What is SpringShop?

SpringShop is an online store which offers the purchase of electronic devices. By accessing the website, the user has options of registering or logging into an existing account. After that, the user selects a product from the existing list and proceeds to finalize the order.

Applied technologies

- Java (version 11)

- Maven

- Spring Boot (version 2.6.6)

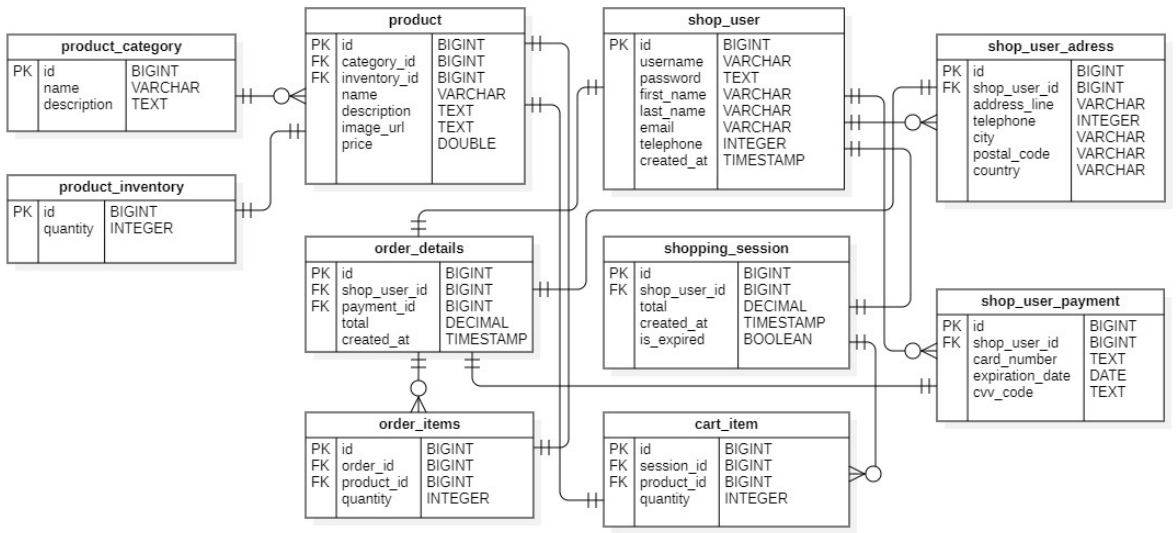
- Lombok – library for automation of creating classes

- Spring Security – a tool to ensure authentication, authorization, other security features

- PostgreSQL

- Spring Data JPA – a tool for using the database

Database UML diagram



Email notifications

JavaMail API was used to send messages to customers. It is a library that provides solutions for sending emails.

First add dependencies to the project:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-mail</artifactId>
</dependency>
```

Then in the applications.yaml file we need to add the configuration:

```
spring:
  mail:
    host: smtp.gmail.com
    port: 587
    username: spr.shop22@gmail.com
    password: qksgtegnzfqzlrnf
    properties:
      mail:
        smtp:
          auth: true
          starttls:
            enable: true
```

Implementation:

```
@Service
@AllArgsConstructor
public class EmailSenderService {
    private JavaMailSender mailSender;

    @Async
    public void sendEmail(String toEmail, String subject, String body) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setFrom("spr.shop22@gmail.com");
        message.setTo(toEmail);
        message.setText(body);
        message.setSubject(subject);
        mailSender.send(message);
    }
}
```

Endpoints

For each of the classes in the project, we created methods which are endpoints. They can be used to display objects of a class, add, or remove them.

1. **ShopUser controller**

```java
@RestController
@RequestMapping(path = "api/v1/shop-users")
@AllArgsConstructor
public class ShopUserController {
    private final ShopUserService shopUserService;
    private final ShopUserAddressService shopUserAddressService;
    private final ShopUserPaymentService shopUserPaymentService;

    @GetMapping
    @PreAuthorize("hasAuthority('ADMIN')")
    public ResponseEntity<Object> getUsers() {
        return ResponseHandler
                .generateResponse( message: "Sukces", HttpStatus.OK, shopUserService.getUsers());
    }

    @GetMapping(path = "/account-information")
    public ResponseEntity<Object> getShopUserInformation(Authentication authentication) {
        return ResponseHandler
                .generateResponse( message: "Sukces", HttpStatus.OK,
                        shopUserService.getShopUserInformation(authentication));
    }

    @GetMapping(path = "/addresses")
    public ResponseEntity<Object> getShopUserAddresses(Authentication authentication) {
        return ResponseHandler
                .generateResponse( message: "Sukces", HttpStatus.OK,
                        shopUserAddressService.getShopUserAddresses(authentication));
    }

    @GetMapping(path = "/payments")
    public ResponseEntity<Object> getShopUserPayments(Authentication authentication) {
        return ResponseHandler
                .generateResponse( message: "Sukces", HttpStatus.OK,
                        shopUserPaymentService.getShopUserPayments(authentication));
    }

    @PostMapping(path = "/addresses/save")
    public ResponseEntity<Object> saveNewShopUserAddress(@RequestBody ShopUserAddressRequest shopUserAddressRequest,
                                                         Authentication authentication) {
        if (shopUserAddressRequest.getAddressLine() == null) {...}
        if (shopUserAddressRequest.getTelephone() == null) {...}
        if (shopUserAddressRequest.getCity() == null) {...}
        if (shopUserAddressRequest.getPostalCode() == null) {...}
        if (shopUserAddressRequest.getCountry() == null) {...}
        shopUserService.saveNewShopUserAddress(shopUserAddressRequest, authentication);
        return ResponseHandler
                .generateResponseWithoutData( message: "Dodano nowy adres", HttpStatus.CREATED);
    }

    @PostMapping(path = "/payments/save")
    public ResponseEntity<Object> saveNewShopUserPayment(@RequestBody ShopUserPaymentRequest shopUserPaymentRequest,
                                                         Authentication authentication) {
        if (shopUserPaymentRequest.getCardNumber() == null) {...}
        if (shopUserPaymentRequest.getExpirationDate() == null) {...}
        if (shopUserPaymentRequest.getCvvCode() == null) {...}
        shopUserService.saveNewShopUserPayment(shopUserPaymentRequest, authentication);
        return ResponseHandler
                .generateResponseWithoutData( message: "Dodano nową metodę płatności", HttpStatus.CREATED);
    }
}
```

- *getUsers* – returns a response which contains a list of shop users
- *getShopUserInformation* – returns a response which contains informations about currently logged in user
- *getShopUserAddresses* – returns a response which contains a list of addresses of currently logged in user
- *getShopUserPayments* – returns a response which contains a list of payments of currently logged in user
- *saveNewShopUserAddress* – adds a new address specified in request body to list of address of currently logged in user; if one of the fields of request body is null it will return response with „400 Bad Request" status
- *saveNewShopUserPayment* – adds a new payment specified in request body to list of payments of currently logged in user; if one of the field of request body is null it will return response with „400 Bad Request" status

2. Product controller

```java
@RestController
@RequestMapping(path = "api/v1/products")
@AllArgsConstructor
public class ProductController {

    private final ProductService productService;

    @GetMapping
    public ResponseEntity<Object> getAllProducts() {
        List<Product> products = productService.getAllProducts();
        return ResponseHandler
                .generateResponse( message: "Sukces", HttpStatus.OK, products);
    }

    @GetMapping(path = "{productId}")
    public ResponseEntity<Object> getProductById(@PathVariable("productId") Long productId) {
        Product product = productService.getProductById(productId);
        return ResponseHandler
                .generateResponse( message: "Sukces", HttpStatus.OK, product);
    }
}
```

- *getAllProducts* - returns a response which contains list of all products
- *getProductById* - returns a response which contains product whose identifier is secified in the parameter

3. ShoppingSession controller

```java
@RestController
@RequestMapping(path = "api/v1/shopping-session")
@AllArgsConstructor
public class ShoppingSessionController {
    private final ShoppingSessionService shoppingSessionService;
    private final CartItemService cartItemService;

    @GetMapping("cart-items")
    public ResponseEntity<Object> getCartItemsByShoppingSessionId(Authentication authentication) {
        List<CartItem> cartItems = cartItemService.getCartItemsByShoppingSessionId(authentication);

        ShoppingSession shoppingSession = shoppingSessionService.getCurrentlyActiveShoppingSession(authentication);
        Map<String, Object> data = new HashMap<>();
        data.put("total", shoppingSession.getTotal());
        List<Map<String, Object>> items = new ArrayList<>();
        for (CartItem cartItem : cartItems) {...}
        data.put("cart_items", items);
        return ResponseHandler.generateResponse( message: "Sukces", HttpStatus.OK, data);
    }

    @PostMapping("cart-items/save")
    public ResponseEntity<Object> addCartItemToCurrentShoppingSession(@RequestBody CartItemRequest cartItemRequest,
                                                                      Authentication authentication) {
        if (cartItemRequest.getQuantity() == null) {...}
        if (cartItemRequest.getProductId() == null) {...}
        if (cartItemRequest.getQuantity() < 1) {...} else {...}
    }


    @PutMapping("cart-items/change-quantity/{cartItemId}/{quantity}")
    public ResponseEntity<Object> changeCartItemQuantity(@PathVariable("cartItemId") Long cartItemId,
                                                         @PathVariable("quantity") Integer quantity,
                                                         Authentication authentication) {
        int quantityBeforeChange = cartItemService.getById(cartItemId).getQuantity();
        if (quantity < 1) {...} else {...}
    }

    @DeleteMapping("cart-items/delete/{cartItemId}")
    public ResponseEntity<Object> deleteCartItemById(@PathVariable("cartItemId") Long cartItemId,
                                                     Authentication authentication) {
        cartItemService.deleteCartItemById(cartItemId, authentication);
        return ResponseHandler
                .generateResponseWithoutData( message: "Usunięto produkt z koszyka o id " +
                        cartItemId, HttpStatus.OK);
    }
}
```

- *getCartItemsByShoppingSessionId* – returns a response which contains a list of cart items that are in current shopping session
- *addCartItemToCurrentShoppingSession* – adds new cart item which is specified in request body to list of cart items; if one of request fields is null or if quantity is less than 1 it will return response with „400 Bad Request” status
- *changeCartItemQuantity* – changes quantity of item whose identifier and new quantity are specified in path variable; if new quantity is less than 1 it will return response with „400 Bad Request” status
- *deleteCartItemById* – removes the cart item which identifier is specified in path variable

4. OrderDetails controller

```java
@RestController
@RequestMapping(path = "api/v1/orders")
@AllArgsConstructor
public class OrderDetailsController {
    private final OrderDetailsService orderDetailsService;

    @PostMapping("make-order")
    private ResponseEntity<Object> makeOrder(@RequestBody OrderDetailsRequest orderDetailsRequest, Authentication authentication) {
        if (orderDetailsRequest.getShopUserPaymentId() == null) {
            return ResponseHandler
                    .generateResponseWithoutData( message: "Nie podano id metody płatności", HttpStatus.BAD_REQUEST);
        }
        if (orderDetailsRequest.getShopUserAddressId() == null) {
            return ResponseHandler
                    .generateResponseWithoutData( message: "Nie podano id adresu", HttpStatus.BAD_REQUEST);
        }
        orderDetailsService.makeOrder(orderDetailsRequest, authentication);
        return ResponseHandler
                .generateResponseWithoutData( message: "Sukces", HttpStatus.CREATED);
    }
}
```

- *makeOrder* – makes an order with products that are in the current shopping session; it requires a request body with payment identifier and user address identifier, if one of them is null, it will response with „400 Bad Request” status

5. Registration controller

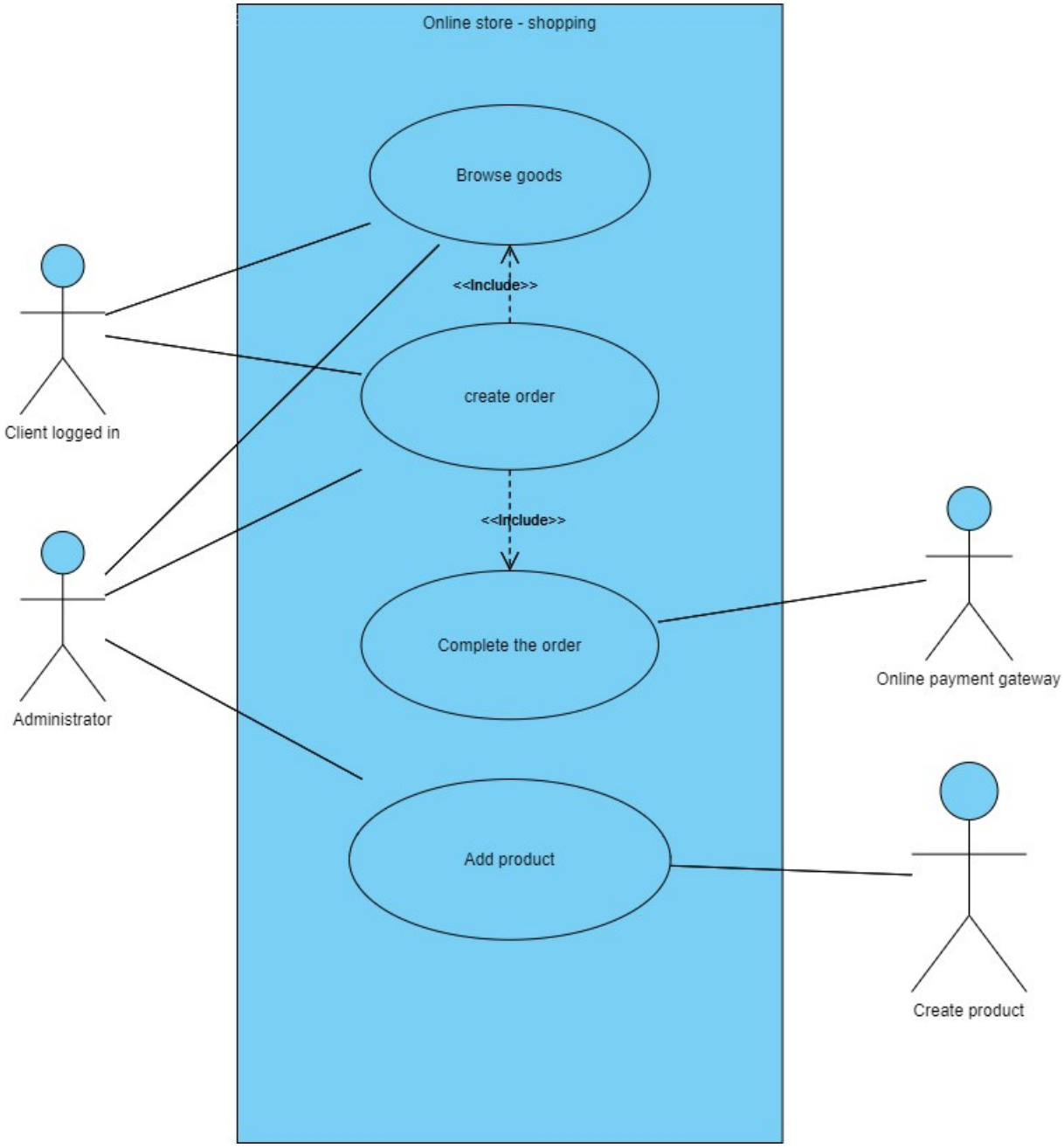
```java
@RestController
@RequestMapping(path = "/registration")
@AllArgsConstructor
public class RegistrationController {

    private final RegistrationService registrationService;

    @PostMapping
    public ResponseEntity<Object> register(@RequestBody RegistrationRequest request) {
        if (request.getFirstName() == null) {...}
        if (request.getLastName() == null) {...}
        if (request.getEmail() == null) {...}
        if (request.getPassword() == null) {...}
        if (request.getTelephone() == null) {...}
        registrationService.register(request);
        return ResponseHandler
                .generateResponseWithoutData( message: "Pomyślnie zarejestrowano użytkownika", HttpStatus.CREATED);
    }

    @GetMapping(path = "confirm")
    public ResponseEntity<Void> confirm(@RequestParam("token") String token) {
        registrationService.confirmToken(token);
        return ResponseEntity.status(HttpStatus.FOUND).location(URI.create("http://localhost:8081")).build();
    }
}
```

- *register* – creates a new token for the user given in request body, and sends an activation link to his email address; if one of request body fields is null it will return request with „400 Bad Request” status

- *confirm* – confirms the token specified in the parameter and adds the user to the database

Use cases

1) shopping



2) register/login