Akdeniz University

Department of Computer Engineering

Senior Design Project

Final Report
Project: E-commerce

Member Name :Tunahan Burak Dirlik

Supervisor: Hüseyin Gökhan Akçay

CONTENT

1. Introduction

This Project is an e-commerce project. There are 2 types of users in the application. Admin and customer. Admin user can upload products to the application. It can update or delete the added product. Admin can access the details of the order of the customer. The shopping made by the customer is recorded in the database and the admin user can see it.

Customer can create an account and log in to the app with that account. If the email selected to create an account has been taken by someone else, this email cannot be used. Multiple accounts cannot be created with the same email. When the user logs into the account, they can see their own information, that is, the session information that ı have implemented in the application. The user can add the product to the cart, delete it from the cart, and see all the products added to the cart. Customer can order the product by entering the address information.

All operations that can be performed in the application are shown below.

- Sig up
- Sign in
- Get session info
- Add product
- Get product (view product)
- Get all products (shopping page, products)
- Update product
- Delete product
- Add product to cart
- Get all products from cart
- Delete product from cart
- Save order

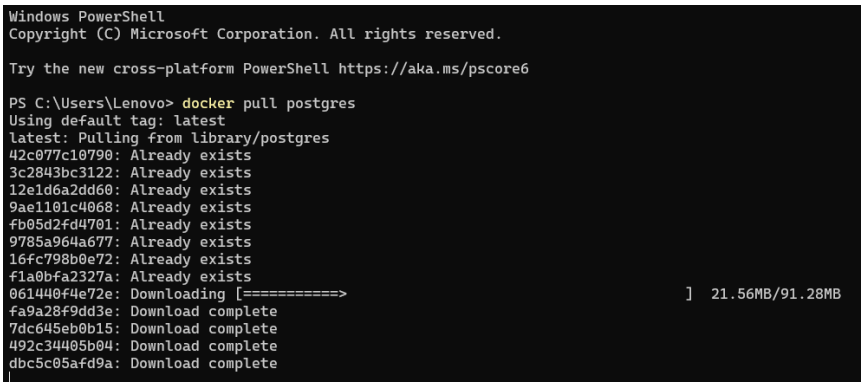
2. Requirements Details

Hardware: Computer
Software:

- For executing application: Docker desktop, docker postgres image, any Java editor.
- For testing application: Postman api, for backend testing, any browser for frontend testing.

2.1 Docker installation and Postgres image setup

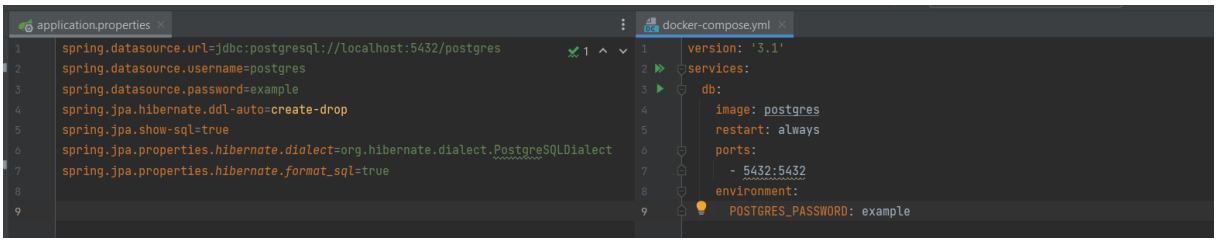
1) we need to install docker desktop. We can install from https://www.docker.com/products/docker-desktop/ We should choose appropriate operating system and install it.
2) We need to setup postgres image from terminal. Execute this command > docker pull postgres

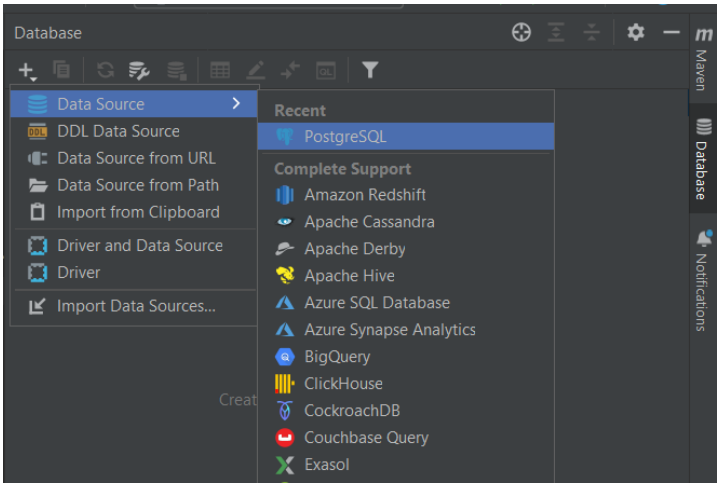


2.2 Database Preparation

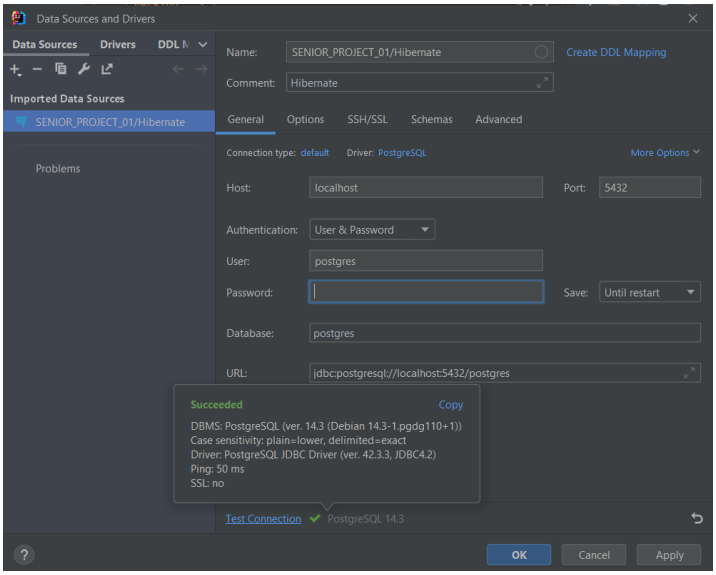
Database settings are present in the application.properties and docker-compose.yml files

- application.properties path: src/main/resources/application.properties
- docker-compose.yml path: src/main/resources/docker-compose.yml





For creating and making connection of database, we should choose PostgreSql in the Database section from Intelij ide.

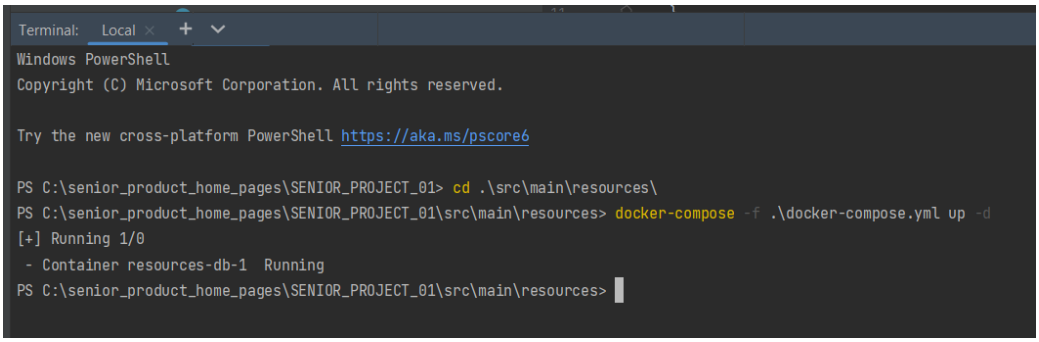


User : postgres
Password : example

When press the Apply button we should see Succeeded message like below

Finally we created database and we will run database as a docker container. For that we need to go ide terminal.

1) Change directory to → src\main\resource from ide terminal. This directory contain docker-compose.yml file, we will use this file with below command.
2) Execute this command → docker-compose -f .\docker-compose.yml up –d
   These steps shown below picture.



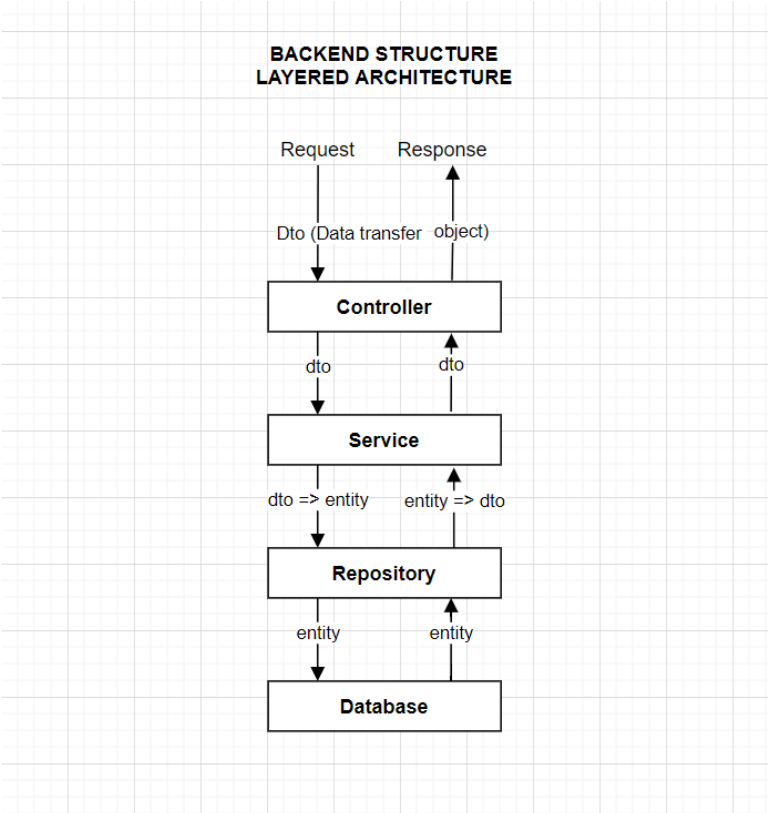
Spring application is ready. We can execute from run button now.

3. Final Architecture and Design Details

Project contains these essential packages:

- Dto
- Enttiy
- Service
- Controller
- Repository

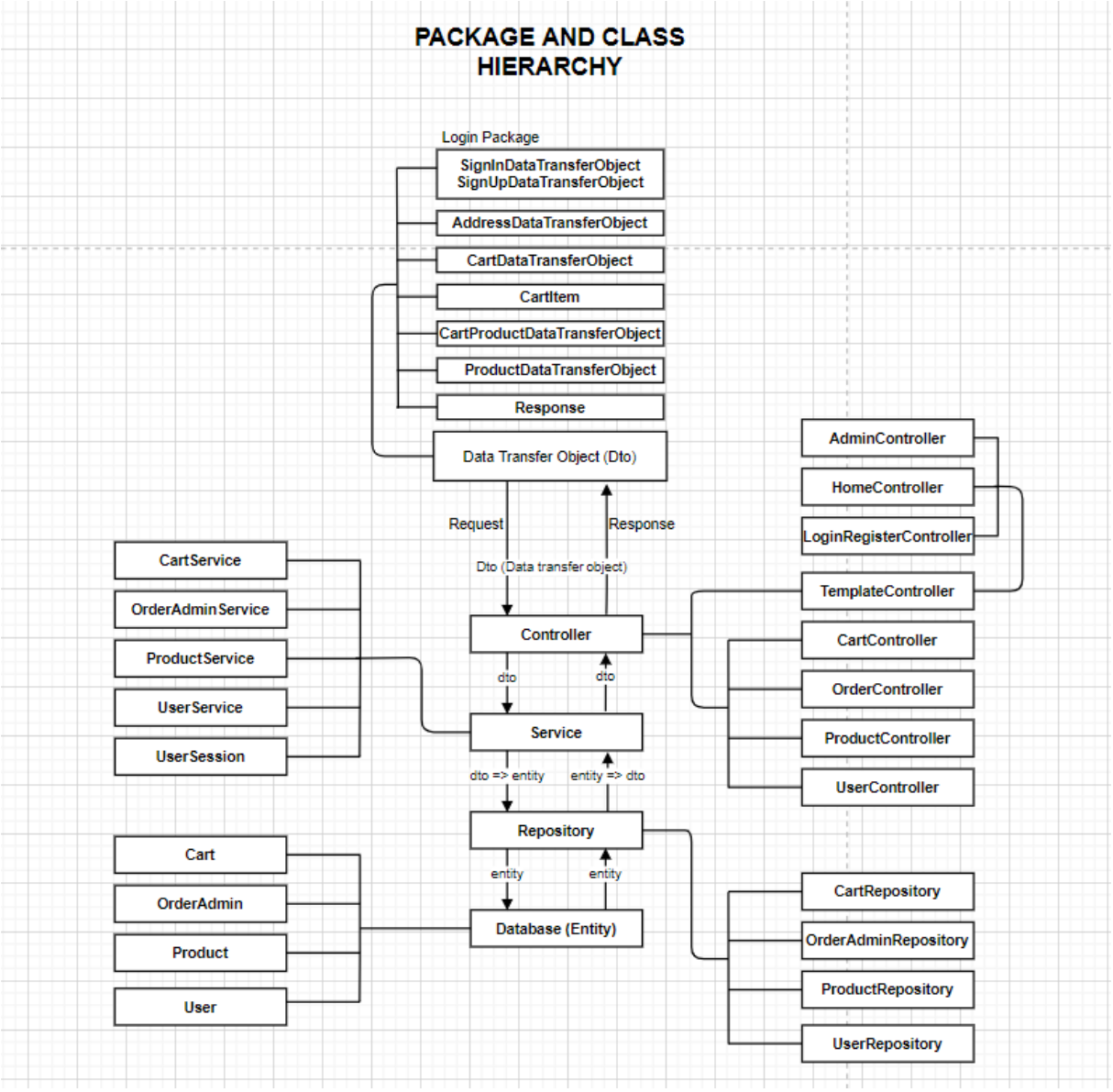
Architectural Design



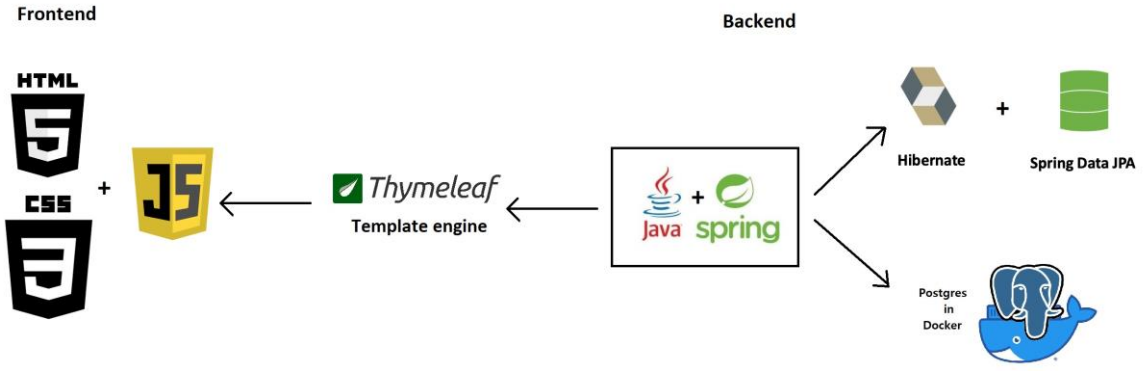
Layered architecture is a architectural pattern, which is preferred especially in Web Servers. It is the division of business logics layer by layer and providing access to each other through these layers. The data coming from the request is mapped to data transfer objects (dto). Controller layer takes this dto and direct to service layer.

Service layer convert dto to entity. Then make needed operations and service layer give these entities to repository for crud operations. Repository interfaces are extends from JpaRepository and has own database method which are: save, remove, findById() and etc. corresponding to crud operations. Crud operations are create, read update, delete. Response, on the other hand, reaches to client by passing the layers in the opposite direction.

Hierarchy of classes and packages

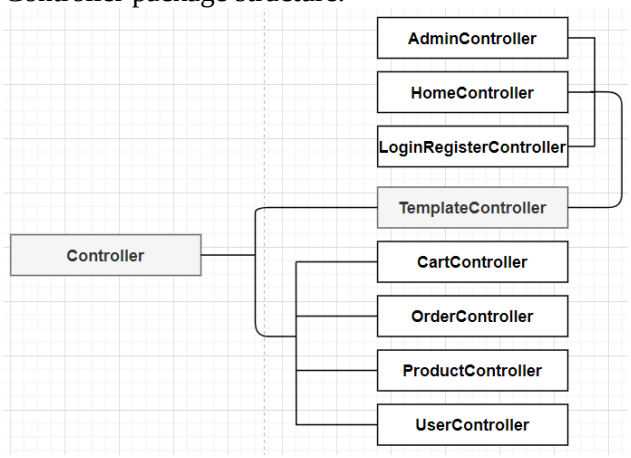


Project structure details & used technologies



4. Development/Implementation Details

Controller package structure.



Controller contain sub package known as TemplateController. TemplateController package contain controller classes that manage frontend.

These are:

- AdminController
- HomeController
- LoginRegisterController

Another classes can be connectable to any frontend framework. Also we can directly test controllers from Postman api.

Lets examine the AdminController class.

```
package com.project.senior_project_01.Controller.TemplateController;
import ...

@Controller
public class AdminController {

    @Autowired
    UserService userService;
    @Autowired
    UserRepository userRepository;
    @GetMapping("/admin")
    public String adminHome() { return "adminHome"; }
```

All controller classes should contain **@Controller** annotation to catch request and responses. **@Autowired** is injection in the Spring framework. We are getting another classes with this.

At the first part of AdminController class we added that we will use classes UserService and UserRepository. adminHome() method is getting adminHome.html file. Return object “adminHome” specify the html file.

```java
//-----------------------------------------------------------product-----------------------------------------------
    @Autowired
    ProductService productService;
    @GetMapping("/admin/products")
    public String getProduct(Model model){
        model.addAttribute( attributeName: "products", productService.getAllProductFromPage());
        return "products";
    }
    @GetMapping("/admin/products/add")
    public String getProductAdd(Model model){
        model.addAttribute( attributeName: "product", new Product());
        return "productsAdd";
    }
    @PostMapping("/admin/products/add")
    public String postProductAdd(@ModelAttribute("product") Product product){

        productService.addProductFromPage(product);
        return "redirect:/admin/products";
    }
    @GetMapping("/admin/products/delete/{id}")
    public String deleteProduct(@PathVariable int id){

        productService.removeProductByIdFromPage(id);
        return "redirect:/admin/products";
    }
    @GetMapping("/admin/products/update/{id}")
    public String updateProduct(@PathVariable int id, Model model){
        Optional<Product> product = productService.getProductByIdFromPage(id);

        if (product.isPresent()){
            model.addAttribute( attributeName: "product", product.get());
            return "productsAdd";
        }
        else
            return "404";
    }
```

At the second part of AdminController class we have another endpoints that manage the framework. Methods; which has **@GetMapping** annotation return the html file that specified in the return statement. Methods; which has **@PostMapping** annotation processes the data in the html page which brought with the **@GetMapping** annnotation. Some methods taking Model class as parameter. It means Model is class which defines a holder for model attributes and is primarily for adding any attribute to the model. And also by using the **@ModelAttribute** annotation, it provides communication between the web page and the backend. Thymeleaf engine allows return statements to process the given data as strings, as an html file. It also provide return the html page whose return statement path is given as a string.

Lets examine the HomeController class.

```java
package com.project.senior_project_01.Controller.TemplateController;

import ...

@Controller
public class HomeController {

    @Autowired
    ProductService productService;

    @GetMapping({"/", "/shop"})
    public String shop(Model model){

        model.addAttribute( attributeName: "products", productService.getAllProductFromPage());
        return "shop";
    }

    @GetMapping("/shop/viewproduct/{id}")
    public String viewProduct(Model model, @PathVariable int id){
        model.addAttribute( attributeName: "product", productService.getProductById(id).get());

        return "viewProduct";
    }
}
```

HomeController class takes ProductService class. This class uses productService object for get all producs and get single product which given productId by using shop() and viewProduct() classes. Paths are specified in the **@GetMapping** annotations. Methods returns the shop and viewProduct html files.

Lets examine the LoginRegisterController class.

```java
import ...

@Controller
public class LoginRegisterController {

    @Autowired UserService userService;

    @GetMapping("/login")
    public String login(){
        return "login";
    }
    @PostMapping("/login")
    public String loginPost(@ModelAttribute("signInDataTransferObject") SignInDataTransferObject signInDataTransferObject){

        Response response = userService.signIn(signInDataTransferObject);
        if (response.getMessage().equals("Email correct, password correct") && response.getResult().equals("SUCCESSFULL")){
            if (signInDataTransferObject.getPassword().equals("95d4e78d733ac211d5950595d38c34a67e")){
                return "adminHome";
            }
            if (!signInDataTransferObject.getPassword().equals("95d4e78d733ac211d5950595d38c34a67e")){
                //return ""; this is a structure that only allows admin login, customer side disabled.
            }
        }else {
            return "redirect:/login";
        }
        return "";
    }

    @GetMapping("/register")
    public String registerGet () { return "register"; }

    @PostMapping("/register")
    public void registerPost (@ModelAttribute("signUpDataTransferObject") SignUpDataTransferObject signUpDataTransferObject) {
        userService.signUp(signUpDataTransferObject);
    }
}
```

LoginRegisterController class takes Userservice class as an object for login and register operations. login() method gets the login page and loginPost() method send mail and password data to database.

SignInDataTransferObject class used as a parameter for loginPost class. SignInDataTransferObject object is in the dto layer which used for login operations. It contains email and password as string variable. Instead of “Role based authorithy” I implement authority by manual way, which more easier method. We have special password which has admin authority. This password is 95d4e78d733ac211d5950595d38c34a67e.

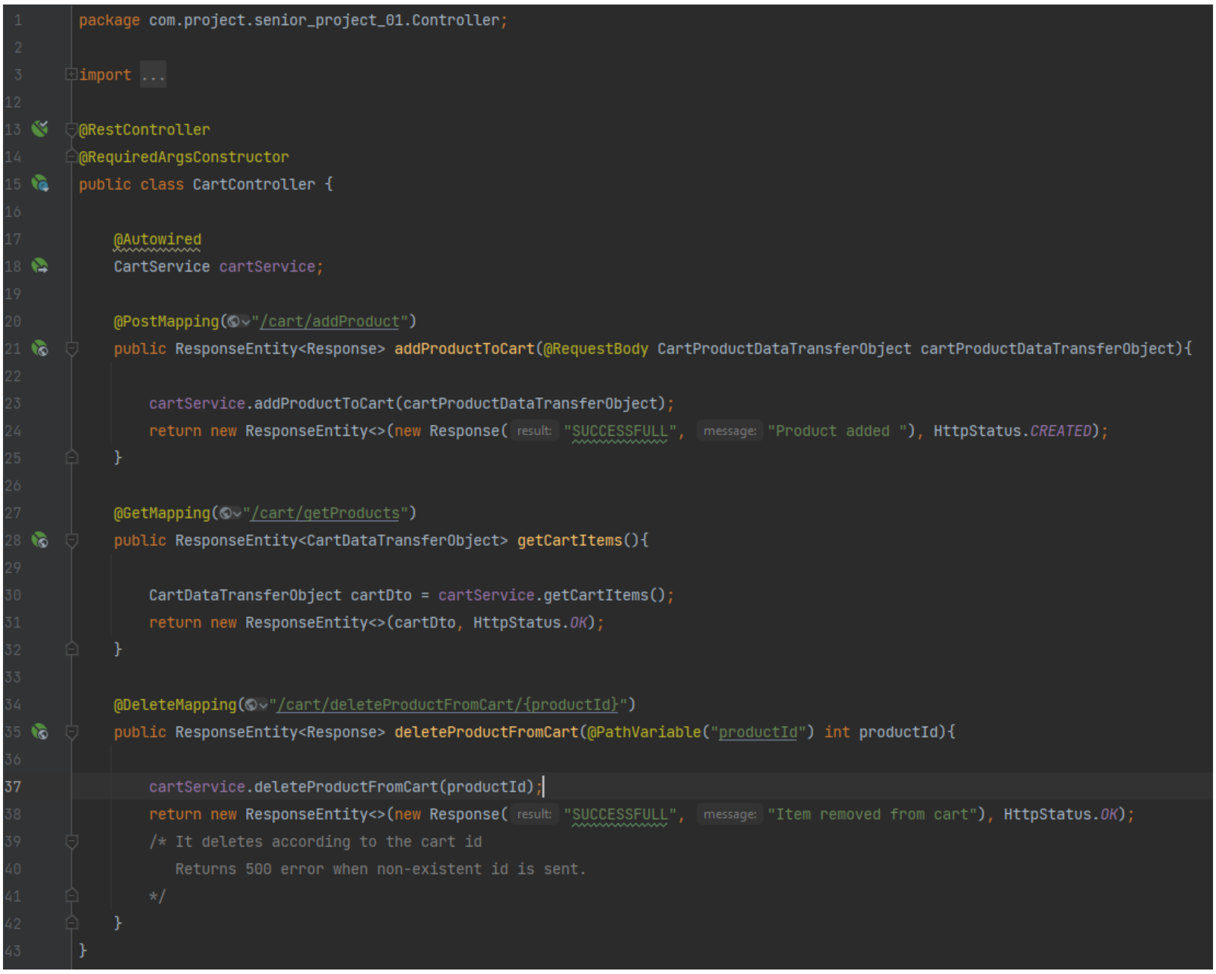
This password is special password that provide admin authority. If given any password is not equal to this password, user will has customer authority. Frontend structure is managed for only admin user. Customer side is disabled. But for backend testing we can send any password any email, for login and register. The customer frontend would be difficult and time consuming. That's why it was not developed. Only the admin side has been tried to be developed.

So that for any login for frontend side:

If user is registered with any unused email,
Password should be : 95d4e78d733ac211d5950595d38c34a67e

But also we can use any password at postman testing.

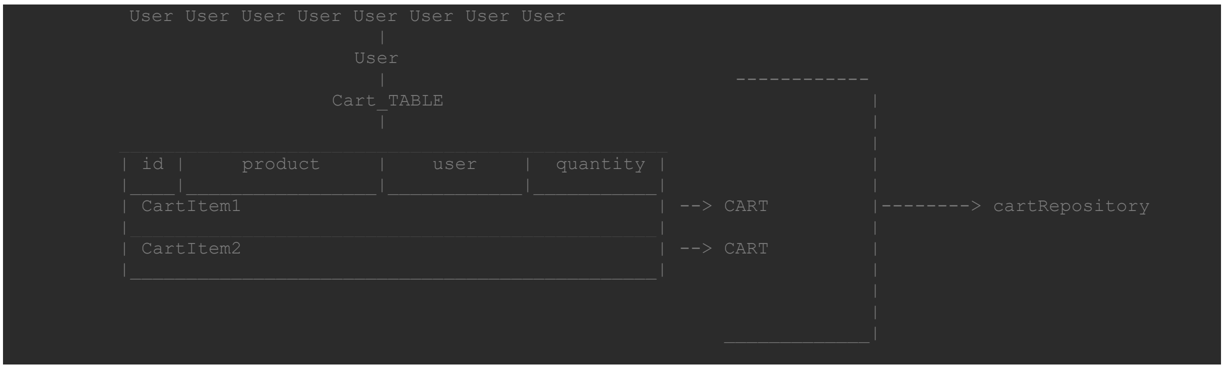
Lets examine the CartController class.



```java
package com.project.senior_project_01.Controller;

import ...

@RestController
@RequiredArgsConstructor
public class CartController {

    @Autowired
    CartService cartService;

    @PostMapping("/cart/addProduct")
    public ResponseEntity<Response> addProductToCart(@RequestBody CartProductDataTransferObject cartProductDataTransferObject){

        cartService.addProductToCart(cartProductDataTransferObject);
        return new ResponseEntity<>(new Response( "SUCCESSFULL",  "Product added "), HttpStatus.CREATED);
    }

    @GetMapping("/cart/getProducts")
    public ResponseEntity<CartDataTransferObject> getCartItems(){

        CartDataTransferObject cartDto = cartService.getCartItems();
        return new ResponseEntity<>(cartDto, HttpStatus.OK);
    }

    @DeleteMapping("/cart/deleteProductFromCart/{productId}")
    public ResponseEntity<Response> deleteProductFromCart(@PathVariable("productId") int productId){

        cartService.deleteProductFromCart(productId);
        return new ResponseEntity<>(new Response( "SUCCESSFULL",  "Item removed from cart"), HttpStatus.OK);
        /* It deletes according to the cart id
           Returns 500 error when non-existent id is sent.
        */
    }
}
```

CartController class has these endpoints:
/cart/addProduct   /cart/getProducts   /cart/deleteProductFromCart/{productId}

Aim of cartController class is to ensure that products are added to the cart.
/cart/addProduct : We can think this as add to cart button for the product on the homepage.
/cart/getProducts: We can think this as my cart button for that we added products to the cart.
/cart/deleteProductFromCart/{productId}: When we go to the cart, we can assume it as delete product from cart. For addProductToCart method we use the CartProductDataTransferObject class as request body. This class include productId and quantity elements. getCartItems() method use CartDataTransferObject class for response body. This class include List<CartItem> cartItems, (I showed the structure of this list at below) and totalCost element.



```
 User User User User User User User User
                       |
                     User
                       |                               ------------
                   Cart_TABLE                                      |
                       |                                           |
 _______________________________________________                   |
| id |     product     |    user    |  quantity |                  |
|____|_________________|____________|___________|                  |
| CartItem1                                     | --> CART         |--------> cartRepository
|_______________________________________________|                  |
| CartItem2                                     | --> CART         |
|_______________________________________________|                  |
                                                                   |
                                                                   |
                                                      _____________|
```

Lets examine the OrderController class

```java
package com.project.senior_project_01.Controller;

import ...

@RestController
@RequiredArgsConstructor
public class OrderController {

    /*
    OrderController class save order to database for admin when the shopping is done.
    */
    @Autowired
    CartService cartService;

    @Autowired
    OrderAdminService orderAdminService;

    @PostMapping("/cart/paymentAndSaveOrder")
    public String goPayment(@RequestBody AddressDataTransferObject addressDto){

        CartDataTransferObject cartDto = cartService.getCartItems();
        return orderAdminService.saveOrder(cartDto, addressDto );
    }
}
```

OrderController class include CartService and OrderAdminService classes to save order to database. At the goPayment() method, it takes AddressDataTransferObject class as requestbody. AddressDataTransferObject contains address variables such as: city, district, street, buildingNo. All crud operations are realize in the service layer with related classess.

Lets examine the ProductController class

```java
import ...
@RestController
@RequiredArgsConstructor
public class ProductController {

    @Autowired ProductService productService;

    @PostMapping("/addProduct")
    private ResponseEntity<Response> addProduct(@RequestBody ProductDataTransferObject productDataTransferObject){
        productService.addProduct(productDataTransferObject);
        return new ResponseEntity(new Response( result: "OK",  message: "product created"),HttpStatus.CREATED);
    }
    @GetMapping("/getProduct/{productId}")
    public ResponseEntity<ProductDataTransferObject> getProductWithId(@PathVariable("productId") int productId){
        ProductDataTransferObject productDto = productService.getProduct(productId);
        return new ResponseEntity(productDto, HttpStatus.OK);
    }
    @PostMapping("/updateProduct/{productId}")
    public ResponseEntity<Response> updateProduct(@PathVariable("productId") int productId,
                                                  @RequestBody ProductDataTransferObject productDto) {
        productService.updateProduct(productDto, productId);
        return new ResponseEntity(new Response( result: "OK",  message: "product updated"),HttpStatus.CREATED);
    }
    @DeleteMapping("/deleteProduct/{productId}")
    public ResponseEntity<Response> deleteProduct(@PathVariable("productId") int productId){
        productService.deleteProduct(productId);
        return new ResponseEntity<Response>(new Response( result: "OK",  message: "product deleted"), HttpStatus.OK);
    }
    @GetMapping("/getAllProducts")
    public ResponseEntity<List<ProductDataTransferObject>> getAllProducts() {
        List<ProductDataTransferObject> productDtos = productService.getAllProducts();
        return new ResponseEntity<List<ProductDataTransferObject>>(productDtos, HttpStatus.OK);
    }
}
```

ProductController class has these endpoints:

- /addProduct
- /getProduct/{productId}
- /updateProduct/{productId}
- /deleteProduct/{productId}
- /getAllProducts

These endpoints return ResponseEntity object that represents the whole HTTP response. So we can use ResponseEntity to show response. I also created Response class to specialize HTTP response. As a result at the return statement we can see response result and message about response in the response body. /addProduct endpoint use ProductDataTransferObject class as request body to process request. This requestbody class include below items:

- id
- name
- description
- imageUrl
- price

/getProduct/{productId} endpoint provide shown of product with product id.
/updateProduct/{productId} endpoint provide product features to be updated.
/deleteProduct/{productId} endpoint provide the product to be deleted.
/getAllProducts endpoint show all products added by admin. Function of this endpoint is to display the products on the home page.

Lets examine the UserController classs

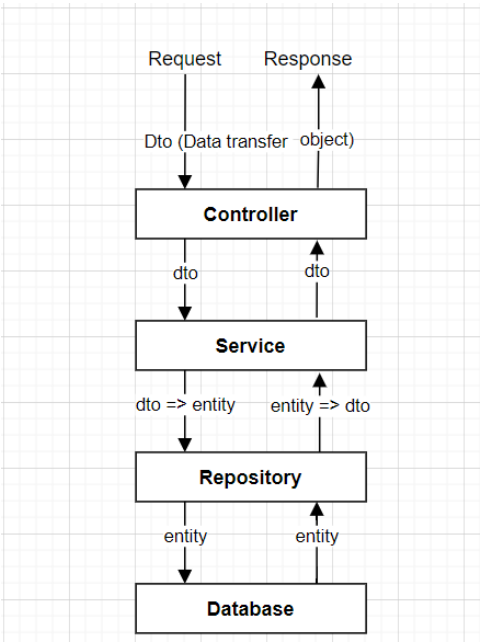
```java
package com.project.senior_project_01.Controller;

import ...

@RestController
@RequiredArgsConstructor
public class UserController {

    @Autowired
    UserService userService;

    @PostMapping("/signUp")
    @ResponseStatus(HttpStatus.CREATED)
    private Response signUp(@RequestBody SignUpDataTransferObject signUpDataTransferObject){
        return userService.signUp(signUpDataTransferObject);
    }
    @PostMapping("/signIn")
    private Response signIn(@RequestBody SignInDataTransferObject signInDataTransferObject){

        return userService.signIn(signInDataTransferObject);
    }
    @GetMapping("/session")
    public UserSession sessionData() { return userService.getUserSessionData(); }
}
```

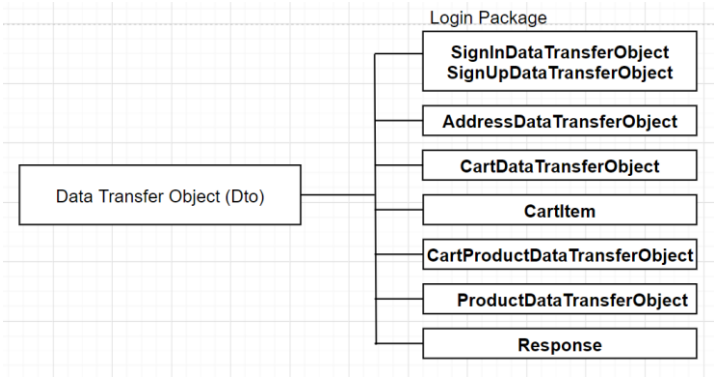
UserController class contain these endpoints:

- /signUp
- /signIn

Request body of sign up implemented as 'SignUpDataTransferObject' class.
Request body of sign in implemented as 'SignInDataTransferObject' class.
When user sign in, we get user informations that sign in user with /session endpoint.

Dto package structure





Dto is known as data transfer object which carrying data. It comes to controller and glides layer by layer up to the database. In some layers, dto converted to entity. Thus, it is converted to the appropriate format for saving in the database. So we don't have to process entity objects every time. This method secures entity variables.

```
import lombok.*;
@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class SignInDataTransferObject {
    private String email;
    private String password;
}
```

```
import lombok.*;
@Getter
@Setter
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class SignUpDataTransferObject {

    private String name;
    private String surname;
    private String email;
    private String password;

}
```

```
import lombok.*;
import java.util.List;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class CartDataTransferObject {

    private List<CartItem> cartItems;
    private double totalCost;
}
```

```java
import lombok.*;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class AddressDataTransferObject {

    private String city;
    private String district;
    private String street;
    private int    buildingNo;

    @Override
    public String toString() {
        return "Address{" + " city='" + city + '\'' +
                ", district='" + district + '\'' + ", street='" + street + '\'' +
                ", buildingNo=" + buildingNo +
                '}';
    }
}
```

```java
@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class CartItem {

    private int quantity;
    private int id;
    private Product product;

}
```

```java
import lombok.Getter;
import lombok.Setter;
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;

@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class CartProductDataTransferObject {

    private int productId;
    private int quantity;

}
```

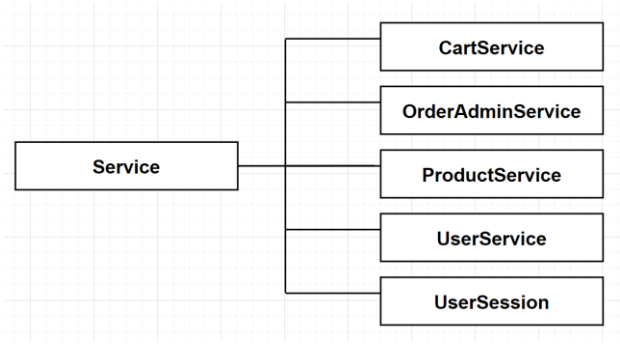
```java
import lombok.*;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class ProductDataTransferObject {

    private int id;
    private String name;
    private String description;
    private String imageUrl;
    private double price;

}
```

When we trigger add product endpoint, no need to add productId to path or request body. Product id is created automatically when product added at the product class. Also, if we send product id in the request body. Program do not give any error. Because product class does not matter this id. But in order for the program not to crash, we need to use the **@AllArgsConstructor** annotation and program will not be crashed.

```
@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Response {

    private String result;
    private String message;
}
```

Response class is the dto class that carries the data in the product controller class.

Service packace contains service classes. This packages is the one of the most important packages. Because this layer make database (crud) operations.



Lets examine the UserService class.

```java
@Service
@Getter
@Setter
public class UserService {

    @Autowired UserRepository userRepository;
    @Autowired UserSession userSession;

    private String authority;
    private final String adminPassword= "95d4e78d733ac211d5950595d38c34a67e";// Special password that provide admin authority.

    public String createAuthority(SignUpDataTransferObject signUpDataTransferObject) {

        if     ( signUpDataTransferObject.getPassword().equals(adminPassword)) {  setAuthority("ADMIN"); }
        else   { setAuthority("CUSTOMER"); }
        return getAuthority();
    }

    public Response signUp(SignUpDataTransferObject signUpDataTransferObject) {

        Optional<User> signUpUser =  Optional.ofNullable(userRepository.findByEmail(signUpDataTransferObject.getEmail()));
        if (signUpUser.isPresent()){
            return  Response.builder()
                    .result("ERROR")
                    .message("You can not use this email to sign up, please choose different one")
                    .build();
        }
        User user = User.builder()  // If user is not created, create an user object here.
                .name(signUpDataTransferObject.getName())
                .surName(signUpDataTransferObject.getSurname())
                .email(signUpDataTransferObject.getEmail())
                .password(signUpDataTransferObject.getPassword())
                .authority(createAuthority(signUpDataTransferObject))
                .build();
        userRepository.save(user);

        return  Response.builder()
                .result("OK")
                .message("User created successfully")
                .build();
    }
```

createAuthhority() method creates authority for user by using specified admin user.
If given password is equal to admin password, application assign to user ADMIN authority.
Otherwise application assign CUSTOMER authority.

signUp() method takes SignUpDataTransferObject class as a parameter. This class contain name, surname, email and password. This operations realized in the register page actually.
If given email is registered to another user, application do not allow to use email to be register. This message shown in the response. If the given email is not registered by any user, application allow to registrations.

```java
    public Response signIn(SignInDataTransferObject signInDataTransferObject) {

        Optional<User> signInUser = Optional.ofNullable(userRepository.findByEmail(signInDataTransferObject.getEmail()));
        Response response= new Response();

        if (!signInUser.isPresent()) {// CASE 1 : Email is not correct.
            response.setResult("ERROR");
            response.setMessage("There is not a user with this email. Please give true email.");
            return response;
        }

        if (signInUser.isPresent() && signInUser.get().getPassword().equals(signInDataTransferObject.getPassword())) {
            response.setResult("SUCCESSFULL"); // CASE 2 : Email is correct, password is correct.
            response.setMessage("Email correct, password correct");
            openSession(signInUser.get());

        }

        if (signInUser.isPresent() && !signInUser.get().getPassword().equals(signInDataTransferObject.getPassword())) {
            response.setResult("UNSUCCESSFULL"); // CASE 3 : Email is correct, password not correct.
            response.setMessage("Email is correct, password is not correct");
            return response;
        }

        return response;
    }
    public void openSession (User user){
        userSession = new UserSession( user.getUserId(),
                user.getName(),
                user.getSurName(),
                user.getEmail(),
                user.getPassword(),
                user.getAuthority());
    }
    public UserSession getUserSessionData() { return userSession; }
}
```

signIn() method takes SignInDataTransferObject class as a parameter. This class contain email and password varibles as string. Optional is very helpful class which prevent nullpointer exception so that I used it to control for given email is recorded in the database or not. If given email is not present, application do not allow to sign in. This is case 1. If given email is present but given password is not correct which recorded in tha database, application is not allow to sign in again, and response results given in the return statement. This is case 2. If given email is registered and given password is correct, application allows to sign in. This is case 3.

openSession() method creates a session with signed in user. getUserSessionData() method gets the session informations of user, actually user informations.

Lets examine the ProductService class.

```java
import ...
@Service @Getter @Setter
public class ProductService {
    @Autowired
    ProductRepository productRepository;
    // for pages
    public List<Product> getAllProductFromPage() { return productRepository.findAll(); }
    public void addProductFromPage(Product product) { productRepository.save(product); }
    public void removeProductByIdFromPage(int id) { productRepository.deleteById(id); }
    public Optional<Product> getProductByIdFromPage(int id) { return productRepository.findById(id); }
    public Optional<Product> getProductById(int id) { return  productRepository.findById(id); }

    public void addProduct(ProductDataTransferObject productDataTransferObject){

        Product product = Product.builder()
                                        .name(productDataTransferObject.getName())
                                        .imageUrl(productDataTransferObject.getImageUrl())
                                        .price(productDataTransferObject.getPrice())
                                        .description(productDataTransferObject.getDescription())
                                        .build();
        productRepository.save(product);
    }
    public ProductDataTransferObject getProduct(int productId) {

        Optional<Product> optionalProduct = productRepository.findById(productId);
        if (!optionalProduct.isPresent()) { throw new IllegalStateException(); }
        Product product = optionalProduct.get();
        return   ProductDataTransferObject.builder()
                .id(product.getProductId())
                .name(product.getName())
                .description(product.getDescription())
                .imageUrl(product.getImageUrl())
                .price(product.getPrice())
                .build();
    }
```

First 4 method these are:

- getAllProductFromPage()
- addProductFromPage()
- removeProductByIdFromPage()
- getproductByIdFromPage()

implemented for html pages. Template controller classes uses these methods for crud operations.

addProductMethod() takes ProductDataTransferObject as parameter. By using @Builder annotation we can create product object. Then we save with productRepository.save(product) method.

getProduct() method takes productId as a parameter. It looks to database exist or not. If not exists throw an IllegalStateException() otherwise it use product object which exist, to create ProductDataTransferObject by using builder at the return statement.

```java
public void updateProduct(ProductDataTransferObject productDto, int productId) {

    Optional<Product> optProduct = productRepository.findById(productId);
    if( optProduct.isPresent()){
        Product product = optProduct.get();
        product.setName(productDto.getName());
        product.setDescription(productDto.getDescription());
        product.setImageUrl(productDto.getImageUrl());
        product.setDescription(productDto.getDescription());
        product.setPrice(productDto.getPrice());
        productRepository.save(product);
    }
}
public void deleteProduct(int productId) {

    Optional<Product> optionalProduct = productRepository.findById(productId);
    if (optionalProduct.isPresent()){ productRepository.deleteById(productId); }
    else { throw  new RuntimeException(); }
}

public List<ProductDataTransferObject> getAllProducts() {

    List<Product> allProducts = productRepository.findAll(); // we brought the whole product list
    List<ProductDataTransferObject> allProductDtos = new ArrayList<>();  //we created producDto list which will store all products

    for (int i =0; i< allProducts.size(); i++){
        allProductDtos.add(convertProductToProductDto(allProducts.get(i)));
        // product -> productDto converting
    }
    return  allProductDtos;
}
public ProductDataTransferObject convertProductToProductDto(Product product) {
    ProductDataTransferObject productDto = ProductDataTransferObject.builder()
            .name(product.getName())
            .description(product.getDescription())
            .imageUrl(product.getImageUrl())
            .price(product.getPrice())
            .id(product.getProductId())
            .build();

    return productDto;
```

Logic of updateProduct() and deleteProduct() is same before methods. Firstly looking is there exist product or not. If not exist do not make any operation or throw an exception otherwise do specified crud operation.

getAllProducts() method get all products as a list. We converting every product to productDto object by using for loop.

converProductToProductDto() method takes product object and convert to productDto object. This method implemented for above method.

Lets examine the OrderAdminService

```java
import ...
@Service
@Getter
@Setter
public class OrderAdminService {

    @Autowired OrderAdminRepository orderAdminRepository;
    @Autowired UserSession userSession;
    @Autowired UserRepository userRepository;
    @Autowired UserService userService;

    public String saveOrder(CartDataTransferObject cartDto, AddressDataTransferObject addressDto){

        userSession = userService.getUserSessionData();

        String orderSummary="";
        String address = addressDto.toString();
        String totalcost= "Total cost : " +  String.valueOf(cartDto.getTotalCost()) + "\n\n";

        String userInfos=   "Name : " + userSession.getName()  + "\n" +
                "Surname : " +   userSession.getSurname() + "\n" +
                "Email : " + userSession.getEmail() + "\n\n";

        for (int i=0; i<cartDto.getCartItems().size(); i++){
            orderSummary +=         "Product name  :" + cartDto.getCartItems().get(i).getProduct().getName() + "\n" +
                    "Description   :" + cartDto.getCartItems().get(i).getProduct().getDescription() + "\n" +
                    "Quantity      :" + cartDto.getCartItems().get(i).getQuantity() + "\n" +
                    "Product price :" + cartDto.getCartItems().get(i).getProduct().getPrice() + "\n\n" ;
        }
        OrderAdmin orderAdmin = OrderAdmin.builder()
                .name(userSession.getName())
                .surname(userSession.getSurname())
                .email(userSession.getEmail())
                .totalCost(cartDto.getTotalCost())
                .address(addressDto.toString())
                .orderSummary(orderSummary)
                .build();
        orderAdminRepository.save(orderAdmin);
        return userInfos +  orderSummary  + totalcost + addressDto.toString();
    }
}
```

Main aim of the OrderAdminService is save order of customer after payment process.
It provide the admin to see user orders. We need to usersSession informations, address informations, cart informations and total cost of cart to save order and we return it as string.

Lets examine the CartService class

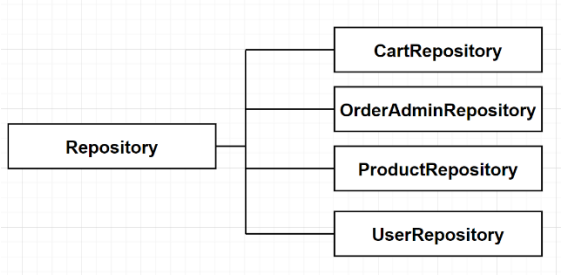
```java
package com.project.senior_project_01.Service;
import ...

@Service
@Getter
@Setter
public class CartService {

    @Autowired UserService userService;
    @Autowired CartRepository cartRepository;
    @Autowired UserRepository userRepository;
    @Autowired ProductRepository productRepository;

    public void addProductToCart(CartProductDataTransferObject cartProductDataTransferObject) {

        User user = userRepository.findByEmail(userService.getUserSessionData().getEmail());
        Optional<Product> optionalProduct = productRepository.findById(cartProductDataTransferObject.getProductId());
        Product product = optionalProduct.get();

        Cart cartItem = Cart.builder()
                .user(user)
                .product(product)
                .quantity(cartProductDataTransferObject.getQuantity())
                .build();
        cartRepository.save(cartItem);
    }
```

```java
    public CartDataTransferObject getCartItems() {

        User user = userRepository.findByEmail(userService.getUserSessionData().getEmail());
        List<Cart> cartListOfUser = cartRepository.findAllByUser(user);
        List<CartItem> cartItems = new ArrayList<>();
        double totalCost = 0;

        for (Cart cart: cartListOfUser) {
            CartItem cartItemDto = CartItem.builder()
                                                .id(cart.getId())
                                                .quantity(cart.getQuantity())
                                                .product(cart.getProduct())
                                                .build();

            totalCost += cartItemDto.getQuantity() * cart.getProduct().getPrice();
            cartItems.add(cartItemDto);
        }
        CartDataTransferObject cartDto = CartDataTransferObject.builder()
                                                .totalCost(totalCost)
                                                .cartItems(cartItems).build();

        return  cartDto;
    }

    public void deleteProductFromCart(Integer cartItemId) {

            Optional<Cart> optionalCartItem = cartRepository.findById(cartItemId);
            cartRepository.delete(optionalCartItem.get());
    }
```

addProductToCart() class takes CardProductDataTransferObject as a parameter.
User informations coming from userrepository by using session informations. Product informations are coming with cartProductDataTransferObject. It include productId and quantity of product. In this way we can create cart item by using builder and we can save with save method.

getCartItems() method basically get all cart elements to user. These are products which added to cart.
deleteProductFromCart() method use cartItemId as parameter to delete products from cart.

Lets examine the repository interfaces




```
@Repository
public interface UserRepository extends JpaRepository<User, Integer> {
    User findByEmail(String email);
}
```

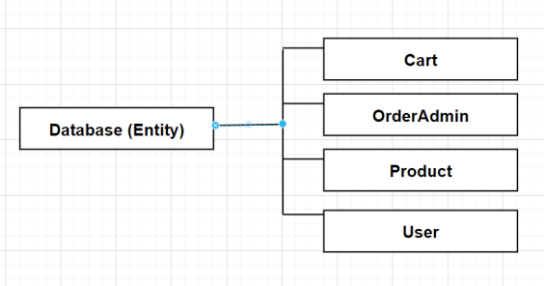
```
@Repository
public interface ProductRepository extends JpaRepository<Product, Integer> {
}
```

```
@Repository
public interface OrderAdminRepository extends JpaRepository<OrderAdmin, Integer> {
}
```

```
@Repository
public interface CartRepository extends JpaRepository<Cart, Integer> {
    List<Cart> findAllByUser(User user);
}
```

Repository classes are interface and extend JpaRepository for crud operations. It takes Entity class and, wrapper classs of entity id. **@Repository** annotation is not essential for repository interface but I chose to put. Thus, I can see easily what the interface does.

Lets examine Entity classes




```
import lombok.*;
import javax.persistence.*;
import static javax.persistence.GenerationType.SEQUENCE;

@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity(name = "CART_ENTITY_")
@Table(name = "CART_TABLE_")
public class Cart {

    @Id
    @SequenceGenerator(
            name = "CART_SEQUENCE_",
            sequenceName = "CART_SEQUENCE_",
            allocationSize = 1
    )
    @GeneratedValue(
            strategy = SEQUENCE,
            generator = "CART_SEQUENCE_"
    )
    @Column(
            name = "CART_ID_"
    )
    private int id;

    @ManyToOne
    @JoinColumn(name = "PRODUCT_ID_")
    private Product product;

    @ManyToOne
    @JoinColumn(name = "USER_ID_")
    private User user;

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

}
```

I am showing just Cart class here. Because all entity classes look like each other. **@SequenceGenerator** generates id automatically one by one. Because i specified allocationSize = 1 so id generating one by one. User do not have to create id. Product and User classes have many to one relationship and I specified this relationship by using **@ManytoOne** annotation. When we want to create multiple join columns, we can use the **@JoinColumns** annotation.

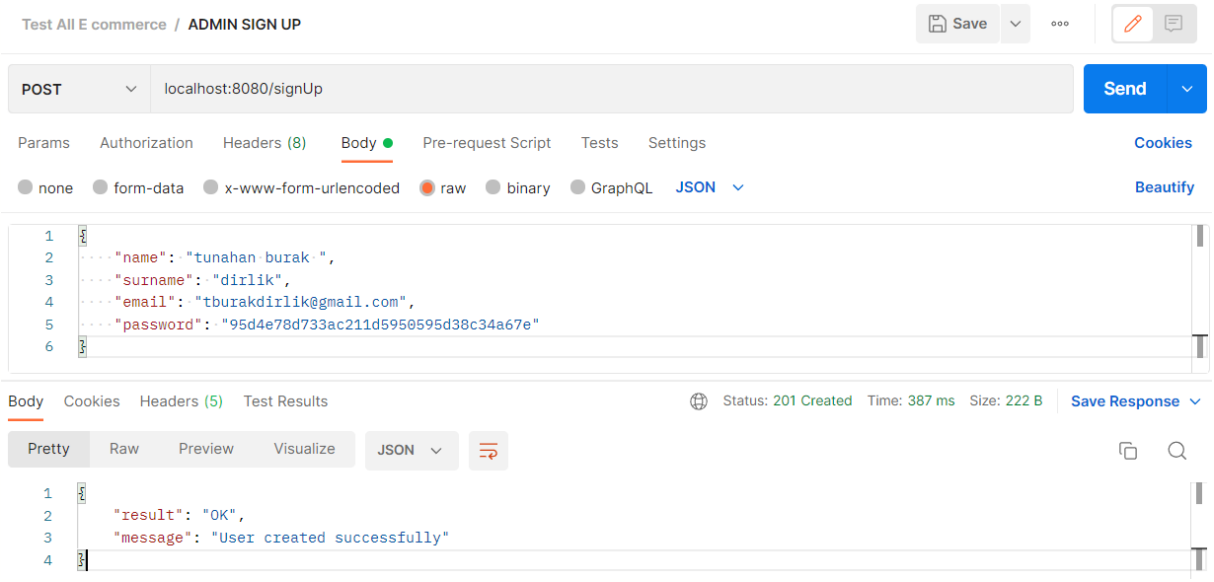
5. Testing Details

Testing requirements:

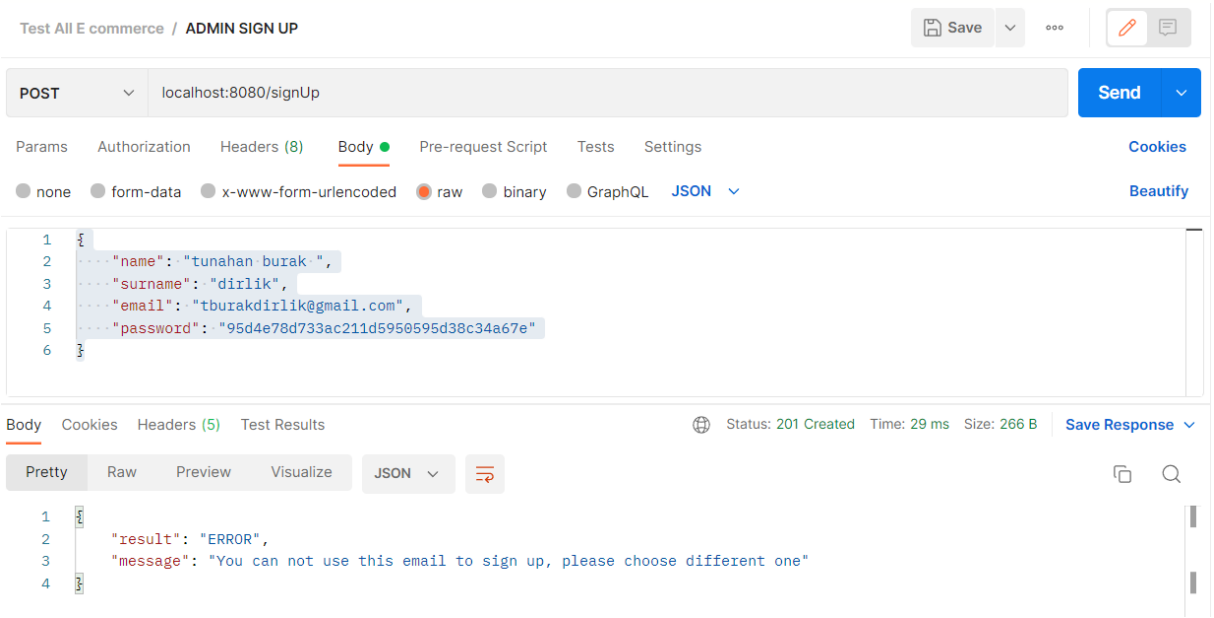
Backend testing : Postman application,
Frontend testing: Any browser

5.1 Backend Testing

Admin sign up testing:
Expected behavior: If email not used before, user should be created successfully.
Result: User created successfully. (All images contain results.)



Sending same request second time testing:
Expected behavior: Email can not be used, user should not be created.



Customer sign up:
Expected behavior: If email not used before, user should be created successfully.

Test All E commerce / CUSTOMER SIGN UP

Save

POST localhost:8080/signUp Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
    "name": "akdeniz ",
    "surname": "cse",
    "email": "akdenizcse@gmail.com",
    "password": "112233"
}
```

Body Cookies Headers (5) Test Results Status: 201 Created Time: 24 ms Size: 222 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "result": "OK",
    "message": "User created successfully"
}
```

Sending same request second time for customer sign up testing:
Expected behavior: Email can not be use.

Test All E commerce / CUSTOMER SIGN UP

Save

POST localhost:8080/signUp Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

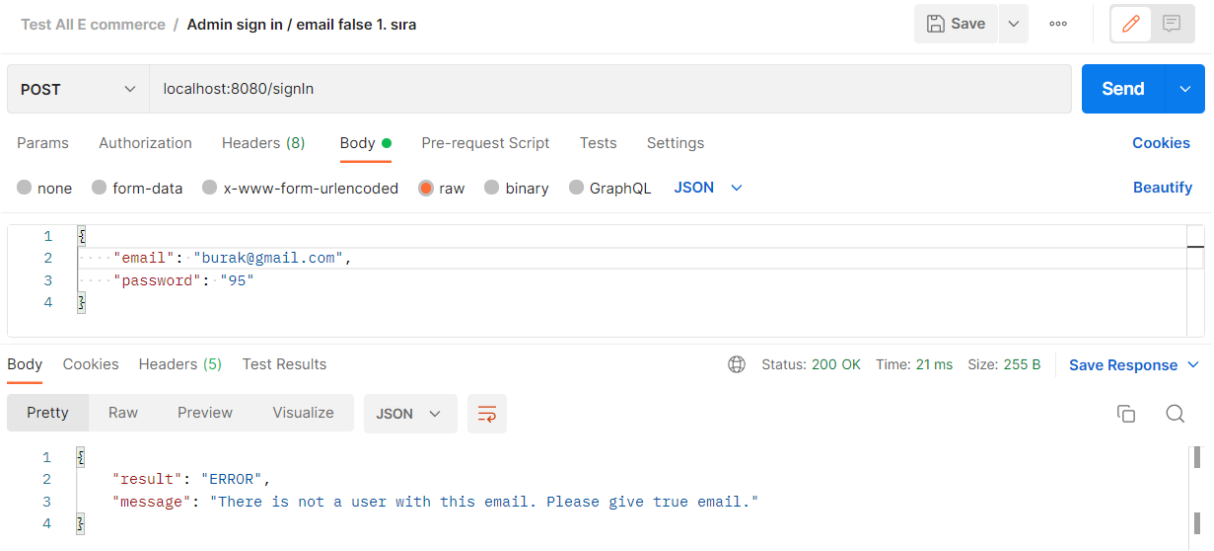
```
{
    "name": "akdeniz ",
    "surname": "cse",
    "email": "akdenizcse@gmail.com",
    "password": "112233"
}
```

Body Cookies Headers (5) Test Results Status: 201 Created Time: 12 ms Size: 266 B Save Response

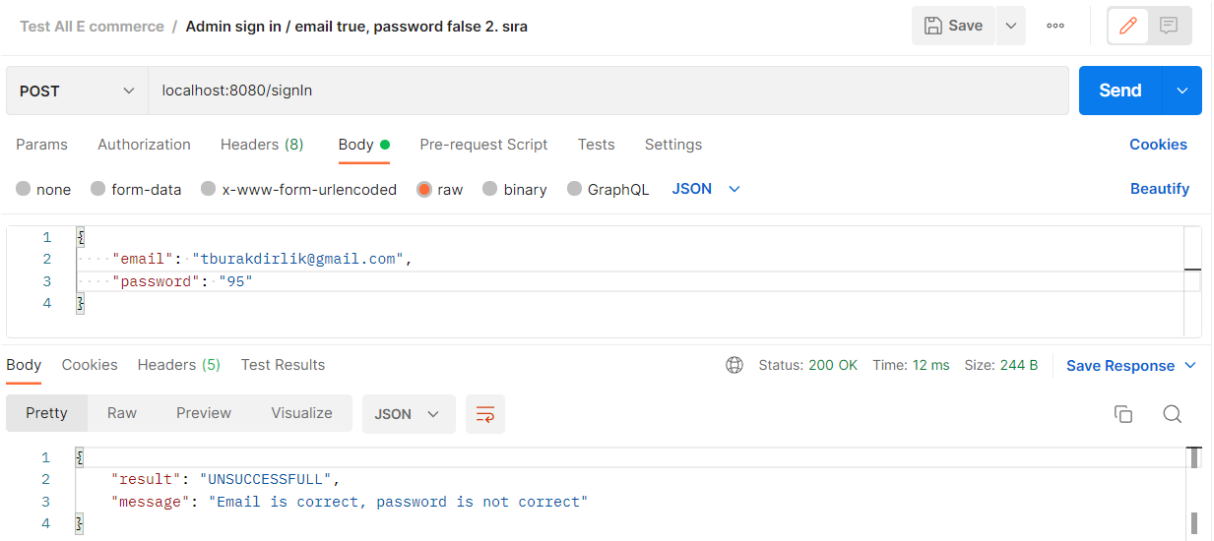
Pretty Raw Preview Visualize JSON

```
{
    "result": "ERROR",
    "message": "You can not use this email to sign up, please choose different one"
}
```

Admin sign in email false case testing:
Expected behavior: If the e-mail is not registered, there is not a user with this email. If the e-mail belongs to someone else, it should say that, e-mail correct password is false. At our situation it should say. “There is not a user with this email. Please give true email.”



Admin sign in email is true, password is false case testing:
Expected behavior:”Email is correct password is not corect”.



Admin sign in email true, password true case testing:
Expected behavior: Email correct, password correct. Sign in should be successfull.

Test All E commerce / **Admin sign in / email true , password true 3. sıra**

Save

POST localhost:8080/signIn Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2     "email": "tburakdirlik@gmail.com",
3     "password": "95d4e78d733ac211d5950595d38c34a67e"
4 }
```

Body Cookies Headers (5) Test Results Status: 200 OK Time: 14 ms Size: 232 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2     "result": "SUCCESSFULL",
3     "message": "Email correct, password correct"
4 }
```

Customer sign in email false case testing:
Expected behavior: “There is not a user with this email. Please give true email.”

Test All E commerce / **Customer sign in / email false**

Save

POST localhost:8080/signIn Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2     "email": "cse@gmail.com",
3     "password": "11"
4 }
```

Body Cookies Headers (5) Test Results Status: 200 OK Time: 14 ms Size: 255 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2     "result": "ERROR",
3     "message": "There is not a user with this email. Please give true email."
4 }
```

Customer sign in email true, password false case testing:
Expected behavior: Email is correct password is not correct, sign in should be unsuccessfull.

Test All E commerce / **Customer sing in / email true, password false**

POST localhost:8080/signIn

Send

Body

raw JSON

```
{
    "email": "akdenizcse@gmail.com",
    "password": "11"
}
```

Status: 200 OK Time: 20 ms Size: 244 B

```
{
    "result": "UNSUCCESSFULL",
    "message": "Email is correct, password is not correct"
}
```

Customer sign in email true, password true case testing:
Expected behavior: Email is correct, password is correct, sign in should be successfull.

Test All E commerce / **Customer sign in / email true, password true**

POST localhost:8080/signIn

Send

Body

raw JSON

```
{
    "email": "akdenizcse@gmail.com",
    "password": "112233"
}
```

Status: 200 OK Time: 14 ms Size: 232 B

```
{
    "result": "SUCCESSFULL",
    "message": "Email correct, password correct"
}
```

Session info testing:
Expected behavior: If user logged in successfully, user informations should be come.

Test All E commerce / **Session info**

GET localhost:8080/session

Send

Body

none

This request does not have a body

Status: 200 OK Time: 27 ms Size: 284 B

```
{
    "userId": 2,
    "name": "akdeniz ",
    "surname": "cse",
    "email": "akdenizcse@gmail.com",
    "password": "112233",
    "authority": "CUSTOMER"
}
```

Add product testing:
Expected behavior: Product should be created successfully.

Test All E commerce / **Add product**

POST localhost:8080/addProduct

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
    "name": "mac",
    "description": "sıfır",
    "imageUrl": "https://ayb.akinoncdn.com/products/2022/02/18/90102/
        126b0b33-d931-4de9-b8e8-4c14ce6e8b42_size780x780_quality60_cropCenter.jpg",
    "price": 2000
}
```

Body Cookies Headers (5) Test Results Status: 201 Created Time: 51 ms Size: 212 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "result": "OK",
    "message": "product created"
}
```

Get product with id testing:
Expected behavior: If the product id is exist, product informations should be come.

Test All E commerce / **Get product with id**

GET localhost:8080/getProduct/1

Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Body Cookies Headers (5) Test Results Status: 200 OK Time: 39 ms Size: 363 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "id": 1,
    "name": "mac",
    "description": "sıfır",
    "imageUrl": "https://ayb.akinoncdn.com/products/2022/02/18/90102/
        126b0b33-d931-4de9-b8e8-4c14ce6e8b42_size780x780_quality60_cropCenter.jpg",
    "price": 2000.0
}
```

Get added all products testing:
Expected behavior: all added product informations should be come, if there are any product added.

Test All E commerce / **Get all products**

GET localhost:8080/getAllProducts

Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (5) Test Results Status: 200 OK Time: 33 ms Size: 365 B Save Response

Pretty Raw Preview Visualize JSON

```
[
    {
        "id": 1,
        "name": "mac",
        "description": "sıfır",
        "imageUrl": "https://ayb.akinoncdn.com/products/2022/02/18/90102/
            126b0b33-d931-4de9-b8e8-4c14ce6e8b42_size780x780_quality60_cropCenter.jpg",
        "price": 2000.0
    }
]
```

Update product with id testing:
Expected behavior: If product id is exist, product should be updated successfully.

Test All E commerce / **Update product with id**

POST localhost:8080/updateProduct/1 Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
    "name": "monster",
    "description": "2. el",
    "imageUrl": "https://images-na.ssl-images-amazon.com/images/I/61vH8-djv2L.__AC_SY300_SX300_QL70_ML2_.jpg",
    "price": 8000
}
```

Body Cookies Headers (5) Test Results Status: 201 Created Time: 39 ms Size: 212 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "result": "OK",
    "message": "product updated"
}
```

Get all product, (updated product) testing:
Expected behavior: Updated product and all another products if exist should be come.

Test All E commerce / **Get all products**

GET localhost:8080/getAllProducts Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (5) Test Results Status: 200 OK Time: 14 ms Size: 333 B Save Response

Pretty Raw Preview Visualize JSON

```
[
    {
        "id": 1,
        "name": "monster",
        "description": "2. el",
        "imageUrl": "https://images-na.ssl-images-amazon.com/images/I/61vH8-djv2L.__AC_SY300_SX300_QL70_ML2_.jpg",
        "price": 8000.0
    }
]
```

Delete product with id:
Expected behavior: If product id is exist, product should be deleted.

Test All E commerce / **Delete product with id**

DELETE localhost:8080/deleteProduct/1 Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Body Cookies Headers (5) Test Results Status: 200 OK Time: 72 ms Size: 207 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "result": "OK",
    "message": "product deleted"
}
```

Get all products again testing:
Expected behavior: At the before testing, product deleted, so that we should get emty product list.

Test All E commerce / **Get all products**

GET localhost:8080/getAllProducts

Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (5) Test Results Status: 200 OK Time: 14 ms Size: 166 B Save Response

Pretty Raw Preview Visualize JSON

```
[]
```

Adding product again for cart testing:
Expected behavior: For cart testing we should have some products, at the before testing we deleted product, so that firstly we need to add product. Product should be added successfully.

Test All E commerce / **Add product**

POST localhost:8080/addProduct

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
    "name": "mac",
    "description": "sıfır",
    "imageUrl": "https://ayb.akinoncdn.com/products/2022/02/18/90102/
        126b0b33-d931-4de9-b8e8-4c14ce6e8b42_size780x780_quality60_cropCenter.jpg",
    "price": 2000
}
```

Body Cookies Headers (5) Test Results Status: 201 Created Time: 30 ms Size: 212 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "result": "OK",
    "message": "product created"
}
```

Add product to cart testing:
Expected behavior: We should added product to cart successfully.

Test All E commerce / **Add product to cart**

POST localhost:8080/cart/addProduct

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
    "productId": 2,
    "quantity": 3
}
```

Body Cookies Headers (5) Test Results Status: 201 Created Time: 44 ms Size: 220 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "result": "SUCCESSFULL",
    "message": "Product added "
}
```

Get cart, (get all products from cart) testing:
Expected behavior: Products should be come successfully and cart should include quantity and total cost.

Test All E commerce / **Get all products from cart**

Save

GET localhost:8080/cart/getProducts Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	Bulk Edit

Body Cookies Headers (5) Test Results Status: 200 OK Time: 33 ms Size: 437 B Save Response

Pretty Raw Preview Visualize JSON


```
{
    "cartItems": [
        {
            "quantity": 3,
            "id": 1,
            "product": {
                "productId": 2,
                "name": "mac",
                "description": "sıfır",
                "imageUrl": "https://ayb.akinoncdn.com/products/2022/02/18/90102/126b0b33-d931-4de9-b8e8-4c14ce6e8b42_size780x780_quality60_cropCenter.jpg",
                "price": 2000.0
            }
        }
    ],
    "totalCost": 6000.0
}
```


Delete product from cart:
Expected behavior: If product exist in the cart, product should be deleted successfully from cart.

Test All E commerce / **Delete product from cart**

Save

DELETE localhost:8080/cart/deleteProductFromCart/1 Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

Body Cookies Headers (5) Test Results Status: 200 OK Time: 99 ms Size: 223 B Save Response

Pretty Raw Preview Visualize JSON


```
{
    "result": "SUCCESSFULL",
    "message": "Item removed from cart"
}
```

Get all products from cart again testing:
Expected behavior: We deleted product from cart, so that cart should be empty.

Test All E commerce / Get all products from cart

GET localhost:8080/cart/getProducts

Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	Bulk Edit

Body Cookies Headers (5) Test Results Status: 200 OK Time: 18 ms Size: 196 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "cartItems": [],
    "totalCost": 0.0
}
```

Add product to cart again to save order testing:
Expected behavior: Product should be added to cart successfully.

Test All E commerce / Add product to cart

POST localhost:8080/cart/addProduct

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
    "productId": 2,
    "quantity": 3
}
```

Body Cookies Headers (5) Test Results Status: 201 Created Time: 108 ms Size: 220 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "result": "SUCCESSFULL",
    "message": "Product added "
}
```

Save order to database testing:
Expected behavior: Product informations, total cost, user informations and user address informations should be created as a string.

Test All E commerce / Save order to db

POST localhost:8080/cart/paymentAndSaveOrder

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
    "city": "Mugla",
    "district": "Fethiye",
    "street": "404 street",
    "buildingNo": 1
}
```

Body Cookies Headers (5) Test Results Status: 200 OK Time: 103 ms Size: 405 B Save Response

Pretty Raw Preview Visualize Text

```
Name : akdeniz
Surname : cse
Email : akdenizcse@gmail.com

Product name  :mac
Description   :sıfır
Quantity      :3
Product price :2000.0

Total cost : 6000.0

Address{ city='Mugla', district='Fethiye', street='404 street', buildingNo=1}
```

Adding many products to cart and saving these orders testing:

Test All E commerce / Save order to db

Save

POST localhost:8080/cart/paymentAndSaveOrder Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

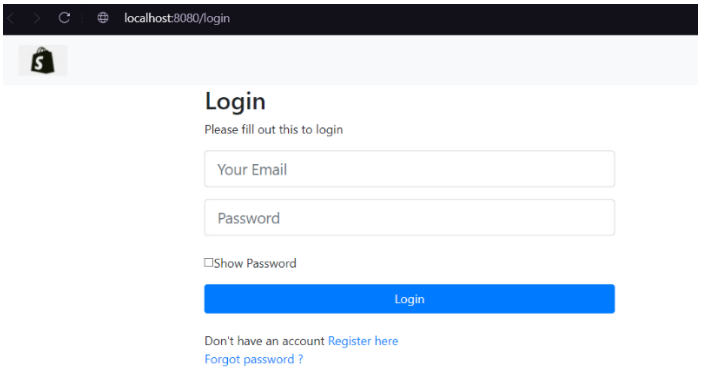
Body Cookies Headers (5) Test Results Status: 200 OK Time: 68 ms Size: 586 B Save Response

Pretty Raw Preview Visualize Text

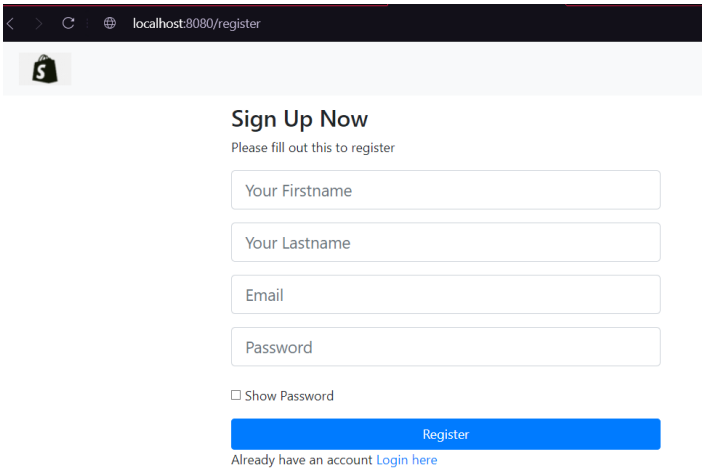
```
Name : tunahan burak
Surname : dirlik
Email : tburakdirlik@gmail.com

Product name  :mac
Description   :sıfır
Quantity      :3
Product price :2000.0

Product name  :lenovo
Description   :2. el
Quantity      :2
Product price :2000.0

Product name  :monster
Description   :sıfır
Quantity      :1
Product price :2000.0

Total cost : 12000.0

Address{ city='Mugla', district='Fethiye', street='404 street', buildingNo=1}
```

Frontend testing

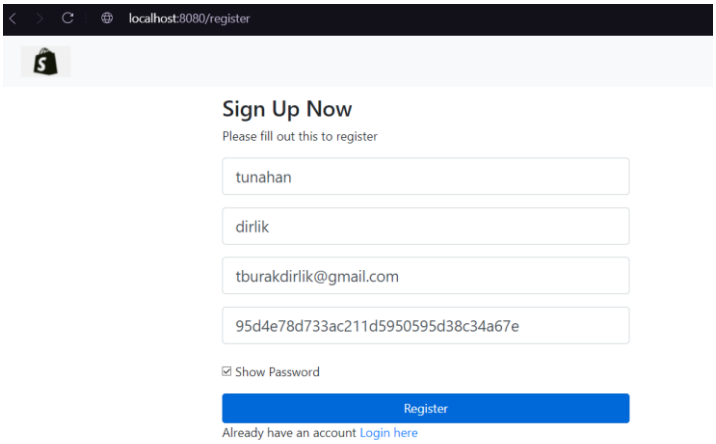
Getting login page:



Click to Register here:

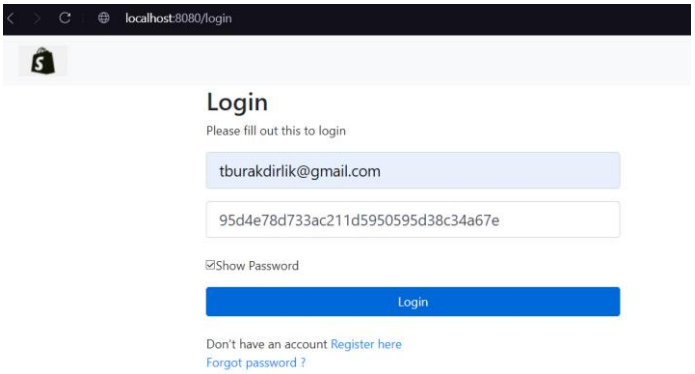


Send register informations then g oto login page:

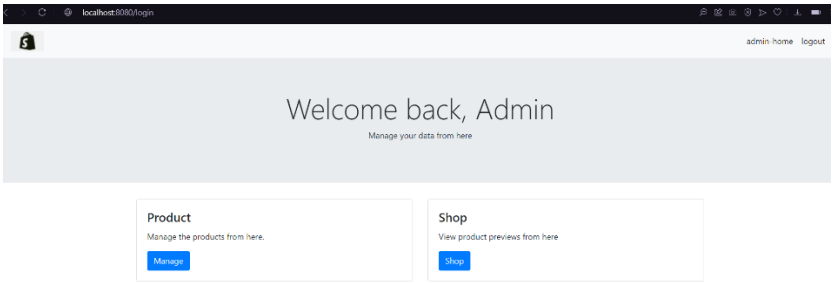


Above password is the special password that provide admin authority.

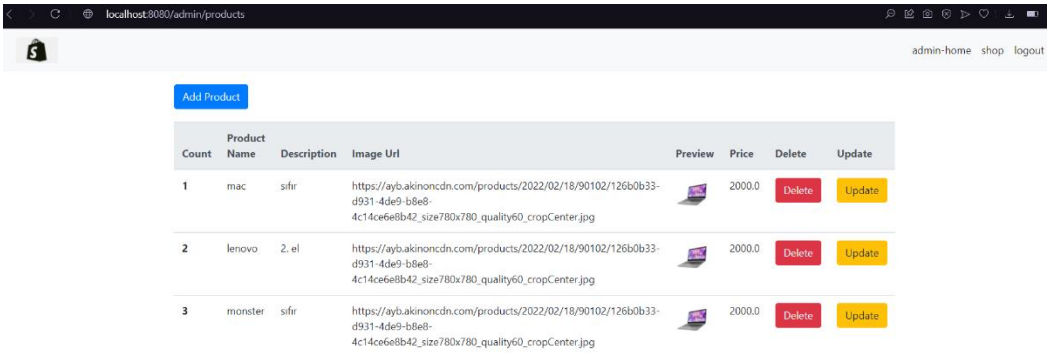
Login page: enter informations:



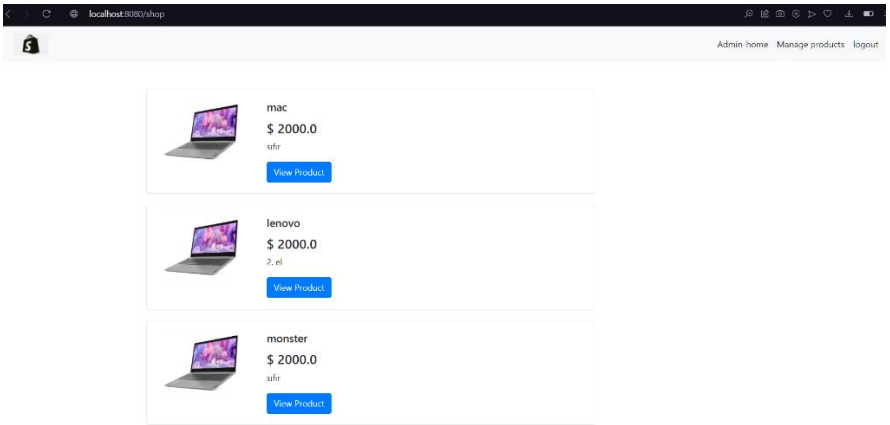
I am clicking to login and page redirect me to admin home page



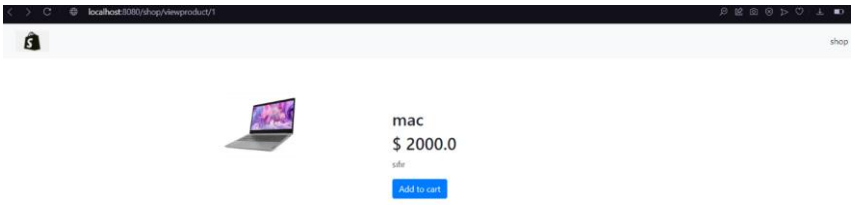
I am going to manage products, added product at the before testing are coming



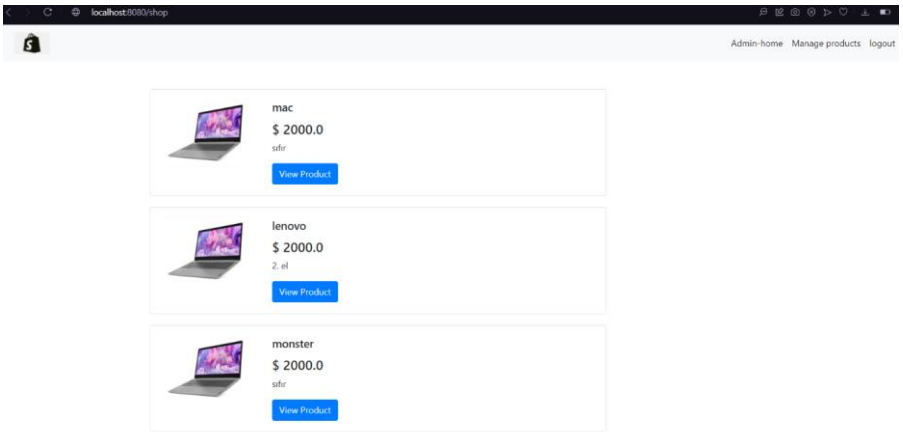
I am going to shop from upper right corner.



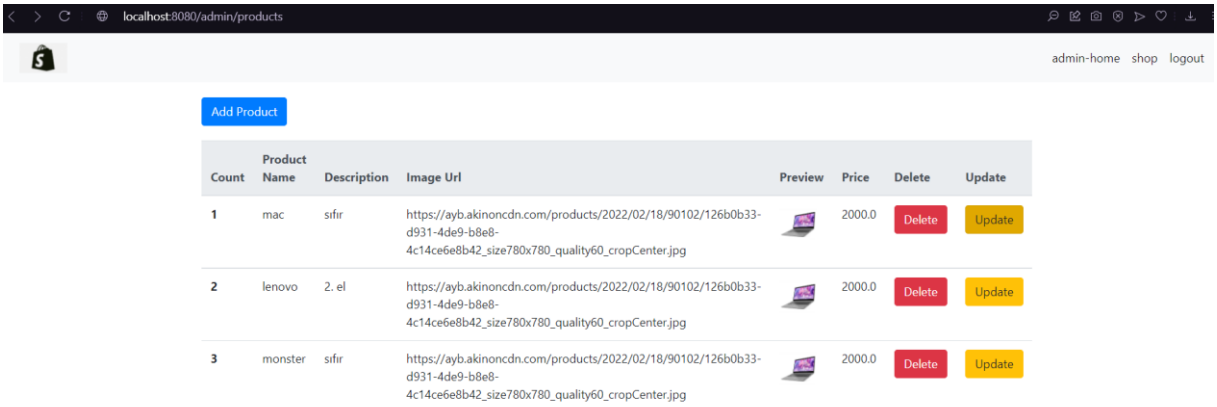
I am going to view product:



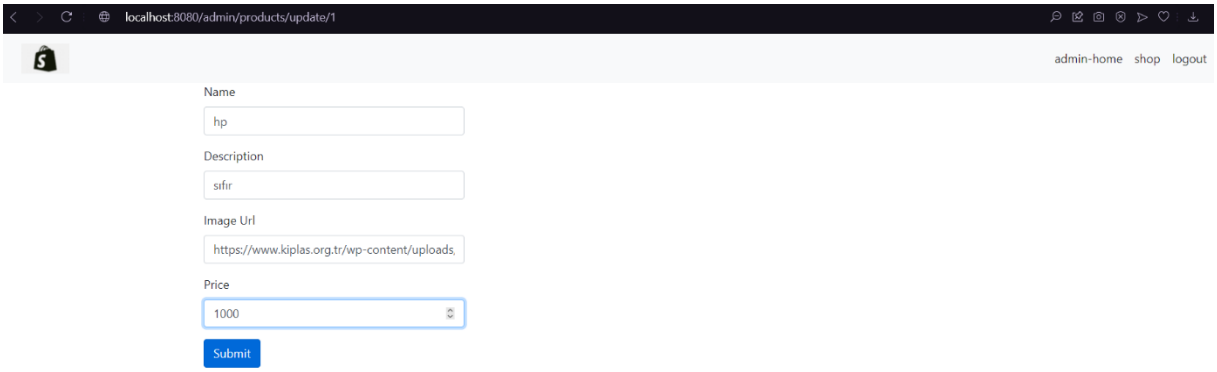
Add to cart end point is not implemented so that I am not click this button, I am turning shop



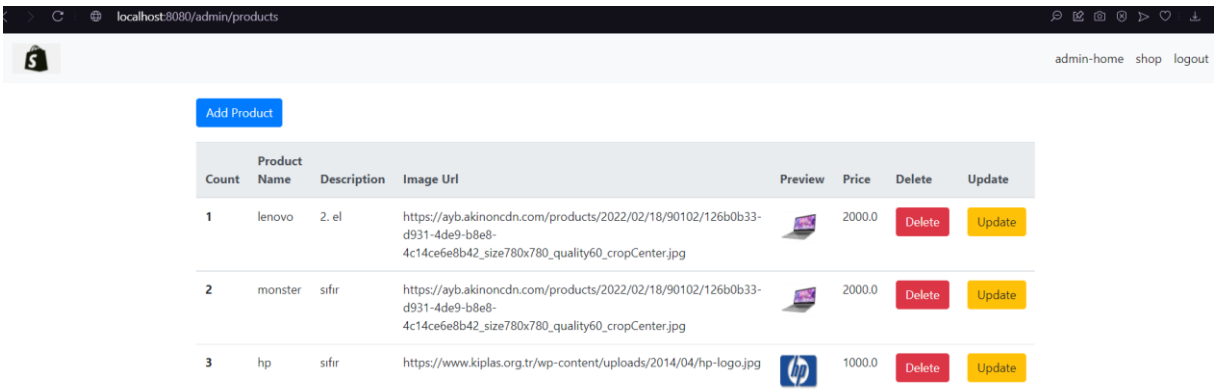
I am going to manage products from uppe right corner:



I am going to update any product:



Product image and informations are changed:



6. Maintenance Plan and Details

In order for this project to be used as a real project, some updates are required. Some of these, instead of providing admin authority with password, role based authority that comes in spring security dependency should be applied. Jwt token can be added for user authentication.

Another situation that needs to be fixed. When the project is run the first time, it works without errors. But when execute second times occur an error that does not affect and stop the application.

Examples

```
Console   Actuator
2022-06-23 03:32:57.974  INFO 18164 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.6.9.Final
2022-06-23 03:32:58.187  INFO 18164 --- [           main] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2022-06-23 03:32:58.309  INFO 18164 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2022-06-23 03:32:58.514  INFO 18164 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2022-06-23 03:32:58.536  INFO 18164 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.PostgreSQLDialect
Hibernate:

    alter table cart_table_
       drop constraint FKbg8qwt22i82vu4k6gkej2qhia
2022-06-23 03:32:59.434  WARN 18164 --- [           main] o.h.t.s.i.ExceptionHandlerLoggedImpl     : GenerationTarget encountered exception accepting command : Error executing DDL "
    alter table cart_table_
       drop constraint FKbg8qwt22i82vu4k6gkej2qhia" via JDBC Statement

org.hibernate.tool.schema.spi.CommandAcceptanceException: Error executing DDL "
    alter table cart_table_
       drop constraint FKbg8qwt22i82vu4k6gkej2qhia" via JDBC Statement <13 internal lines>
    at org.springframework.orm.jpa.vendor.SpringHibernateJpaPersistenceProvider.createContainerEntityManagerFactory(SpringHibernateJpaPersistenceProvider.java:58) ~[spring-orm-5.3.20.jar:5.3.20]
    at org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean.createNativeEntityManagerFactory(LocalContainerEntityManagerFactoryBean.java:365) ~[spring-orm-5.3.20.jar:5.3.20]
    at org.springframework.orm.jpa.AbstractEntityManagerFactoryBean.buildNativeEntityManagerFactory(AbstractEntityManagerFactoryBean.java:409) ~[spring-orm-5.3.20.jar:5.3.20]
    at org.springframework.orm.jpa.AbstractEntityManagerFactoryBean.afterPropertiesSet(AbstractEntityManagerFactoryBean.java:396) ~[spring-orm-5.3.20.jar:5.3.20]
    at org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean.afterPropertiesSet(LocalContainerEntityManagerFactoryBean.java:341) ~[spring-orm-5.3.20.jar:5.3.20]
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.invokeInitMethods(AbstractAutowireCapableBeanFactory.java:1863) ~[spring-beans-5.3.20.jar:5.3.20]
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.initializeBean(AbstractAutowireCapableBeanFactory.java:1800) ~[spring-beans-5.3.20.jar:5.3.20]
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:620) ~[spring-beans-5.3.20.jar:5.3.20]
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:542) ~[spring-beans-5.3.20.jar:5.3.20]
```

And the program continues to run. And the tables are created by hibernate.

```
SeniorProject01Application
Console   Actuator
    at org.postgresql.core.v3.QueryExecutorImpl.execute(QueryExecutorImpl.java:355) ~[postgresql-42.3.5.jar:42.3.5]
    at org.postgresql.jdbc.PgStatement.executeInternal(PgStatement.java:498) ~[postgresql-42.3.5.jar:42.3.5]
    at org.postgresql.jdbc.PgStatement.execute(PgStatement.java:408) ~[postgresql-42.3.5.jar:42.3.5]
    at org.postgresql.jdbc.PgStatement.executeWithFlags(PgStatement.java:329) ~[postgresql-42.3.5.jar:42.3.5]
    at org.postgresql.jdbc.PgStatement.executeCachedSql(PgStatement.java:315) ~[postgresql-42.3.5.jar:42.3.5]
    at org.postgresql.jdbc.PgStatement.executeWithFlags(PgStatement.java:291) ~[postgresql-42.3.5.jar:42.3.5]
    at org.postgresql.jdbc.PgStatement.execute(PgStatement.java:286) ~[postgresql-42.3.5.jar:42.3.5]
    at com.zaxxer.hikari.pool.ProxyStatement.execute(ProxyStatement.java:94) ~[HikariCP-4.0.3.jar:na]
    at com.zaxxer.hikari.pool.HikariProxyStatement.execute(HikariProxyStatement.java) ~[HikariCP-4.0.3.jar:na] <1 internal line>
    ... 35 common frames omitted

Hibernate:

    drop table if exists cart_table_ cascade
2022-06-23 03:32:59.441  WARN 18164 --- [           main] o.h.engine.jdbc.spi.SqlExceptionHelper   : SQL Warning Code: 0, SQLState: 00000
2022-06-23 03:32:59.441  WARN 18164 --- [           main] o.h.engine.jdbc.spi.SqlExceptionHelper   : table "cart_table_" does not exist, skipping
Hibernate:

    drop table if exists order_admin_table_ cascade
2022-06-23 03:32:59.443  WARN 18164 --- [           main] o.h.engine.jdbc.spi.SqlExceptionHelper   : SQL Warning Code: 0, SQLState: 00000
2022-06-23 03:32:59.444  WARN 18164 --- [           main] o.h.engine.jdbc.spi.SqlExceptionHelper   : table "order_admin_table_" does not exist, skipping
Hibernate:

    drop table if exists product_table_ cascade
2022-06-23 03:32:59.445  WARN 18164 --- [           main] o.h.engine.jdbc.spi.SqlExceptionHelper   : SQL Warning Code: 0, SQLState: 00000
2022-06-23 03:32:59.446  WARN 18164 --- [           main] o.h.engine.jdbc.spi.SqlExceptionHelper   : table "product_table_" does not exist, skipping
Hibernate:
```

I found solution for this.

When I replace the create-drop part with update in the application.properties file, the error disappears. The relevant part of the file I mentioned is below.

```
Product.java   UserController.java   UserService.java   application.properties
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
spring.datasource.username=postgres
spring.datasource.password=example
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.properties.hibernate.format_sql=true
```

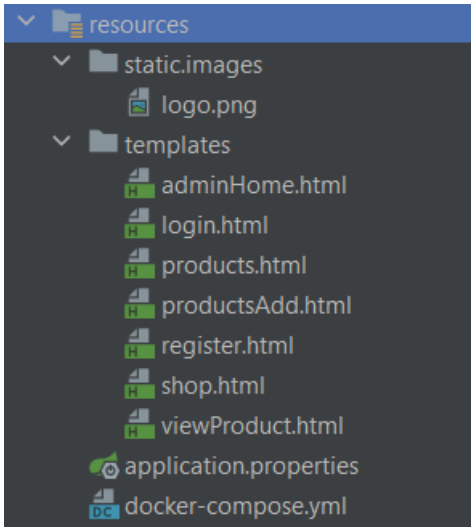
When I change this with update the console runs cleanly.

```
Console   Actuator

2022-06-23 04:23:47.081  INFO 8904 --- [           main] c.p.s.SeniorProject01Application         : Starting SeniorProject01Application using Java 18.0.1 on DESKTOP-VVIMOH3 with PID 8904 (C:\senior product home page
2022-06-23 04:23:47.086  INFO 8904 --- [           main] c.p.s.SeniorProject01Application         : No active profile set, falling back to 1 default profile: "default"
2022-06-23 04:23:48.395  INFO 8904 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2022-06-23 04:23:48.520  INFO 8904 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 107 ms. Found 4 JPA repository interfaces.
2022-06-23 04:23:50.630  INFO 8904 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2022-06-23 04:23:50.646  INFO 8904 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2022-06-23 04:23:50.647  INFO 8904 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.63]
2022-06-23 04:23:50.832  INFO 8904 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2022-06-23 04:23:50.832  INFO 8904 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 3636 ms
2022-06-23 04:23:50.997  INFO 8904 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2022-06-23 04:23:51.068  INFO 8904 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.6.9.Final
2022-06-23 04:23:51.268  INFO 8904 --- [           main] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2022-06-23 04:23:51.466  INFO 8904 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2022-06-23 04:23:51.639  INFO 8904 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2022-06-23 04:23:51.656  INFO 8904 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.PostgreSQLDialect
Hibernate:

    create table cart_table_ (
       cart_id_ int4 not null,
        quantity_ int4,
        product_id_ int4,
        user_id_ int4,
        primary key (cart_id_)
    )
Hibernate:

    create table order_admin_table_ (
       order_id_ int4 not null,
        address_ varchar(255) not null,
        email_ varchar(255) not null,
        name_ varchar(255) not null,
        order_summary_ varchar(255) not null,
        surname_ varchar(255) not null,
        total_cost_ float8 not null,
        primary key (order_id_)
    )
```

But when I run it with update for the 2nd time, the program crashes. For this reason
Update and create-drop should be changed each time. Or run with create-drop that produces errors but do not harm and affect the program.

7. Other Project Elements

In this section, we can talk a little about the frontend.



The logo image in Static.image creates the logo in the upper left corner of the frontend.

The html files in the templates are included to the project by showing in the reference section from another sample code. The backend and frontend of the project were made suitable by making the necessary additions and deletions at the frontend.

Where was Thymeleaf used?

```
products.html
        </div>
    </div>
</nav>
<div class="container">

    <a th:href="@{/admin/products/add}" style="margin: 20px 0" class="btn btn-primary">Add Product</a>
    <table class="table">
        <thead class="thead-light">
        <tr>
            <th scope="col">Count</th>
            <th scope="col">Product Name</th>
            <th scope="col">Description</th>
            <th scope="col">Image Url</th>
            <th scope="col">Preview</th>
            <th scope="col">Price</th>
            <th scope="col">Delete</th>
            <th scope="col">Update</th>
        </tr>
        </thead>
        <tbody>

        <tr th:each="product, iStat : ${products}">
            <th scope="row" th:text="${iStat.index + 1}">1</th>
            <td th:text="${product.name}"></td>
            <td th:text="${product.description}"></td>
            <td th:text="${product.imageUrl}"></td>
            <td>
                <img class="" height="50px" width="50px" th:src="@{${ product.imageUrl}}" >
            </td>
            <td th:text="${product.price}"></td>

            <td><a href="" th:href="@{/admin/products/delete/{id}(id=${product.productId})}" class="btn btn-danger">Delete</a></td>
            <td><a href="" th:href="@{/admin/products/update/{id}(id=${product.productId})}" class="btn btn-warning">Update</a></td>

        </tr>
        </tbody>
    </table>
</div>
```

In all lines containing "th", thmeleaf engine managed backend objects or object variables. Let's give an example

```
<p th:text="#{home.welcome}">Welcome to our grocery store!</p>
```

What we can see here are in fact two different features of the Thymeleaf Standard Dialect:

The th:text attribute, which evaluates its value expression and sets the result of this evaluation as the body of the tag it is in, effectively substituting that “Welcome to our grocery store!” text we see in the code. At the same time, to use thymelaf, the bottom 2nd line should always be added to the beginning of the html file like below.

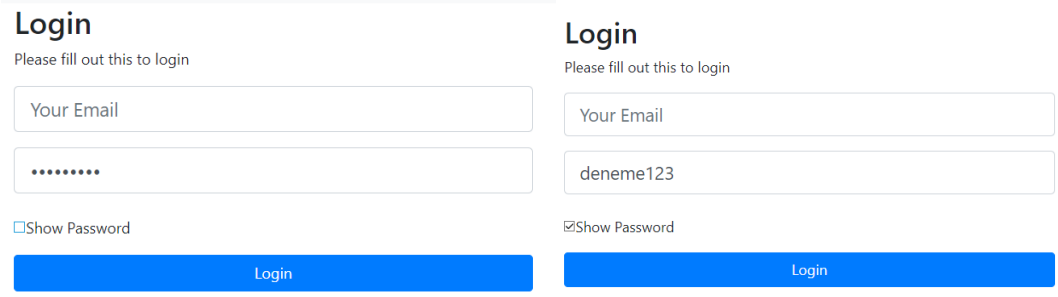
```
<!doctype html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
```

Where I use JavaScript?:

I used it to hide and show password on login and register pages. Below is an example.

```
<div class="form-group">
   <input type="password" class="form-control form-control-lg" placeholder="Password"
name="password" id="password">
   <br>
   <input type="checkbox" onclick="myFunction()">Show Password
</div>
```

```
<script>
   function myFunction() {
      var passwordArea = document.getElementById("password");
      if (passwordArea.type === "password") {
         passwordArea.type = "text";
      } else {
         passwordArea.type = "password";
      }
   }
</script>
```



7.1 Ethics and Professional Responsibilities

I did the project by myself without any help. All the resources that I used and while developing the project are shown in the reference section.

8. New Knowledge Acquired and Applied

I learned to use Docker while developing the project. At the same time, I learned to connect frontend and the backend with Thymeleaf engine. I also learned a little bit VueJs for frontend. However, I changed this idea with thymleaf engine because I got many errors during the implementation phase.

9. Conclusion and Future Work

I gained experience on how to develop an e-commerce project. If I develop this project again, I think I will become more professional.

10. References

https://spring.io/

https://start.spring.io/

https://www.baeldung.com/spring-angular-ecommerce

https://github.com/tburakdirlik/Programming-Languages-Notes/tree/main/Programming%20Notes/Java/SPRING

https://github.com/tburakdirlik/Programming-Languages-Notes/tree/main/Programming%20Notes/Java/OPTIONALS

https://github.com/Adarsh-gupta/Adarsh-gupta-spring-major-starter

https://www.baeldung.com/thymeleaf-in-spring-mvc

https://www.baeldung.com/building-a-restful-web-service-with-spring-and-java-based-configuration