Online Shop Application

Author: Bodea Răzvan-Marius

Team: Bodea Răzvan-Marius

Haragâș Alexandru

Bărăgan Andrei

Group: English_30432

Year: 2022

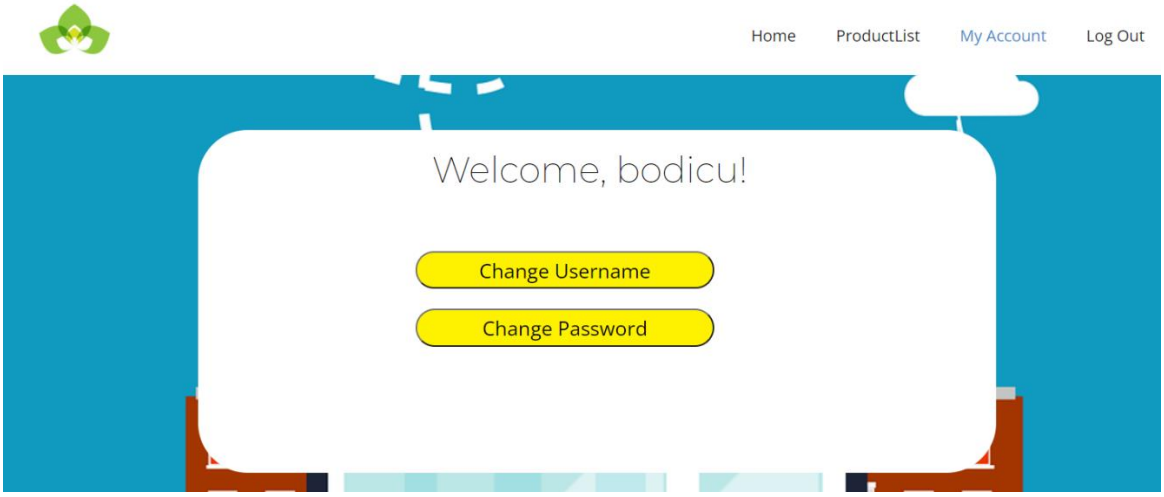
TABLE OF CONTENTS

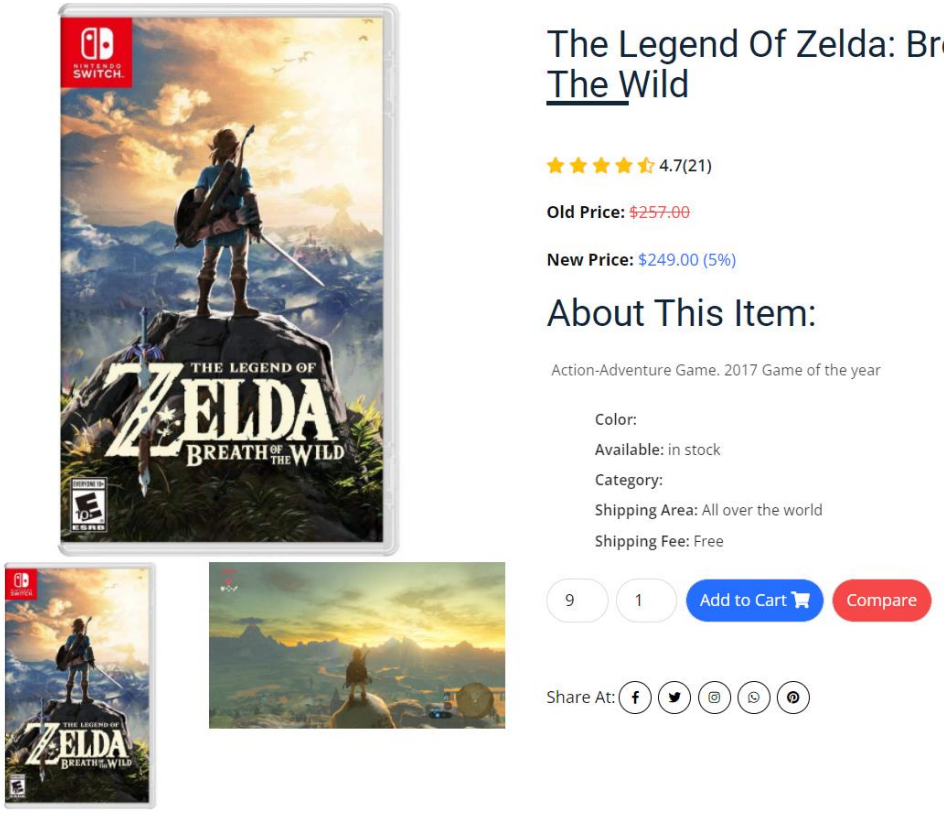
1. Assignment Objective & Abstract

Design an online shopping application that allows multiple users to login based on their accounts and see a list of products and order products. The workers/admins should be able to handle some specific operations for the online shop.

Sub-Objectives:

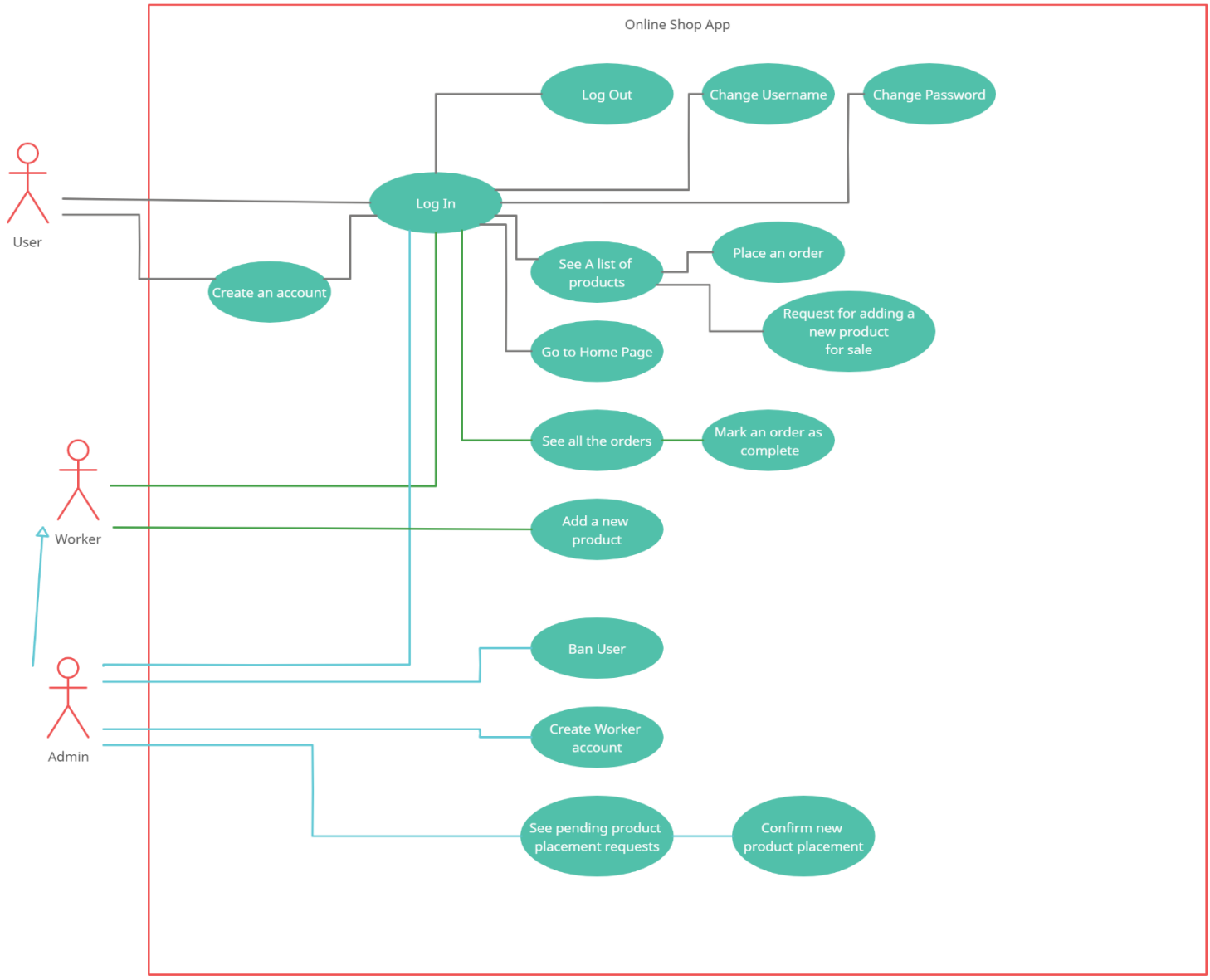
- The app should work on a phone
- Java & Spring application
- Implement the app using design patterns & protocols
- Test the application





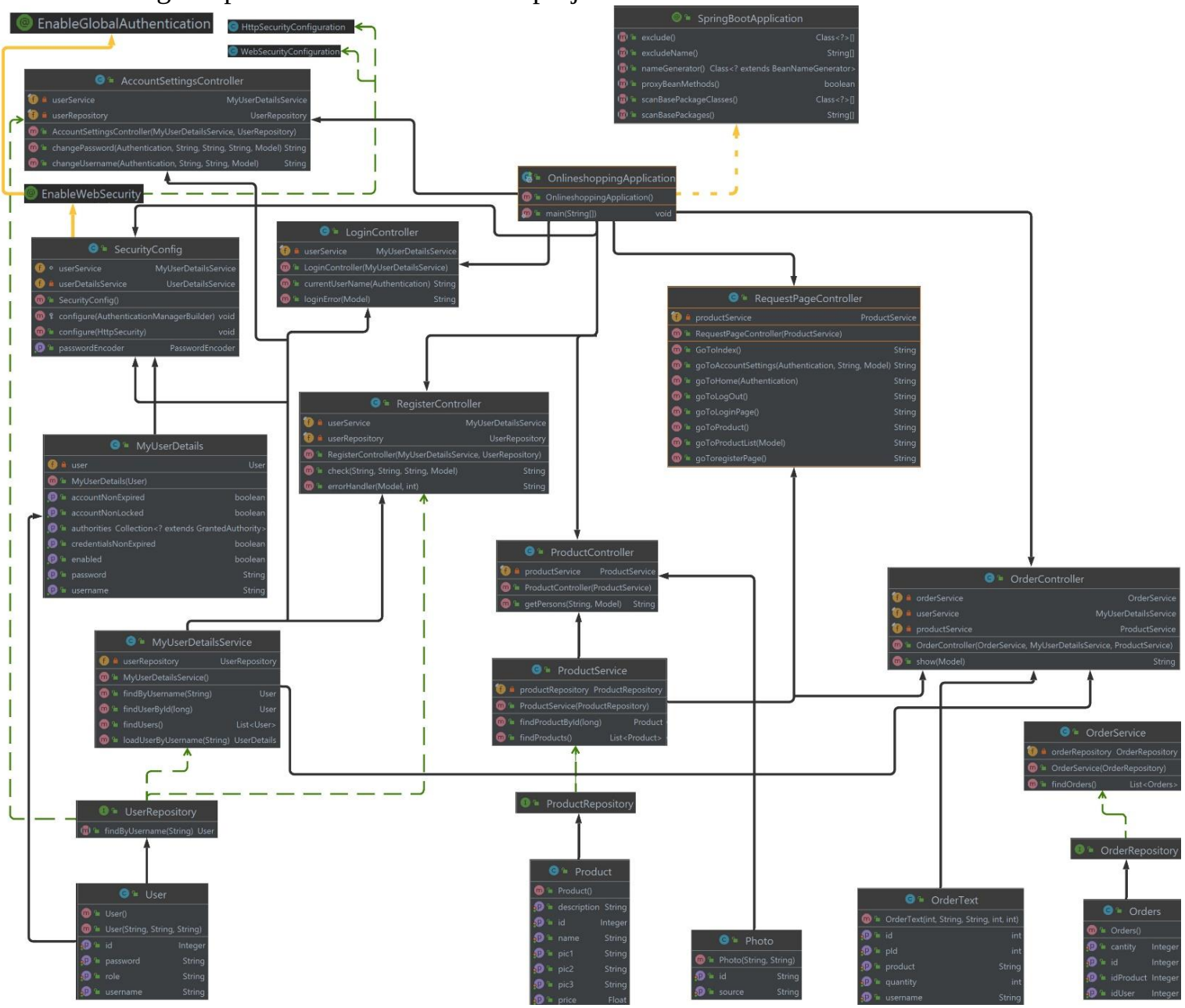
2. Use case diagram

The use case diagram presents a normal(expected) usage of the app and its main functionalities.



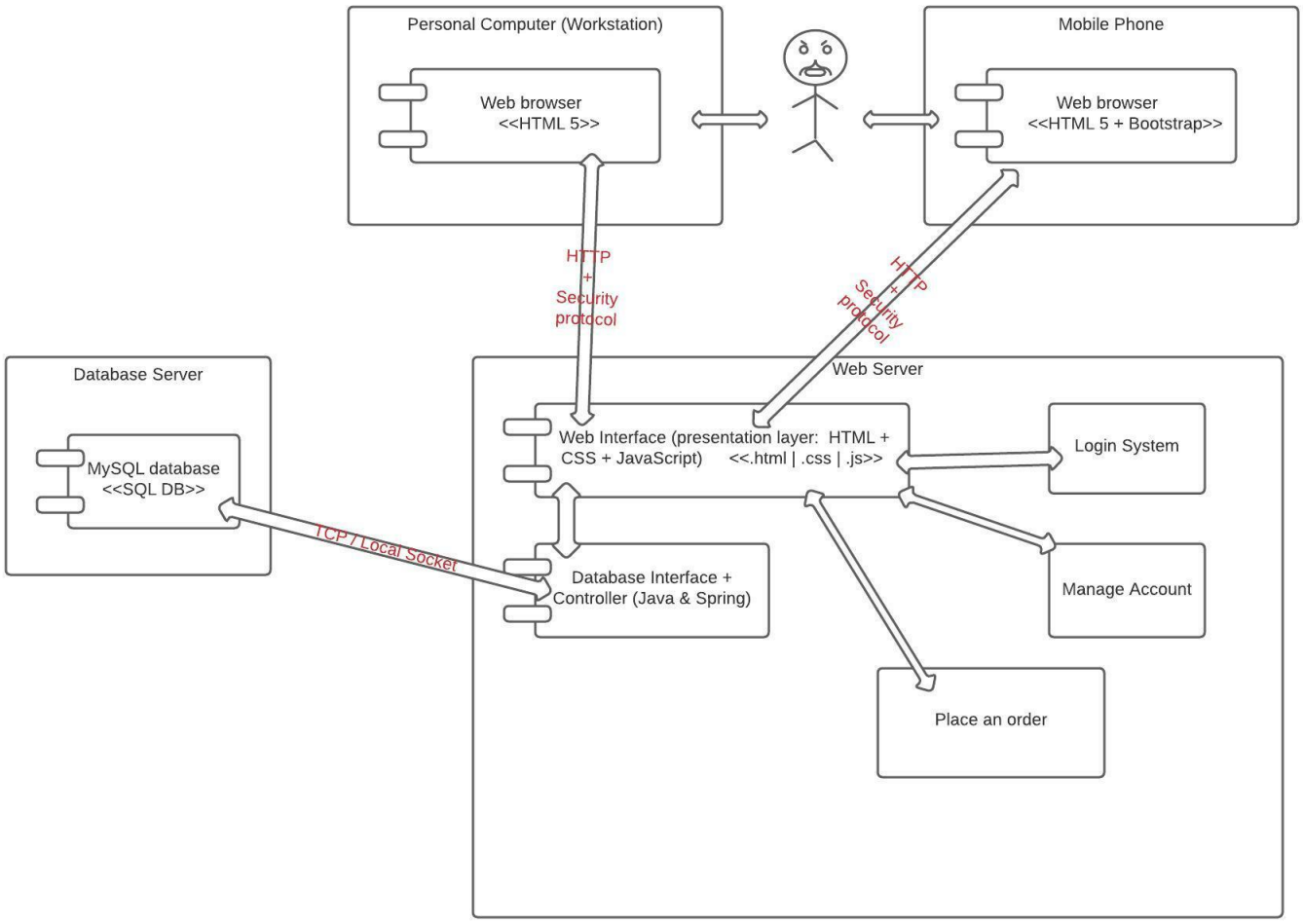
3. UML Diagram

The UML diagram presents the classes of the project and the interactions between them.

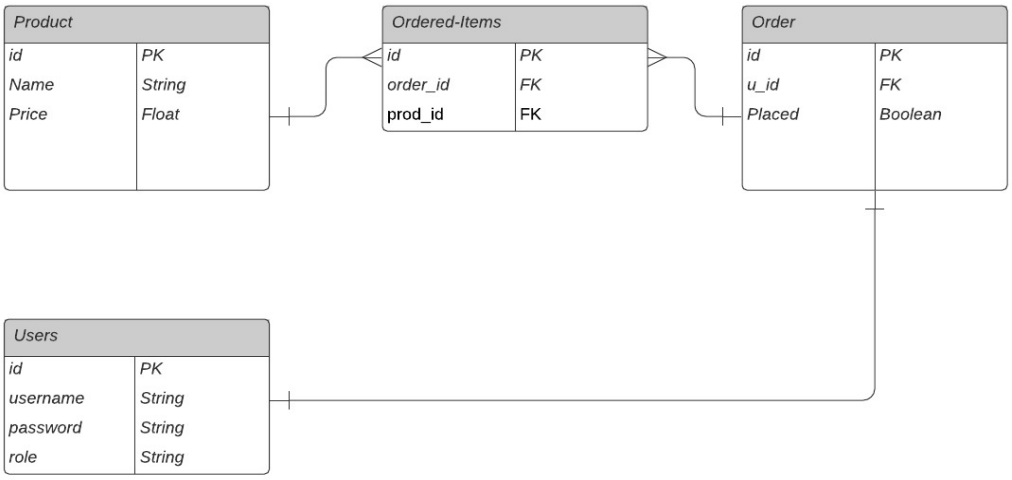


4. Deployment diagram

The Deployment Diagram of the app presents a visualization of the hardware and the links of communication between it and the software part of the project.

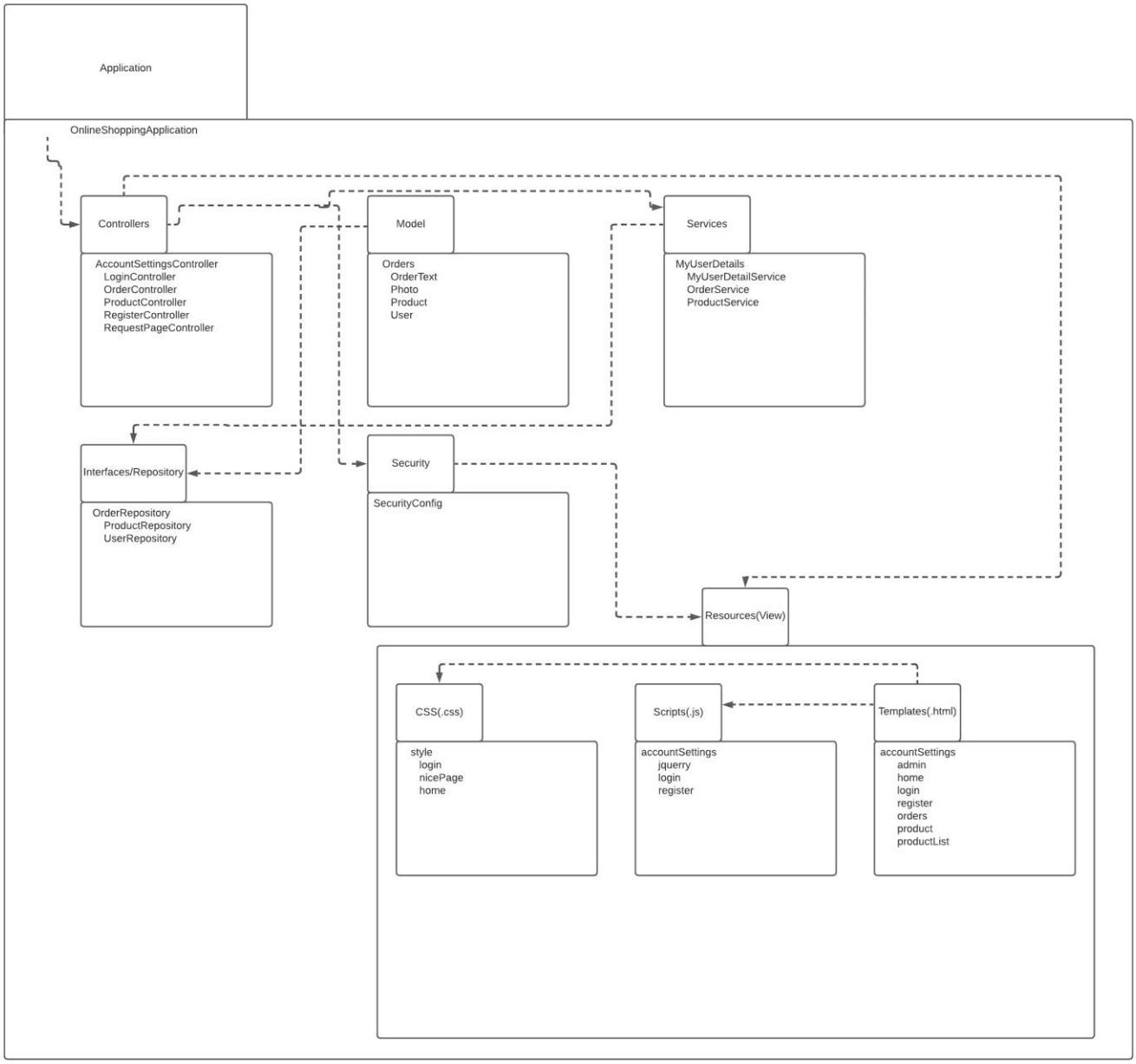


5. Database Diagram



6. Design patterns & protocols

(Package diagram)



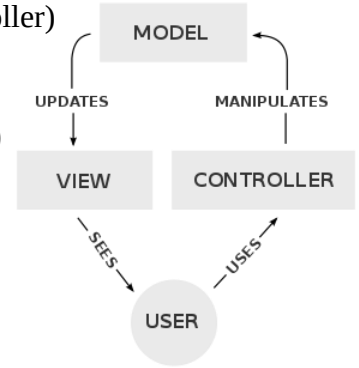
1.MVC

The design pattern at the base of this project is the MVC (model-view-controller) design pattern.

The model works with the logical operations.

The view contains the graphical representation of the project (HTML + CSS)

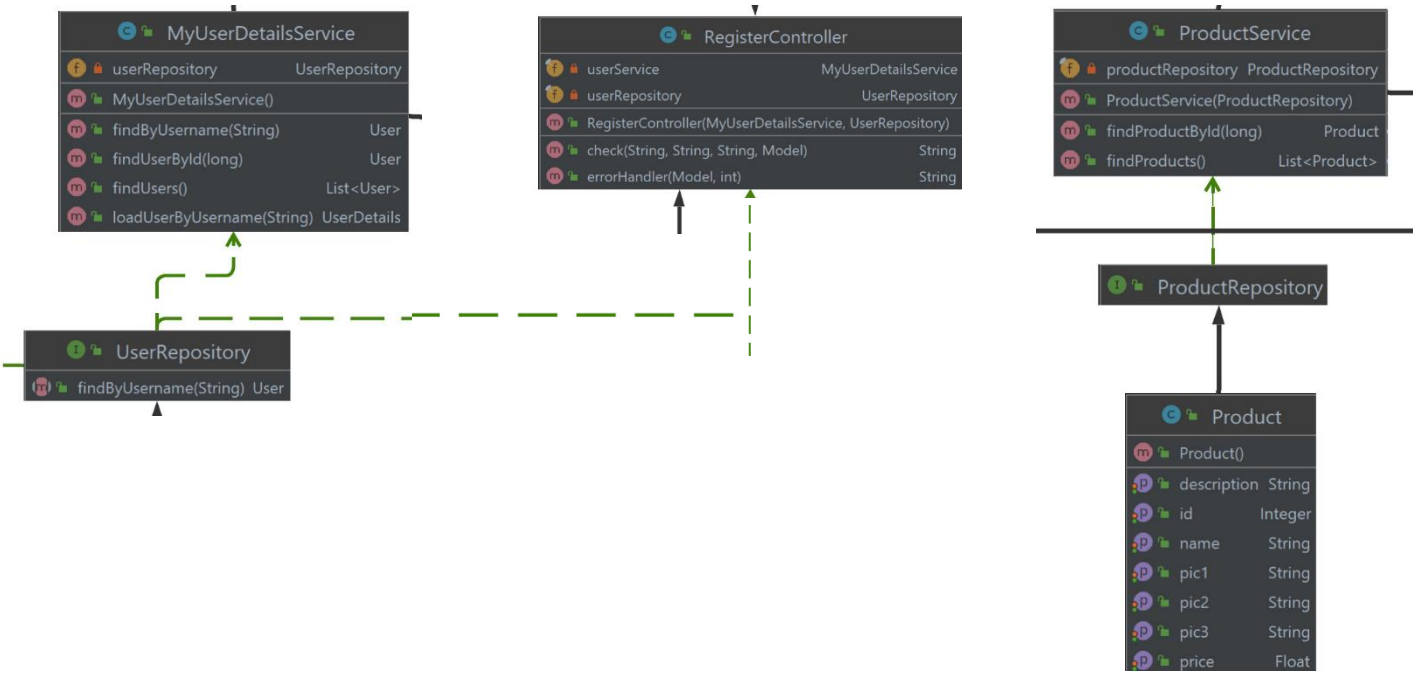
The controller handles the access to the app and the different functions of the app.



2.Bridge

The bridge design pattern is used to decouple an abstraction from its implementation so that the two can vary independently.

In our project, the bridge pattern is used for an easier work with the database data and with the mobile part of the app.



7. My contribution to the project

My main contribution to the project was related to the login, register and security of these actions.

The login & register protocols:

```
protocol login { //Client side
   begin.
   ![
      !<String> //Send Username
      !<String> //Send Password
   ]*.
   !{ //Select one of
   SUCCESS:  ?(Model (HTML home page)) //go to the home page of the app
   FAILURE:  ?(Model (HTML login page)).?<String> //stay on login page and get error message
   }
}
protocol register {  //Client side
   begin.
   ![
      !<String> //Send Username
      !<String> //Send Password1
      !<String> //Send Password2
   ]*.
   !{ //Select one of
   SUCCESS:  ?(Model (HTML login page)).?<String> //go to login page and recieve a success message
   FAILURE:  ?(Model (HTML register page)).?<String> //stay on register page and recieve an error
message
   }
}
```

Session: for our project, the session is used in order to keep track of the users that are using the site (we need to know all the users that are on the site at a specific point and be able to differentiate between them because every user should be able to place an order)

The security aspect of the app handles this necessity by implementing a User Authentication method. The authentication information can be passed between the server and the client, thus providing a way to identify the clients thar are logged on.

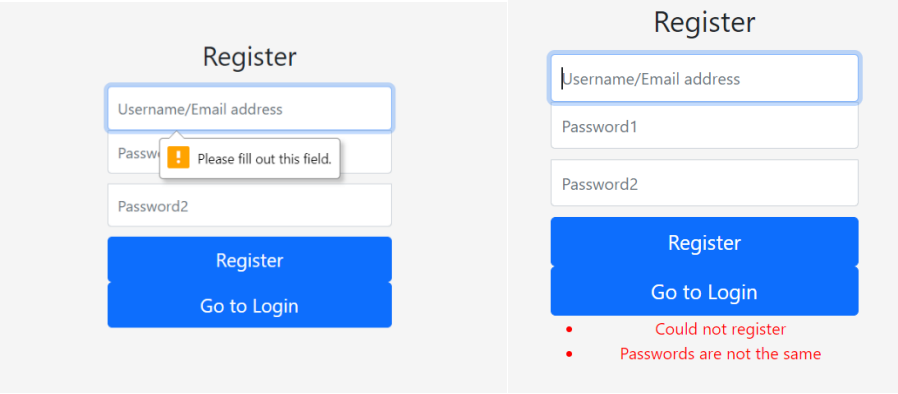
At the end of a session, the client will log off.

A user is allowed to enter the home page of the app and access the features of the app, only if he logs in. The user has the option to create a new account.

```
public void configure(final HttpSecurity http) throws Exception {
    http
            .csrf().csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())
            .and()
            .authorizeRequests()
            .antMatchers("/public/**").permitAll()
            .antMatchers("/actuator/health").permitAll()
            .antMatchers("/swagger-ui/**").permitAll()
            .antMatchers("/swagger-resources/**").permitAll()
            .antMatchers("/webjars/**").permitAll()
```

```
            .antMatchers("/home").permitAll()
            .antMatchers("/register", "/registerCheck", "/register-Error").permitAll()
            .anyRequest()
            .authenticated()
            .and()
            .formLogin()
            .loginPage("/login")
            .defaultSuccessUrl("/accountSettings")
            .failureUrl("/login-error")
            .permitAll()
            .and()
            .logout()
            .logoutUrl("/logout")
            .permitAll();
}
```

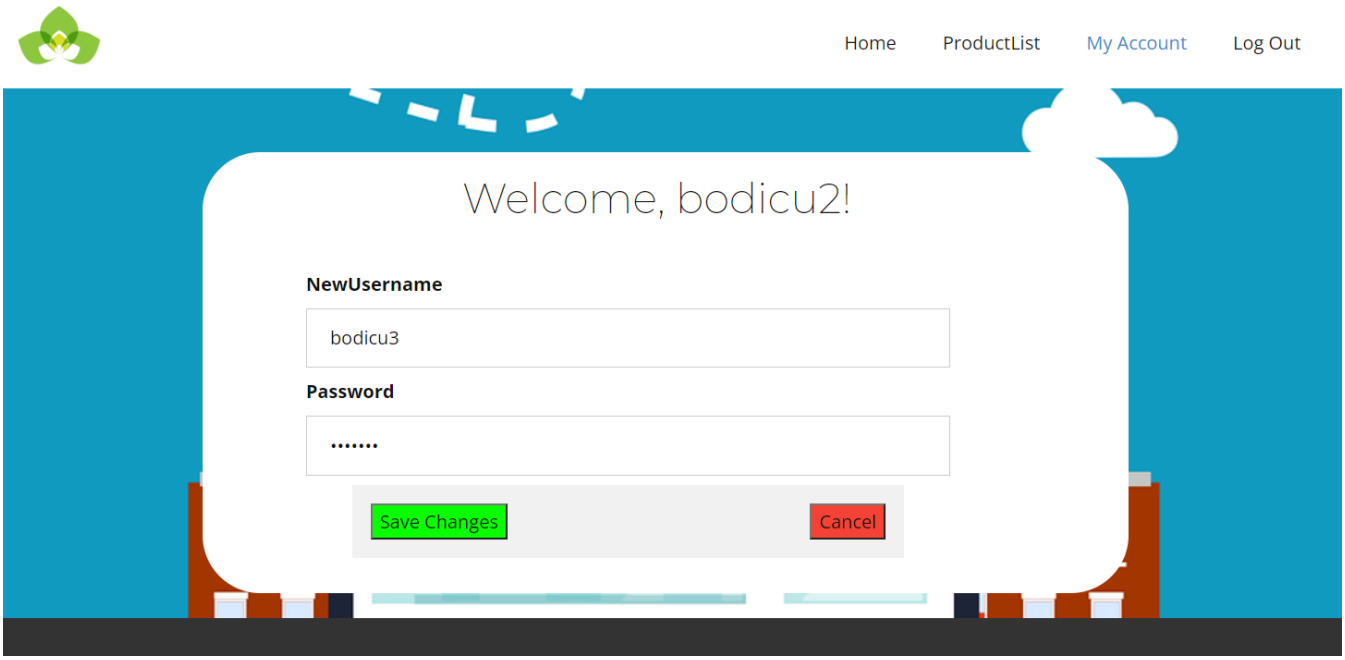
The Register and Login functions check if the account entered is a valid one and displays possible error/success messages depending on the input.

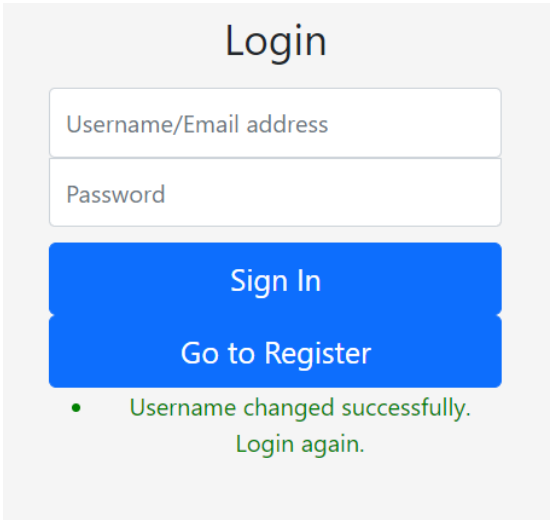


The user is also allowed to change details about his account (password or username)

```
@RequestMapping("/changePassword")
public String changePassword(Authentication authentication, @ModelAttribute("password2")
String password, @ModelAttribute("newpassword1") String newPass1,
@ModelAttribute("newpassword2") String newPass2, final Model model){
    User u = userService.findByUsername(authentication.getName());
    if(password.equals(u.getPassword())){
        if(newPass1.equals(newPass2) && !newPass1.isEmpty()){
            u.setPassword(newPass1);
            userRepository.save(u);
            model.addAttribute("changePasswordSuccess",true);
            return "login";
        }else{
            if(newPass1.isEmpty() || newPass2.isEmpty()){
                model.addAttribute("emptyPassword",true);
                return "accountSettings";
            }
            model.addAttribute("differentPasswords",true);
            return "accountSettings";
        }
    }else{
        model.addAttribute("wrongPassword",true);
        return "accountSettings";
    }
}
```

```java
@RequestMapping("/changeUsername")
public String changeUsername(Authentication authentication, @ModelAttribute("username")
String newUsername, @ModelAttribute("password") String password, final Model model){
    User u = userService.findByUsername(authentication.getName());
    if(newUsername.isEmpty()){
        model.addAttribute("emptyPassword",true);
        return "accountSettings";
    }
    if(password.equals(u.getPassword())){
        u.setUsername(newUsername);
        userRepository.save(u);
        model.addAttribute("changeUserSuccess",true);
        return "login";
    }
    model.addAttribute("wrongPassword",true);
    return "accountSettings";
}
```





8. Code Section

The whole code for the project can be found at https://github.com/baragan30/OnlineShop

AccountSettingsController: the class that handles the editing of the information of an account, by the user of that account.

```
package com.onlineshopping.controllers;

import com.onlineshopping.model.User;
import com.onlineshopping.repositories.UserRepository;
import com.onlineshopping.services.MyUserDetailsService;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class AccountSettingsController {
    private final MyUserDetailsService userService;
    private final UserRepository userRepository;

    public AccountSettingsController(MyUserDetailsService userService, UserRepository
userRepository) {
        this.userService = userService;
        this.userRepository = userRepository;
    }

    @RequestMapping("/changeUsername")
    public String changeUsername(Authentication authentication,
@ModelAttribute("username") String newUsername, @ModelAttribute("password") String
password, final Model model){
        User u = userService.findByUsername(authentication.getName());
        if(newUsername.isEmpty()){
            model.addAttribute("emptyPassword",true);
            return "accountSettings";
        }
        if(password.equals(u.getPassword())){
            u.setUsername(newUsername);
            userRepository.save(u);
            model.addAttribute("changeUserSuccess",true);
            return "login";
        }
        model.addAttribute("wrongPassword",true);
        return "accountSettings";
    }

    @RequestMapping("/changePassword")
    public String changePassword(Authentication authentication,
@ModelAttribute("password2") String password, @ModelAttribute("newpassword1") String
newPass1, @ModelAttribute("newpassword2") String newPass2, final Model model){
        User u = userService.findByUsername(authentication.getName());
        if(password.equals(u.getPassword())){
            if(newPass1.equals(newPass2) && !newPass1.isEmpty()){
                u.setPassword(newPass1);
                userRepository.save(u);
                model.addAttribute("changePasswordSuccess",true);
                return "login";
            }else{
                if(newPass1.isEmpty() || newPass2.isEmpty()){
                    model.addAttribute("emptyPassword",true);
```

```
                    return "accountSettings";
                }
                model.addAttribute("differentPasswords",true);
                return "accountSettings";
            }
        }else{
            model.addAttribute("wrongPassword",true);
            return "accountSettings";
        }
    }
}
```

RegisterController: handles the operation of creating a new account and adding it to the database (if it doesn t exist already). The data entered is also verified in order to create a valid account.

```
package com.onlineshopping.controllers;

import com.onlineshopping.model.User;
import com.onlineshopping.repositories.UserRepository;
import com.onlineshopping.services.MyUserDetailsService;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class RegisterController {
    private final MyUserDetailsService userService;
    private final UserRepository userRepository;

    public RegisterController(MyUserDetailsService userService, UserRepository
userRepository) {
        this.userService = userService;
        this.userRepository = userRepository;
    }

    @RequestMapping("/registerCheck")
    public String check(@ModelAttribute("username") String username,
@ModelAttribute("password1") String password1, @ModelAttribute("password2") String
password2, final Model model){
        if(password1.equals(password2)){
            User tmp = userService.findByUsername(username);
            if(tmp == null){
                User u = new User(username,password1,"client");
                userRepository.save(u);
                return "login";
            }else {
                errorHandler(model,1);
            }
        }else{
            errorHandler(model,2);
        }
        return errorHandler(model,0);
    }

    @RequestMapping("/register-Error")
    public String errorHandler(final Model model, int errorNr){
        model.addAttribute("registerError",true);
        if (errorNr==1)
            model.addAttribute("alreadyExists", true);
        if (errorNr==2)
            model.addAttribute("notSamePasswords", true);
        return "register";
```

```
    }

}
```

RequstPageController: Handles the transition between pages of the app.

```
package com.onlineshopping.controllers;

import org.springframework.security.core.Authentication;
import com.onlineshopping.model.Product;
import com.onlineshopping.services.ProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

import java.util.ArrayList;

@Controller
public class RequestPageController {
    private final ProductService productService;
    @Autowired
    public RequestPageController(ProductService productService) {
        this.productService = productService;
    }

    @RequestMapping("/login")
    public String goToLoginPage() {
        return "login";
    }

    @RequestMapping("/register")
    public String goToregisterPage() {
        return "register";
    }

    @RequestMapping("/")
    public String GoToIndex() {
        return "/home";
    }

    @RequestMapping("/home")
    public String goToHome(Authentication authentication) {
        if(authentication!=null){
            return "home";
        }
        else{
            return "homeUnlogged";
        }
    }

    @RequestMapping("/accountSettings")
    public String goToAccountSettings(Authentication authentication,
@RequestParam(name="username", required=false, defaultValue="World") String name, Model
model) {
        model.addAttribute("username", authentication.getName());
        return "accountSettings";
    }

    @RequestMapping("/logOut")
    public String goToLogOut() {
        return "logOut";
```

```
    }

    @RequestMapping("/product")
    public String goToProduct() {
        return "product";
    }

    @RequestMapping("/productList")
    public String goToProductList(Model model) {
        ArrayList<Product> p= new ArrayList<Product>(productService.findProducts());
        model.addAttribute("products",p);
        return "productList";
    }
}
```

The following classes are entities of the database mapped as objects, in order to have a way of representing them inside the Java application.
Orders:

```
@Entity
@Table (name = "orders")
public class Orders {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;

    @Column(name = "idUser")
    private Integer idUser;
    private Integer idProduct;
    private Integer cantity;

    public Integer getId(){
        return id;
    }

    public void setId(Integer id){
        this.id=id;
    }

    public Integer getIdUser() {
        return idUser;
    }

    public void setIdUser(Integer user) {
        this.idUser = user;
    }

    public Integer getIdProduct() {
        return idProduct;
    }

    public void setIdProduct(Integer idProduct) {
        this.idProduct = idProduct;
    }

    public Integer getCantity() {
        return cantity;
    }

    public void setCantity(Integer cantity) {
        this.cantity = cantity;
```

```
    }
}
```

Product:

```
@Entity
@Table(name = "Products")
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;
    private String name;
    private String description;
    private Float price;
    private String pic1;
    private String pic2;
    private String pic3;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {return description;}

    public void setDescription(String description) {this.description = description;}

    public Float getPrice() {
        return price;
    }

    public void setPrice(Float price) {
        this.price = price;
    }

    public String getPic1() {return pic1;}

    public void setPic1(String pic1) {this.pic1 = pic1;}

    public String getPic2() {return pic2;}

    public void setPic2(String pic2) {this.pic2 = pic2;}

    public String getPic3() { return pic3;}

    public void setPic3(String pic3) { this.pic3 = pic3;}
}
```

User:

```java
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    private String username;
    private String password;
    private String role;

    //Constructor for saving a new user to the database
    public User(String username, String password, String role) {
        this.username = username;
        this.password = password;
        this.role = role;
    }

    public User() {

    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getRole() {
        return role;
    }

    public void setRole(String role) {
        this.role = role;
    }
}
```

AccountSettings.js: javascript code for handling events at the browser level

```javascript
const defaultScene = document.getElementById("defaultScene").innerHTML;
const usernameScene = document.getElementById("usernameScene").innerHTML;
const passwordScene = document.getElementById("passwordScene").innerHTML;

const mainDiv = document.getElementById('mainDiv');
let changePasswordButton = null;
```

```
let changeUsernameButton = null;
let cancelButton = null;
uploadDefaultScene();

function initialize(state){
    switch(state){
        case 0 :
            changePasswordButton = document.getElementById('changePasswordButton');
            changeUsernameButton = document.getElementById("changeUsernameButton")
            changeUsernameButton.addEventListener('click',uploadChangeUsernameScene);
            changePasswordButton.addEventListener('click',uploadChangePasswordScene);
            break;
        case 1 :
            // let saveButton = document.getElementById("saveButton");
            cancelButton = document.getElementById("cancelButton1");
            cancelButton.addEventListener('click',uploadDefaultScene);
            break;
        case 2 :
            // let saveButton = document.getElementById("saveButton");
            cancelButton = document.getElementById("cancelButton2");
            cancelButton.addEventListener('click',uploadDefaultScene);
            break;
    }
}

function uploadDefaultScene(){
    mainDiv.innerHTML = defaultScene;
    initialize(0);
}
function uploadChangeUsernameScene(){
    mainDiv.innerHTML = usernameScene;
    initialize(1);
}
function uploadChangePasswordScene(){
    mainDiv.innerHTML = passwordScene;
    initialize(2);
}
```

```
const imgs = document.querySelectorAll('.img-select a');
const imgBtns = [...imgs];
let imgId = 1;

imgBtns.forEach((imgItem) => {
    imgItem.addEventListener('click', (event) => {
        event.preventDefault();
        imgId = imgItem.dataset.id;
        slideImage();
    });
});

function slideImage(){
    const displayWidth = document.querySelector('.img-showcase img:first-
child').clientWidth;

    document.querySelector('.img-showcase').style.transform = `translateX(${- (imgId - 1)
* displayWidth}px)`;
}

window.addEventListener('resize', slideImage);
```

The following classes are using Interfaces as a bridge for handling the working with database entities.

MyUserDetails:

```
public class MyUserDetails implements UserDetails {
    private User user;
    public MyUserDetails(User user) {
        this.user = user;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return Arrays.asList(new SimpleGrantedAuthority("ROLE_USER"));
    }

    @Override
    public String getPassword() {
        return user.getPassword();
    }

    @Override
    public String getUsername() {
        return user.getUsername();
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
    //...
}
```

MyUserDetailsService

```
@Service
public class MyUserDetailsService implements UserDetailsService {
    @Autowired
    private UserRepository userRepository;


    public List<User> findUsers(){
        return userRepository.findAll();
    }

    public User findUserById(long id){
        return userRepository.getById((int) id);
    }

    public User findByUsername(String username){return
```

```
userRepository.findByUsername(username);}

    @Override
    public UserDetails loadUserByUsername(String username) {
        User user = userRepository.findByUsername(username);
        if (user == null) {
            throw new UsernameNotFoundException(username);
        }
        return new MyUserDetails(user);
    }
}
```

9. Bibliography

--saas book (browser and mobile pg.27 pg.154)
http://www.saasbook.info/
--session types
http://www.simonjf.com/2016/05/28/session-type-implementations.html
--prism case studies (ideas for communication and security protocols)
https://www.prismmodelchecker.org/casestudies/
--for help and examples related to Java & Spring
https://www.baeldung.com/
--design patterns in java
https://www.tutorialspoint.com/design_pattern