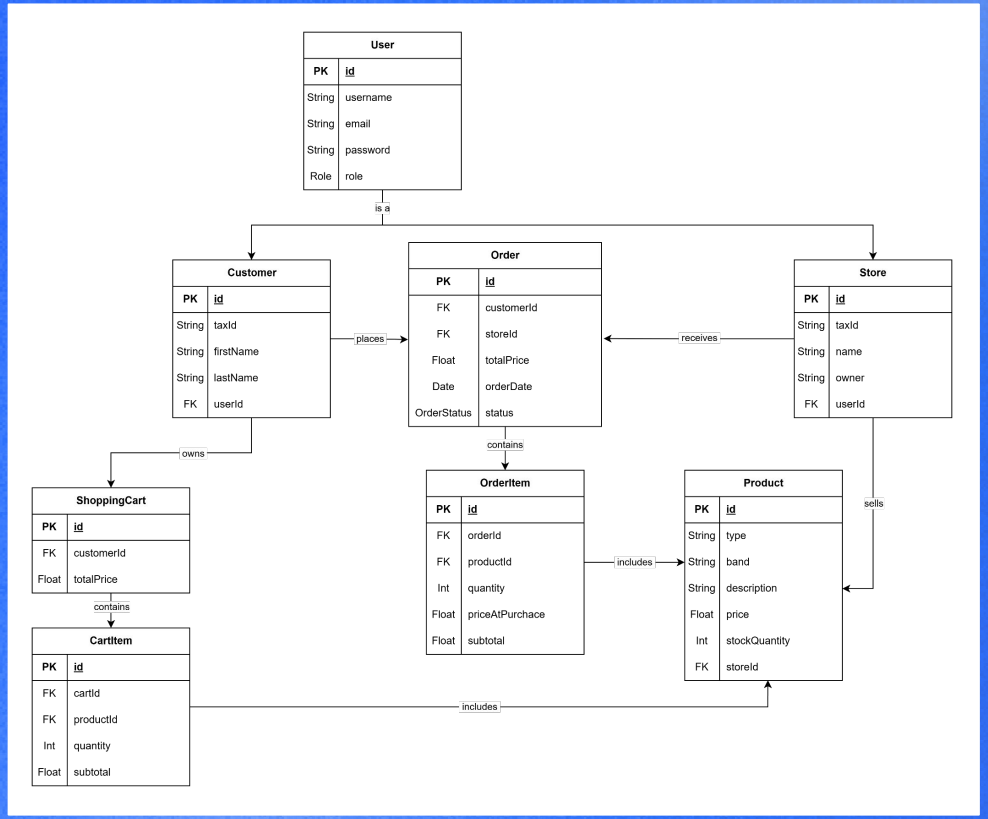
- **Monorepo** hosted on GitHub
- **Backend** folder contains all server side logic
- **Frontend** folder contains client-side code and assets
- Both folders have their own **dockerfile** that dictates how the component is containerized.
- **docker-compose.yml** file configures the all needed services and orchestrates deployment and management of the whole application.



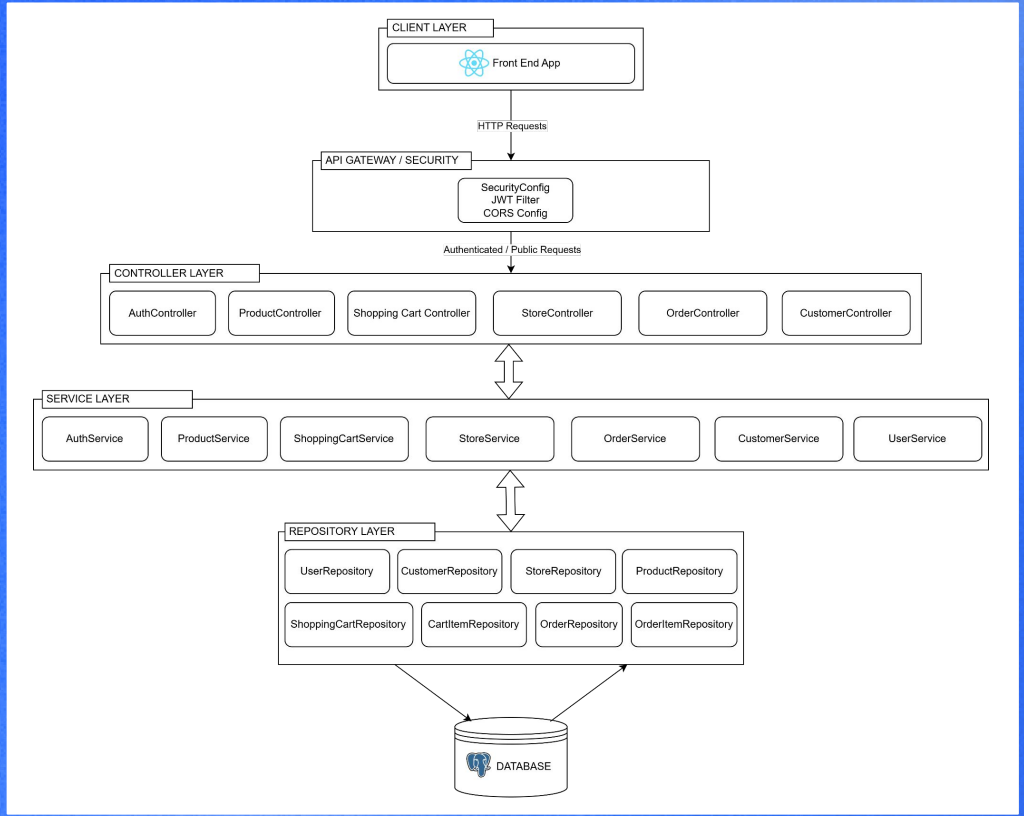
03

Backend Architecture

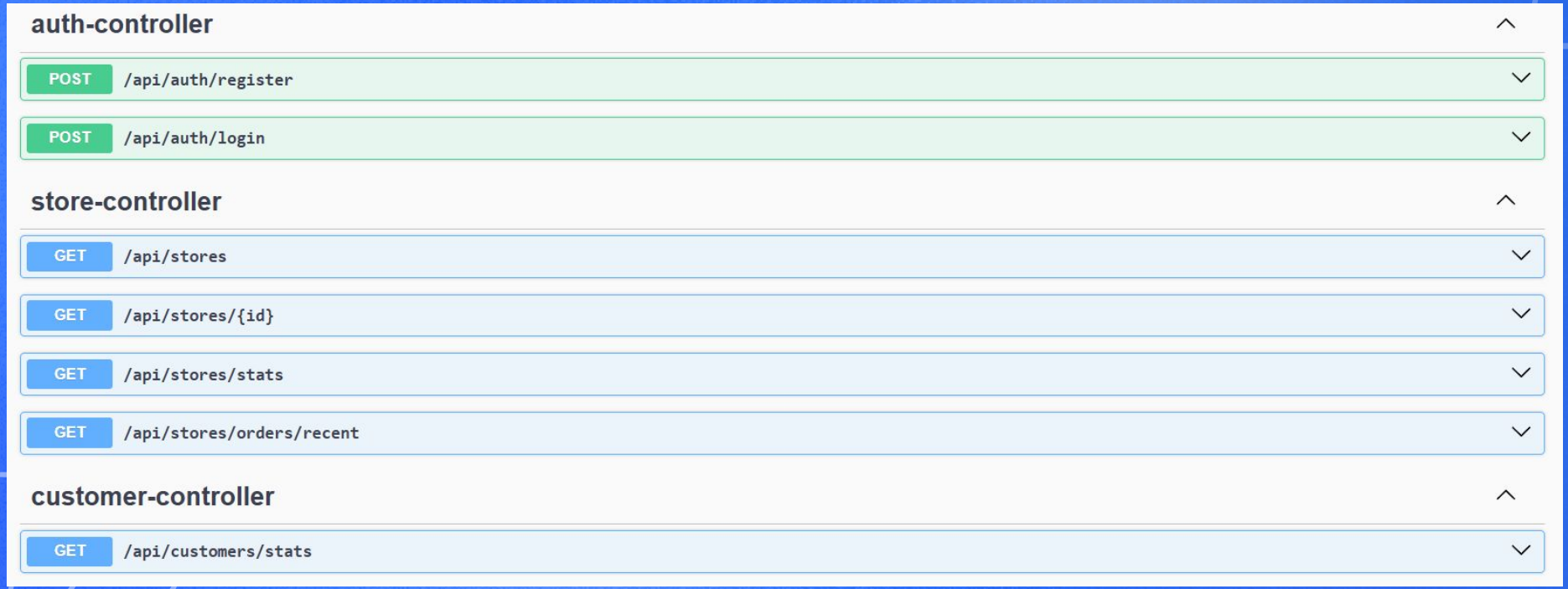
Models


User
PK id
String username
String email
String password
Role role
is a
Customer
PK id
String taxId
String firstName
String lastName
FK userId
places
Order
PK id
FK customerId
FK storeId
Float totalPrice
Date orderDate
OrderStatus status
receives
Store
PK id
String taxId
String name
String owner
FK userId
owns
contains
sells
ShoppingCart
PK id
FK customerId
Float totalPrice
contains
CartItem
PK id
FK cartId
FK productId
Int quantity
Float subtotal
OrderItem
PK id
FK orderId
FK productId
Int quantity
Float priceAtPurchace
Float subtotal
includes
Product
PK id
String type
String band
String description
Float price
Int stockQuantity
FK storeId
includes

System Architecture and Data Flow

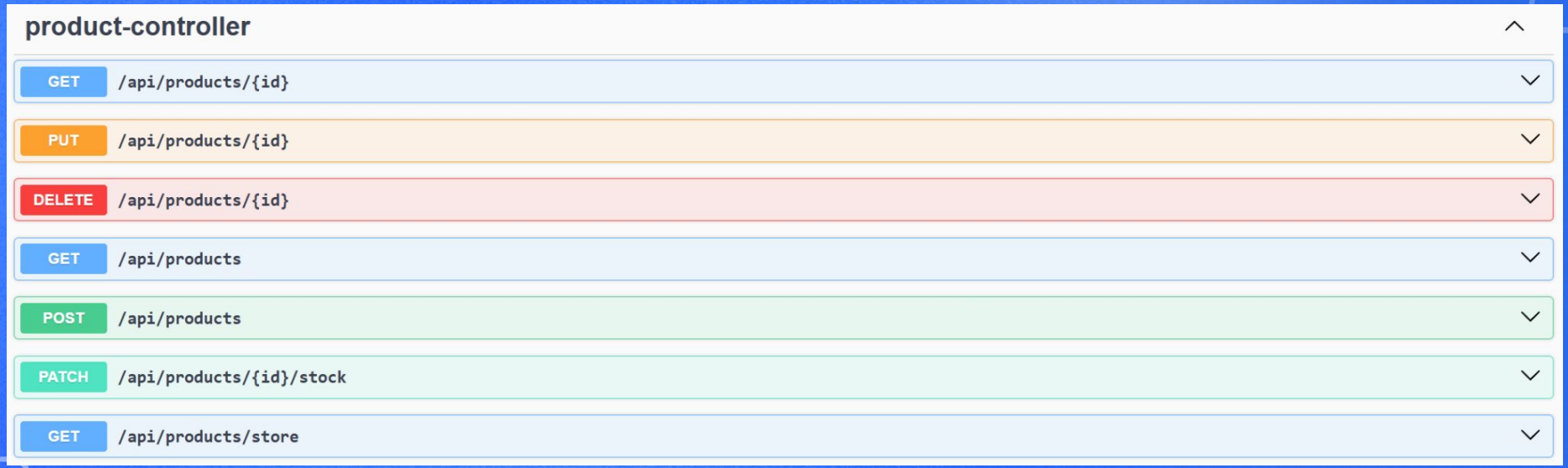


Endpoints



Swagger UI Docs: http://localhost:8080/swagger-ui/index.html

Endpoints



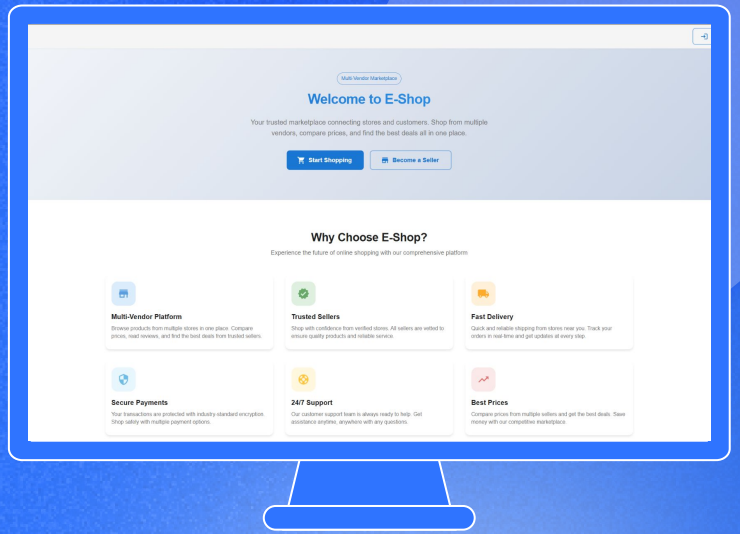
Swagger UI Docs: http://localhost:8080/swagger-ui/index.html

Endpoints

Swagger UI Docs: http://localhost:8080/swagger-ui/index.html

Live Demo

of the application



Key Workflows

Product Search

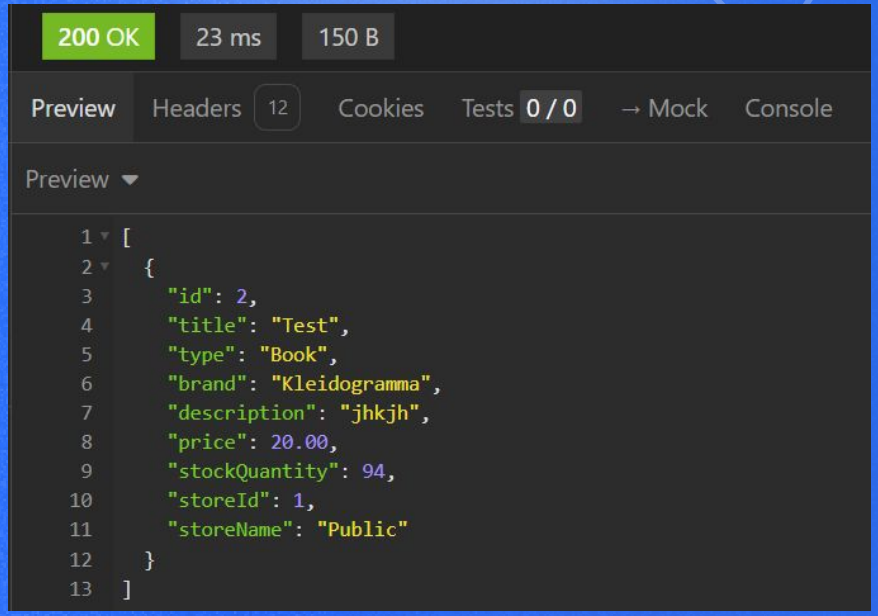
Product Search

Search Criteria:

- Title (`title`)
- Type (`type`)
- Brand (`brand`)
- Minimum Price (`minPrice`)
- Maximum Price (`maxPrice`)
- Store ID (`storeId`)

Endpoint:

GET api/products?<search parameters>

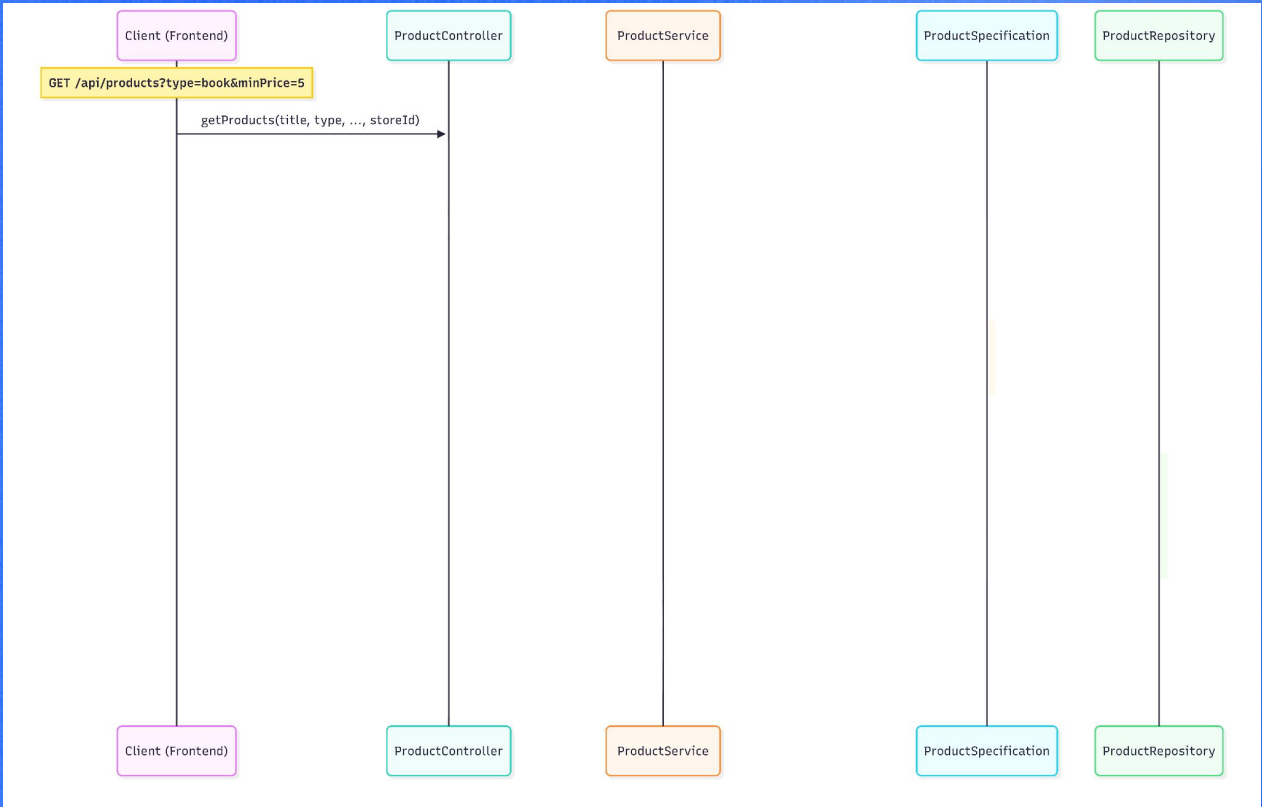


api/products?type=book&minPrice=5&maxPrice=20

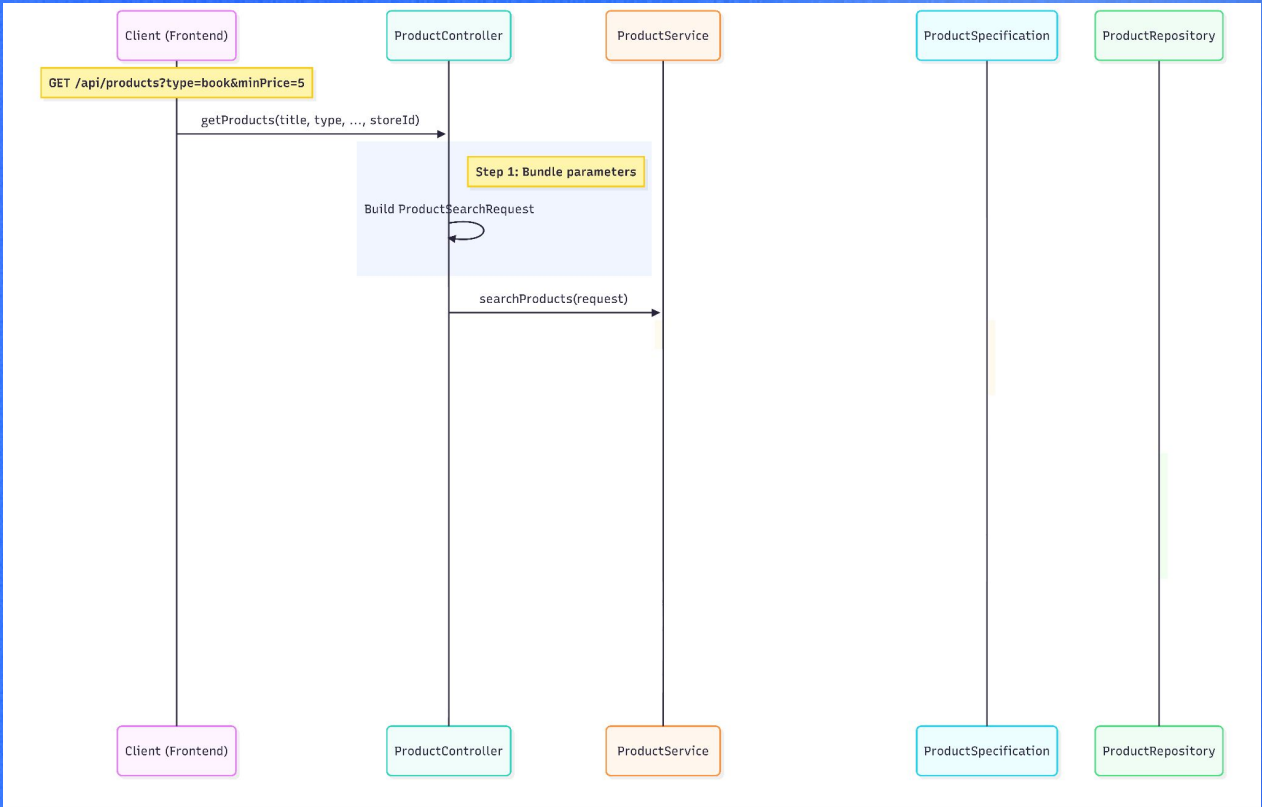
Product Search Implementation

- Uses Specifications and the Criteria API to build queries based on input fields
- Helps us avoid to write many `findBy...` methods on the ProductRepository
- Search logic is encapsulated within a dedicated **ProductSpecification** class, keeping the service and repository layer lean
- This allows for easy future modifications on search filters (e.g. filter by color)

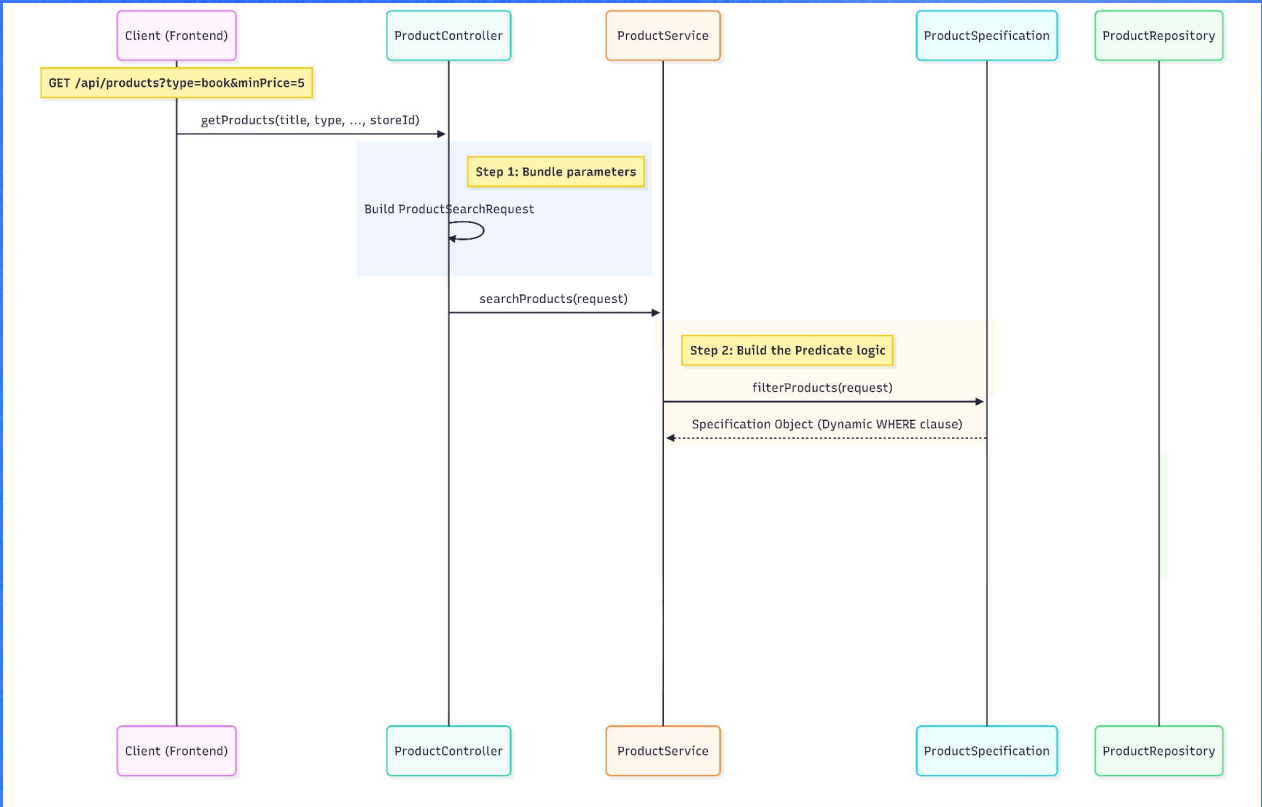
Product Search Request Life



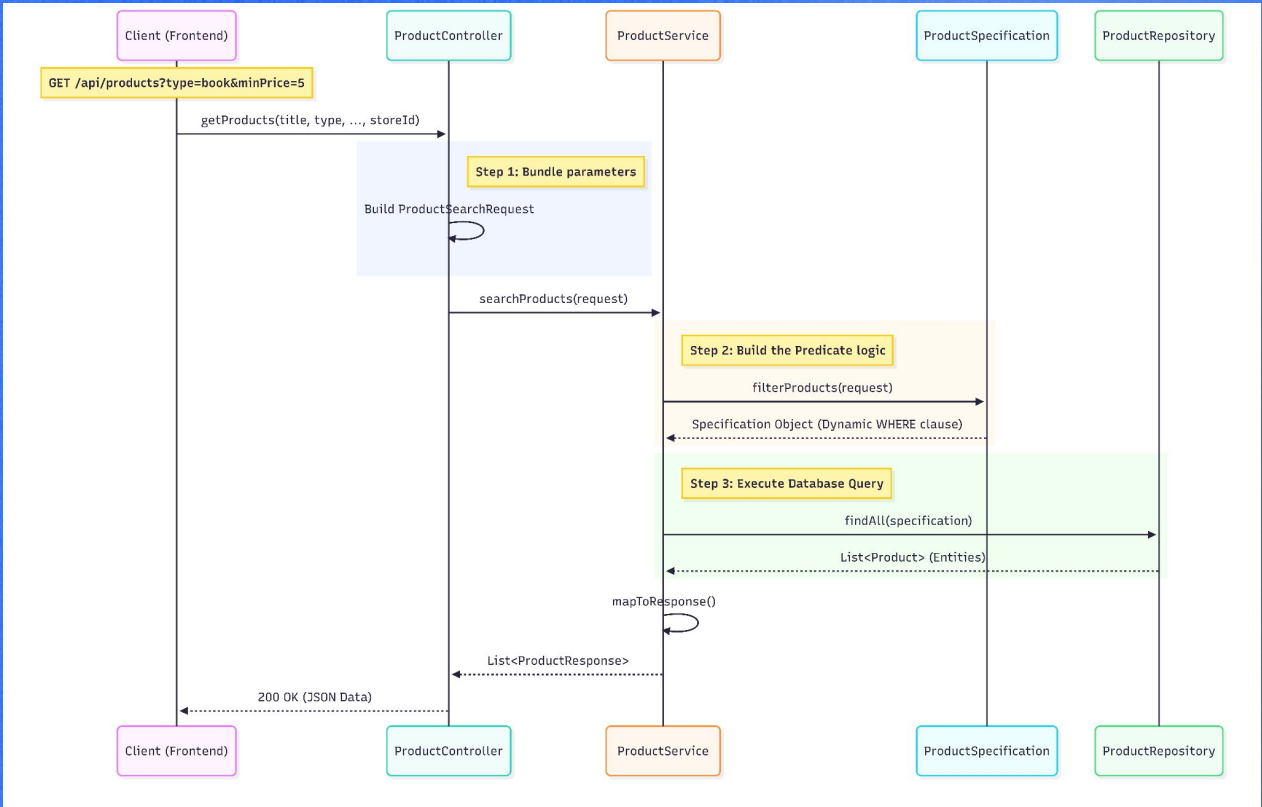
Product Search Request Life



Product Search Request Life



Product Search Request Life



ProductSpecification (Product Search Implementation)

```java
public class ProductSpecification {  2 usages   Archontis Kostis

    public static Specification<Product> filterProducts(ProductSearchRequest request) {  1 usage
        return Specification
                .where(titleContains(request.getTitle()))
                .and(typeContains(request.getType()))
                .and(brandContains(request.getBrand()))
                .and(minPrice(request.getMinPrice()))
                .and(maxPrice(request.getMaxPrice()))
                .and(storeIdEquals(request.getStoreId()));
    }

    private static Specification<Product> titleContains(String title) {...}

    private static Specification<Product> typeContains(String type) {...}

    private static Specification<Product> brandContains(String brand) {...}

    private static Specification<Product> minPrice(BigDecimal minPrice) {...}

    private static Specification<Product> maxPrice(BigDecimal maxPrice) {...}

    private static Specification<Product> storeIdEquals(Long storeId) {...}

    private static boolean isBlank(String value) { return value == null || value.isBlank(); }
}
```

ProductController (Product Search Implementation)

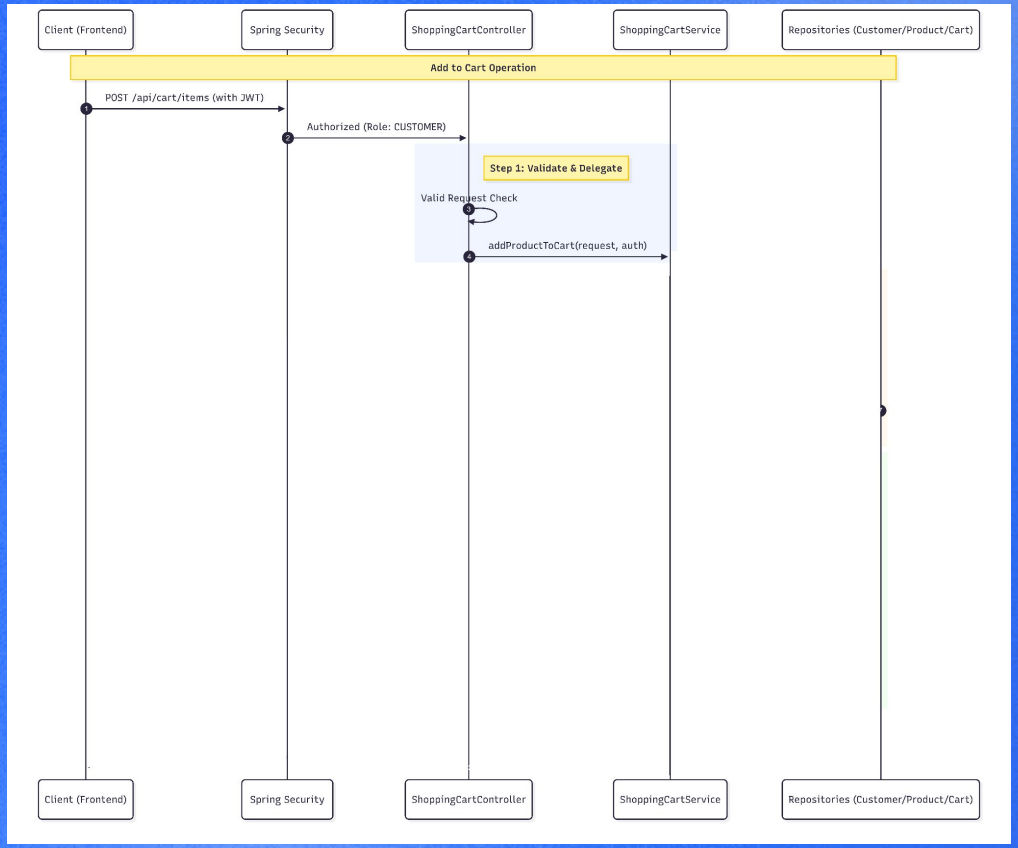
```java
@GetMapping  no usages  Archontis Kostis
public ResponseEntity<List<ProductResponse>> getProducts(
        @RequestParam(required = false) String title,
        @RequestParam(required = false) String type,
        @RequestParam(required = false) String brand,
        @RequestParam(required = false) java.math.BigDecimal minPrice,
        @RequestParam(required = false) java.math.BigDecimal maxPrice,
        @RequestParam(required = false) Long storeId) {

    ProductSearchRequest request = ProductSearchRequest.builder()
            .title(title)
            .type(type)
            .brand(brand)
            .minPrice(minPrice)
            .maxPrice(maxPrice)
            .storeId(storeId)
            .build();

    List<ProductResponse> products = productService.searchProducts(request);
    return ResponseEntity.ok(products);
}
```

ProductService (Product Search Implementation)

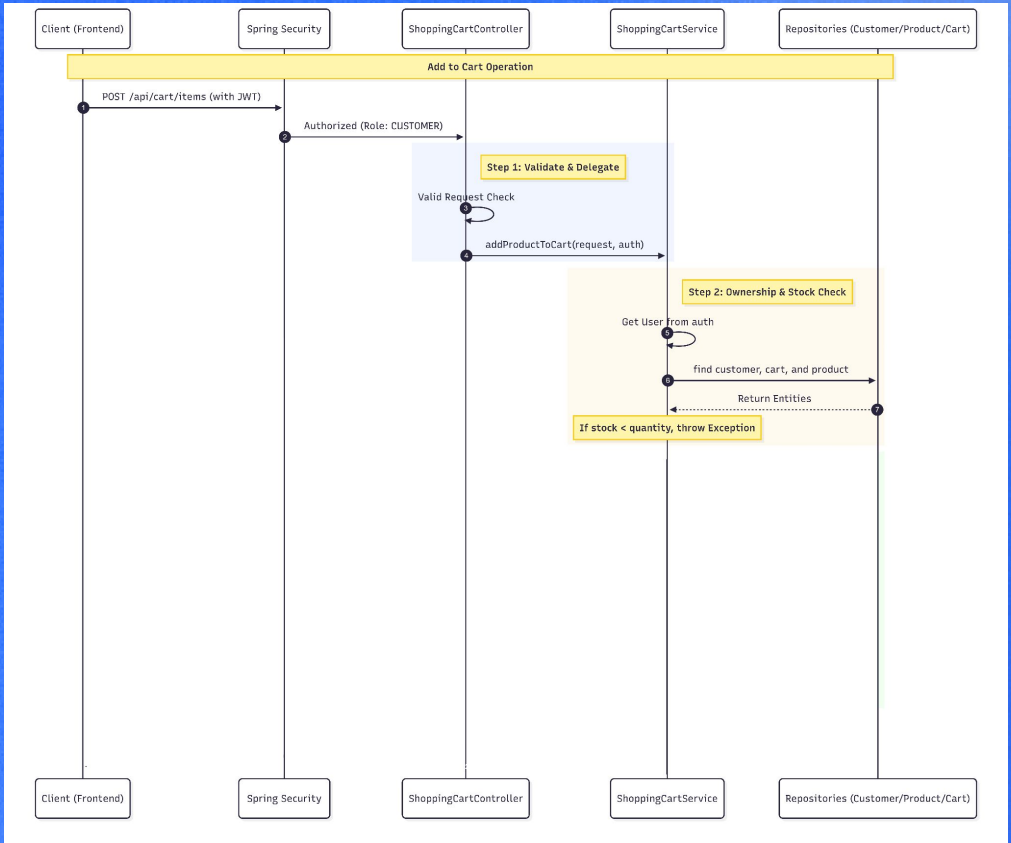
```java
@Transactional(readOnly = true)  1 usage  Archontis Kostis *
public List<ProductResponse> searchProducts(ProductSearchRequest request) {
    Specification<Product> specification = ProductSpecification.filterProducts(request);
    List<Product> products = productRepository.findAll(specification);

    return products.stream()  Stream<Product>
            .map(this::mapToResponse)  Stream<ProductResponse>
            .collect(Collectors.toList());
}
```

Add to Cart

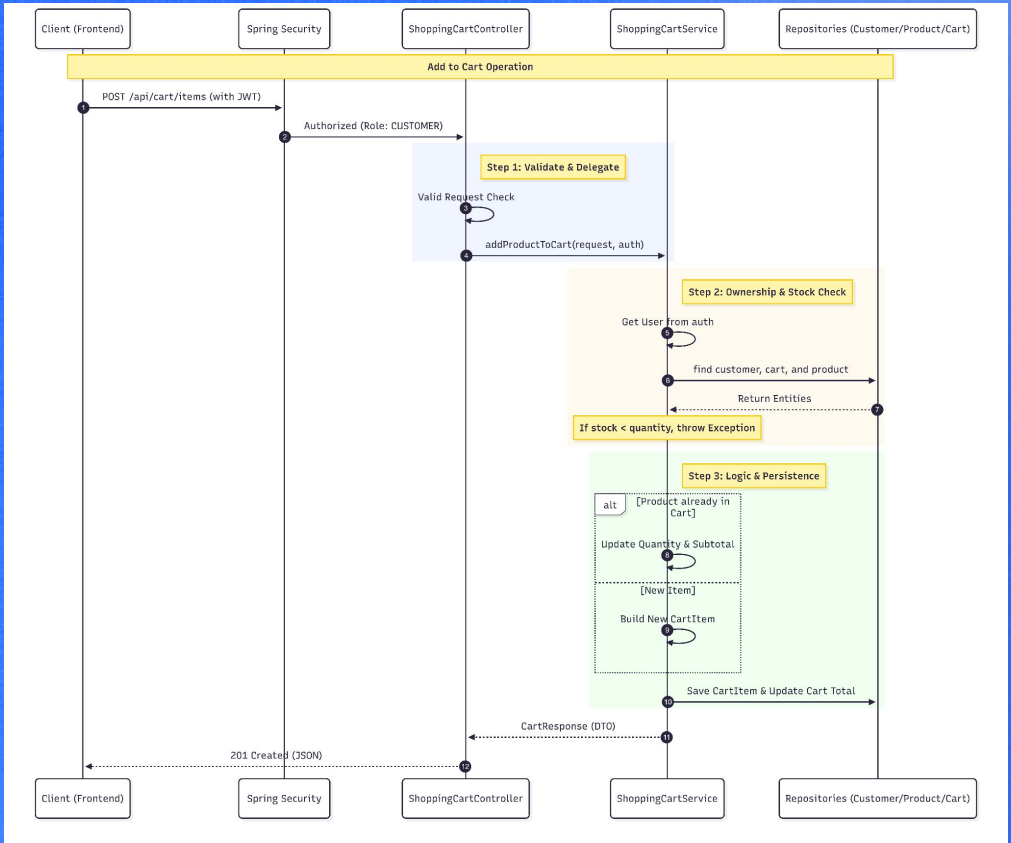
Add to Cart Operation



Add to Cart Operation



Add to Cart Operation

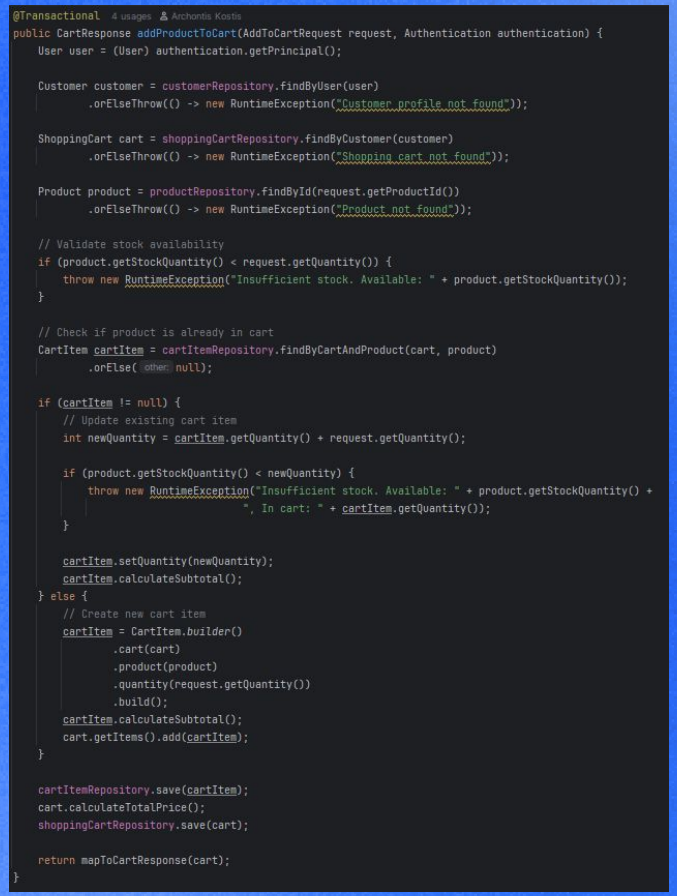


Code - AddToCartRequest & ShoppingCartController

```java
@Data  11 usages  Archontis Kostis
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class AddToCartRequest {

    @NotNull(message = "Product ID is required")
    private Long productId;

    @NotNull(message = "Quantity is required")
    @Min(value = 1, message = "Quantity must be at least 1")
    private Integer quantity;
}
```

```java
@PostMapping("/items")  no usages  Archontis Kostis *
public ResponseEntity<CartResponse> addToCart(
        @Valid @RequestBody AddToCartRequest request,
        Authentication authentication) {

    CartResponse response = shoppingCartService
            .addProductToCart(request, authentication);

    return ResponseEntity
            .status(HttpStatus.CREATED)
            .body(response);
}
```

Code - AddToCartRequest & ShoppingCartController



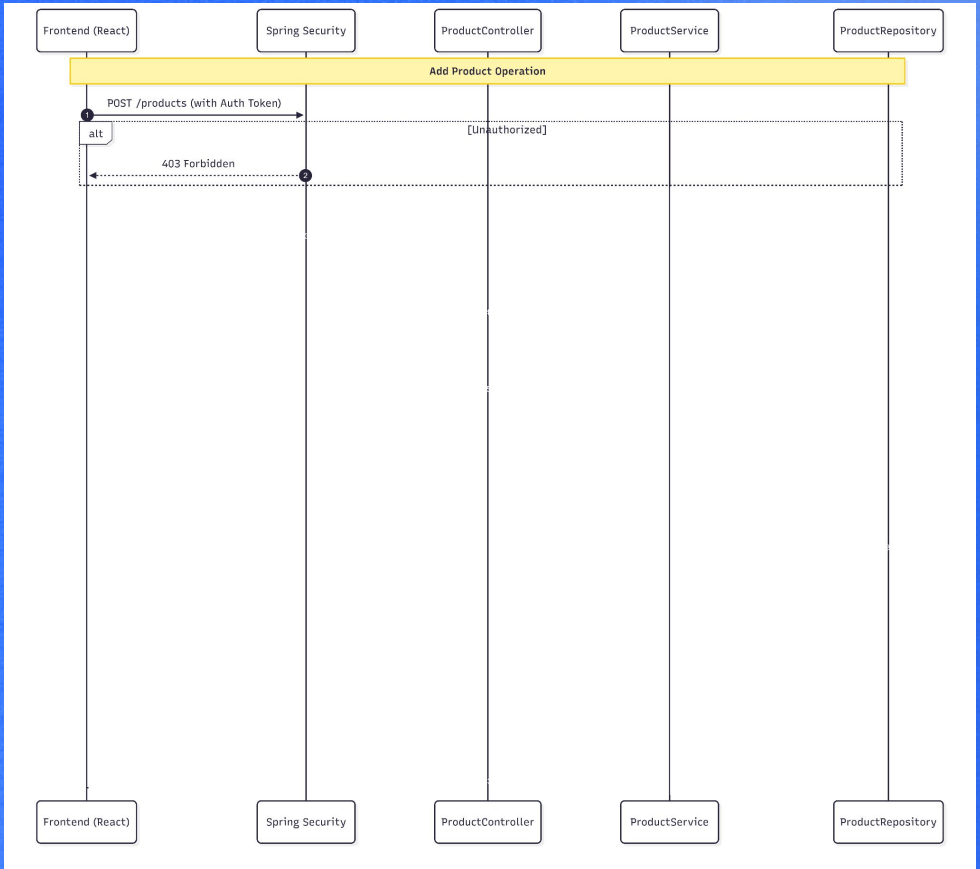
```java
@Transactional  4 usages  Archontis Kostis
public CartResponse addProductToCart(AddToCartRequest request, Authentication authentication) {
    User user = (User) authentication.getPrincipal();

    Customer customer = customerRepository.findByUser(user)
            .orElseThrow(() -> new RuntimeException("Customer profile not found"));

    ShoppingCart cart = shoppingCartRepository.findByCustomer(customer)
            .orElseThrow(() -> new RuntimeException("Shopping cart not found"));

    Product product = productRepository.findById(request.getProductId())
            .orElseThrow(() -> new RuntimeException("Product not found"));

    // Validate stock availability
    if (product.getStockQuantity() < request.getQuantity()) {
        throw new RuntimeException("Insufficient stock. Available: " + product.getStockQuantity());
    }

    // Check if product is already in cart
    CartItem cartItem = cartItemRepository.findByCartAndProduct(cart, product)
            .orElse( other: null);

    if (cartItem != null) {
        // Update existing cart item
        int newQuantity = cartItem.getQuantity() + request.getQuantity();

        if (product.getStockQuantity() < newQuantity) {
            throw new RuntimeException("Insufficient stock. Available: " + product.getStockQuantity() +
                                       ", In cart: " + cartItem.getQuantity());
        }

        cartItem.setQuantity(newQuantity);
        cartItem.calculateSubtotal();
    } else {
        // Create new cart item
        cartItem = CartItem.builder()
                .cart(cart)
                .product(product)
                .quantity(request.getQuantity())
                .build();
        cartItem.calculateSubtotal();
        cart.getItems().add(cartItem);
    }

    cartItemRepository.save(cartItem);
    cart.calculateTotalPrice();
    shoppingCartRepository.save(cart);

    return mapToCartResponse(cart);
}
```

Thanks!

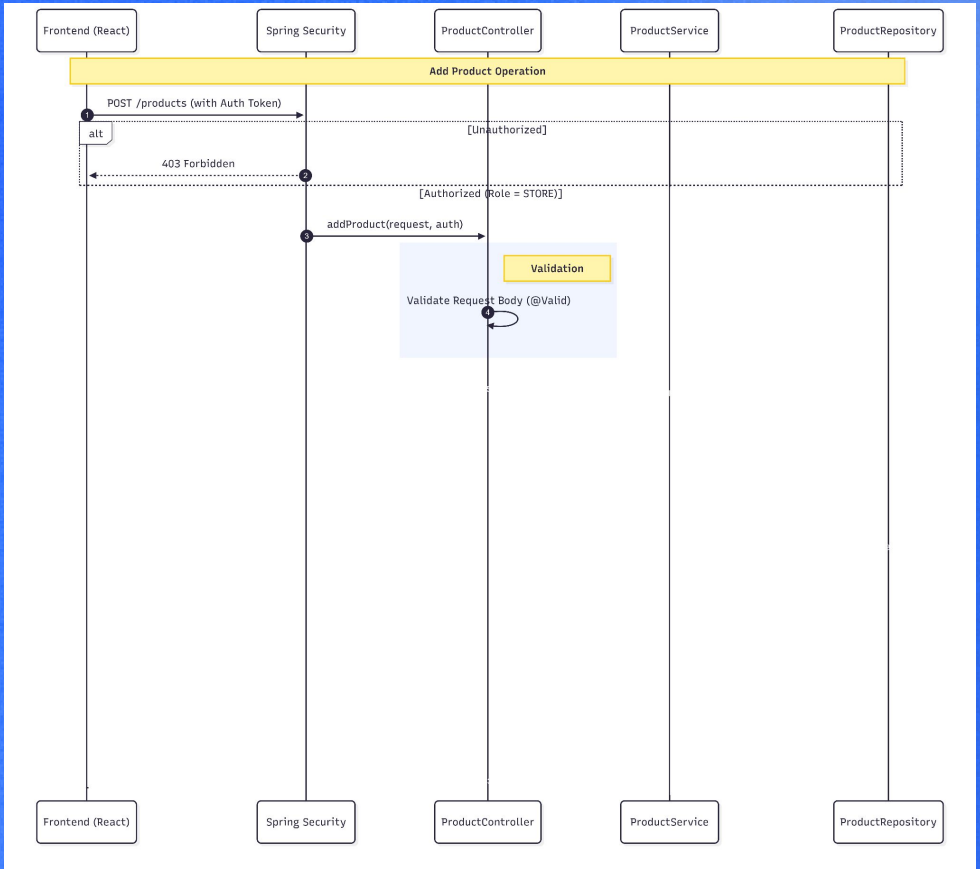
Questions?

extras

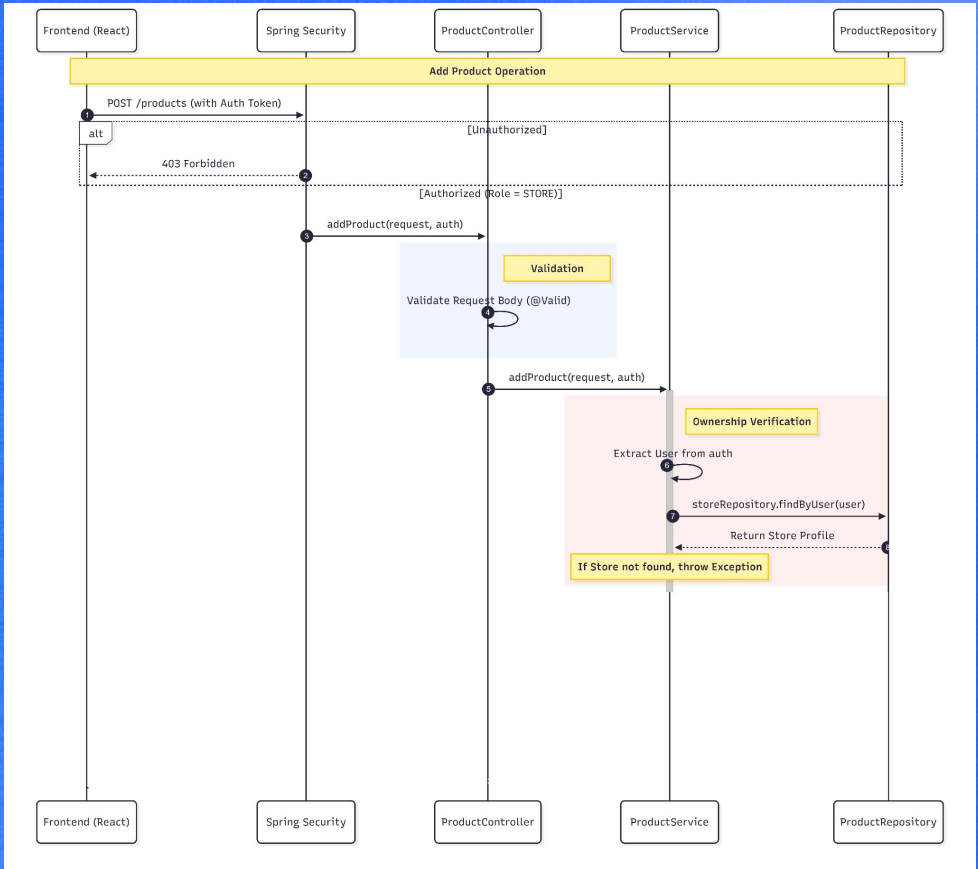
Add Product Operation



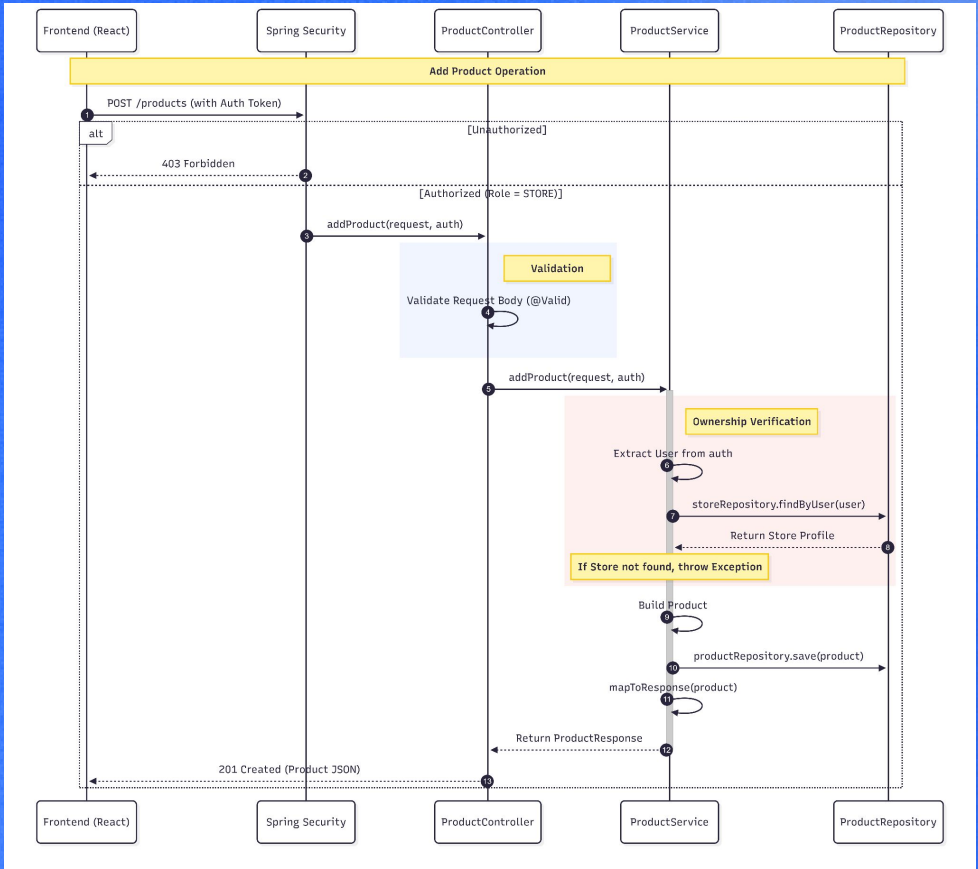
Add Product Operation



Add Product Operation



Add Product Operation



Code - AddProductRequest

```java
@Data  9 usages  Archontis Kostis
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class AddProductRequest {

    @NotBlank(message = "Title is required")
    @Size(max = 200, message = "Title must not exceed 200 characters")
    private String title;

    @NotBlank(message = "Product type is required")
    @Size(max = 100, message = "Type must not exceed 100 characters")
    private String type;

    @NotBlank(message = "Brand is required")
    @Size(max = 100, message = "Brand must not exceed 100 characters")
    private String brand;

    @NotBlank(message = "Description is required")
    @Size(max = 500, message = "Description must not exceed 500 characters")
    private String description;

    @NotNull(message = "Price is required")
    @DecimalMin(value = "0.01", message = "Price must be greater than 0")
    @Digits(integer = 10, fraction = 2, message = "Price must have at most 10 integer digits and 2 decimal places")
    private BigDecimal price;

    @NotNull(message = "Stock quantity is required")
    @Min(value = 0, message = "Stock quantity cannot be negative")
    private Integer stockQuantity;
}
```

Code - ProductController

```
@PostMapping  no usages  Archontis Kostis
@PreAuthorize("hasRole('STORE')")
public ResponseEntity<ProductResponse> addProduct(
        @Valid @RequestBody AddProductRequest request,
        Authentication authentication) {
    ProductResponse response = productService.addProduct(request, authentication);
    return ResponseEntity.status(HttpStatus.CREATED).body(response);
}
```

Code - ProductService

```java
@Transactional  3 usages  Archontis Kostis
public ProductResponse addProduct(AddProductRequest request, Authentication authentication) {
    User user = (User) authentication.getPrincipal();

    Store store = storeRepository.findByUser(user)
            .orElseThrow(() -> new RuntimeException("Store profile not found for user"));

    Product product = Product.builder()
            .title(request.getTitle())
            .type(request.getType())
            .brand(request.getBrand())
            .description(request.getDescription())
            .price(request.getPrice())
            .stockQuantity(request.getStockQuantity())
            .store(store)
            .build();

    product = productRepository.save(product);

    return mapToResponse(product);
}
```


slidesgo