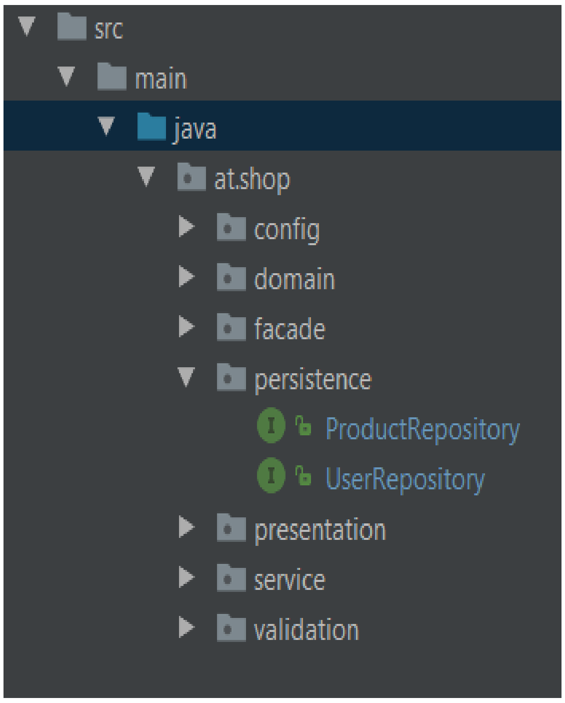
Introduction:

The desired learning outcome was defined with the words: “Design and implement a prototype dynamic web site incorporating database access. “The website I created was meant to be a web shop for selling products to customers. In the end, it would have been out of scope due to the complexity, so as an alternative the website was changed to work as a platform for downloading pictures of the products (for logged in users/employees/admin only). The website provides a Login, Registration and Edit Data function for every user and employees can additionally use the CRUD operations on the products while the admin even has the CRUD operation privilege on the users. A normal user can only see the products and download a specific one and has no rights to change the product data.

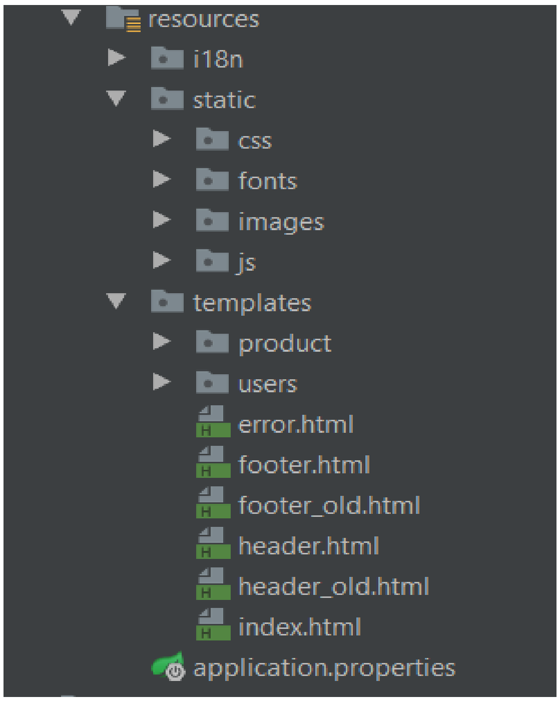
Technology:

Spring Boot + Tomcat Server	Open source framework for java application development
Thymeleaf + Thymeleaf security	Template engine for HTML5 and more
Spring Security	Authentication, authorization and more
Bootstrap	Open CSS framework
Maven	Build-management tool
MySQL	Relational database
JPA	Java specification data functions between Java objects and a relational database
Spring Devtools	Improves the development-time experience
Lombok	Reduce “boilerplate” code in the project

Overview:



```
▼ src
  ▼ main
    ▼ java
      ▼ at.shop
        ▶ config
        ▶ domain
        ▶ facade
        ▼ persistence
            ProductRepository
            UserRepository
        ▶ presentation
        ▶ service
        ▶ validation
```



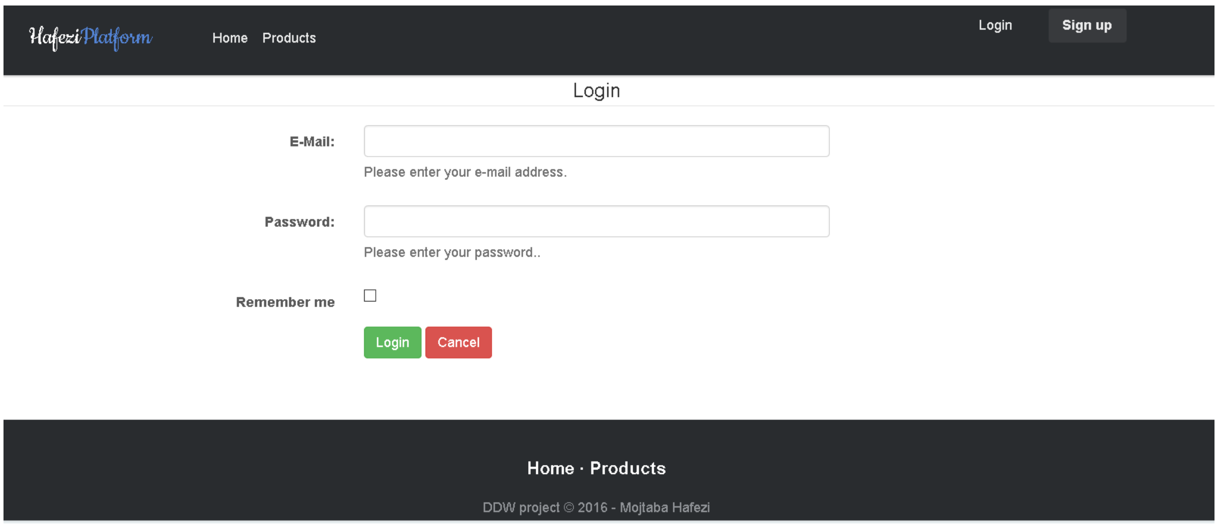
```
▼ resources
  ▶ i18n
  ▼ static
    ▶ css
    ▶ fonts
    ▶ images
    ▶ js
  ▼ templates
    ▶ product
    ▶ users
      error.html
      footer.html
      footer_old.html
      header.html
      header_old.html
      index.html
  application.properties
```

On the left side, the java directories can be seen with the architecture used. On the right side, the resources directories are shown. The façade pattern architecture is used, so any changes on the business logic are done in the façade classes. The domain layer consists of the user and product entities, the repositories use JPA to provide all needed functions between the plain Java objects and the relational database. The services are the interfaces between the façade and the repositories by providing the CRUD operations. In the presentation layer the controllers exist to use the façades to provide the corresponding views by operating in the way needed. The configurations for the web functions and the Thymeleaf engine are made in the config directory. As an additional feature validators were written to provide a wider control on the validation.

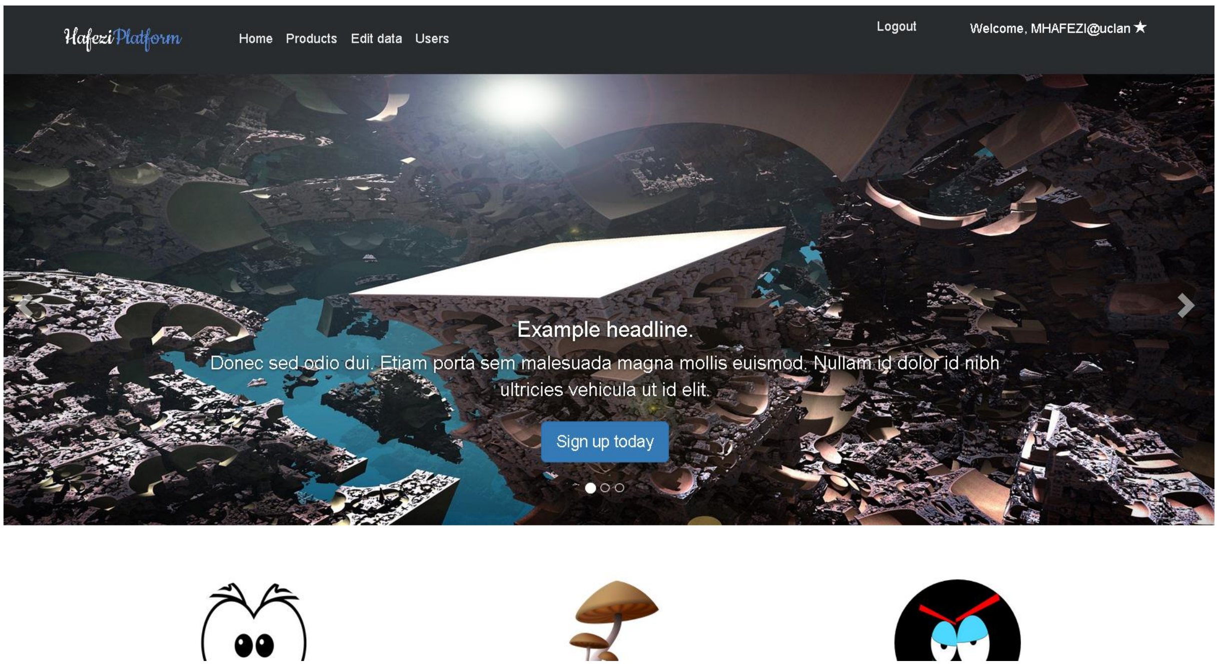
The resources are self-explanatory as they provide the resources needed to present the website.

Requirements:

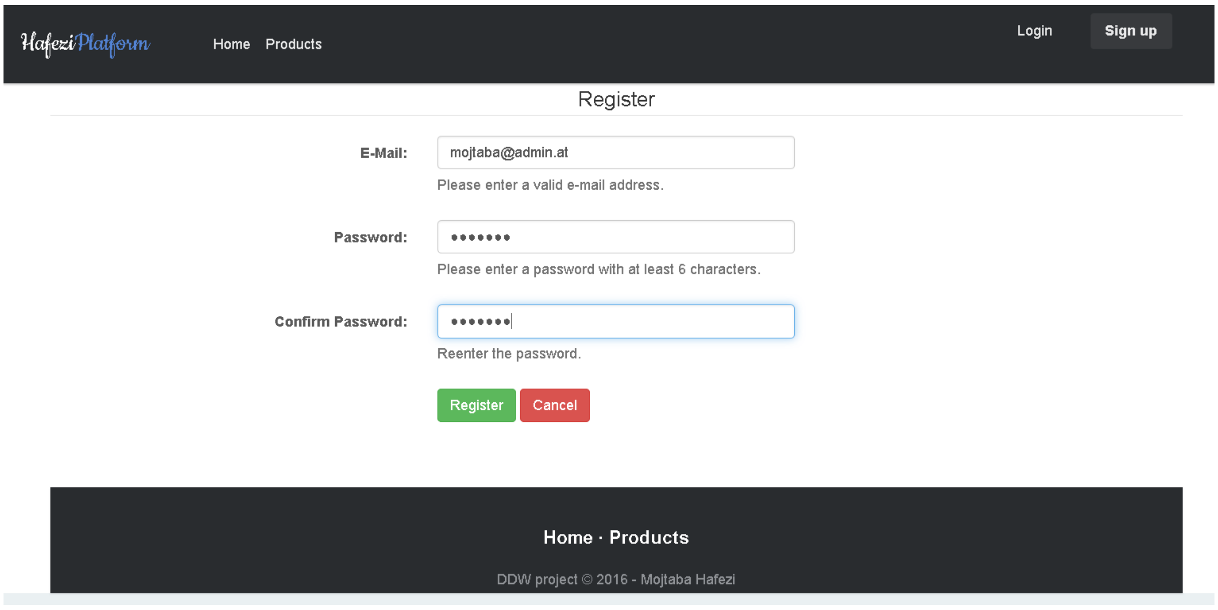
1.) Existing user should be able to login with username and password:


HafeziPlatform
Home
Products
Login
Sign up
Login
E-Mail:
Please enter your e-mail address.
Password:
Please enter your password..
Remember me
Login
Cancel
Home · Products
DDW project © 2016 - Mojtaba Hafezi

After successful login, the user gets redirected to the previous page he came from or the standard page if the corresponding rights are not available. On the right upper corner one can see the email address used to login – this could easily be changed for a username etc.


HafeziPlatform
Home
Products
Edit data
Users
Logout
Welcome, MHAFEZI@uclan ★
Example headline.
Donec sed odio dui. Etiam porta sem malesuada magna mollis euismod. Nullam id dolor id nibh ultricies vehicula ut id elit.
Sign up today

2.) Registration of users with username and password:



Register

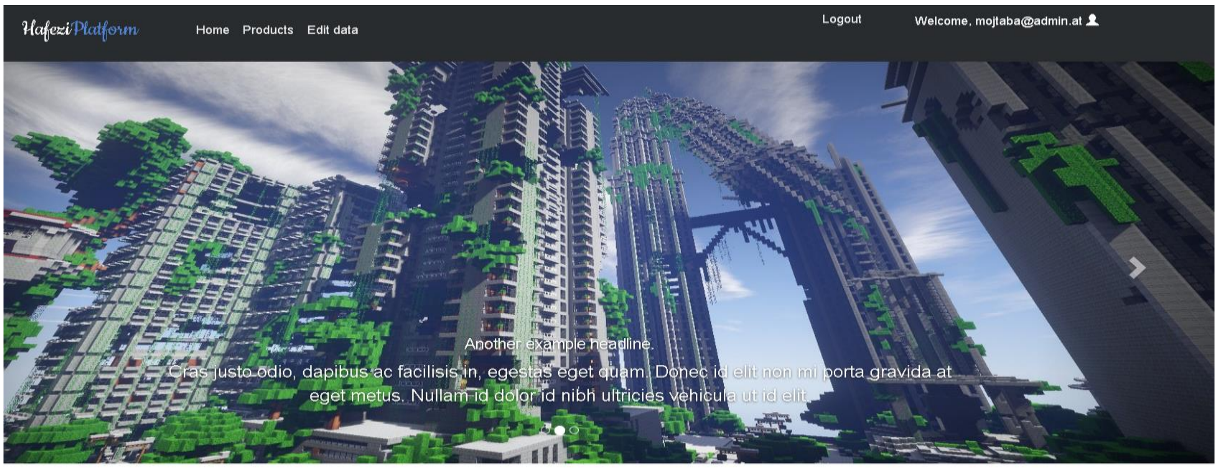
E-Mail: mojtaba@admin.at
Please enter a valid e-mail address.

Password: •••••••
Please enter a password with at least 6 characters.

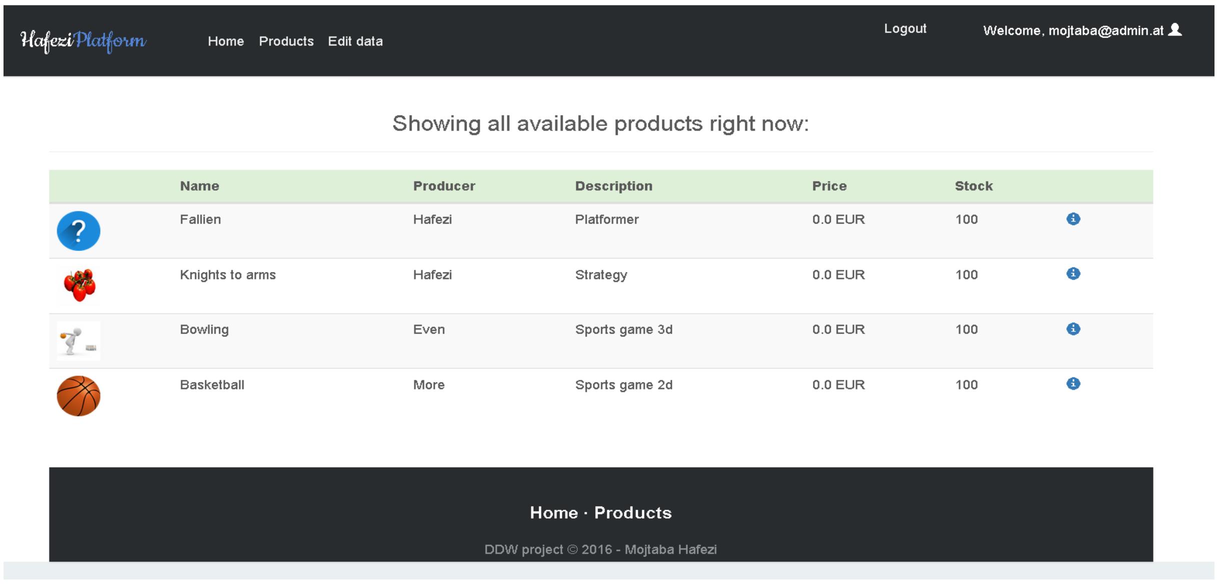
Confirm Password: •••••••
Reenter the password.

DDW project © 2016 - Mojtaba Hafezi

After successful registration the user stays logged in and gets redirected to the homepage.


HafeziPlatform
Home
Products
Edit data
Logout
Welcome, mojtaba@admin.at
Another example headline.
Cras justo odio, dapibus ac facilisis in, egestas eget quam. Donec id elit non mi porta gravida at eget metus. Nullam id dolor id nibh ultricies vehicula ut id elit.

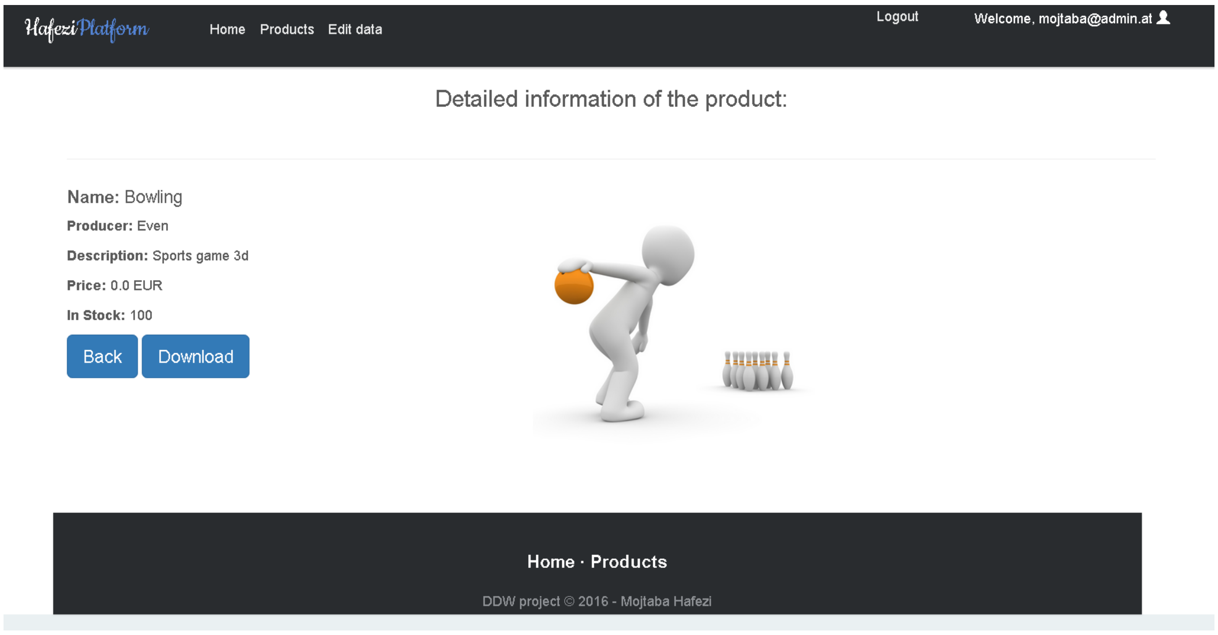
3.) Persistent state between pages:



Showing all available products right now:

	Name	Producer	Description	Price	Stock	
	Fallien	Hafezi	Platformer	0.0 EUR	100	ℹ
	Knights to arms	Hafezi	Strategy	0.0 EUR	100	ℹ
	Bowling	Even	Sports game 3d	0.0 EUR	100	ℹ
	Basketball	More	Sports game 2d	0.0 EUR	100	ℹ

DDW project © 2016 - Mojtaba Hafezi



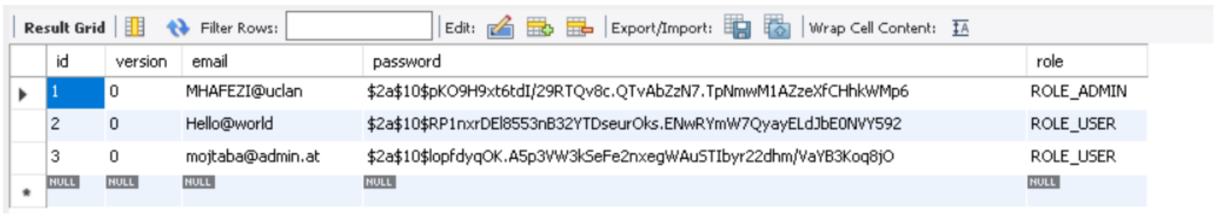
Detailed information of the product:

Name: Bowling
Producer: Even
Description: Sports game 3d
Price: 0.0 EUR
In Stock: 100

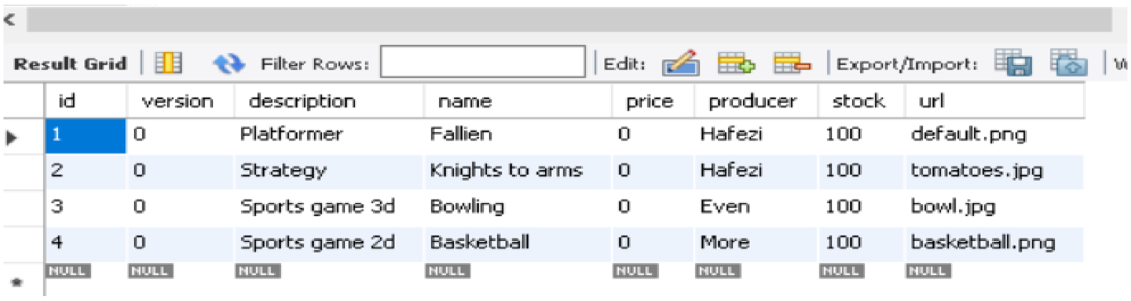
By clicking on the information icon (which only logged in users can see) the user gets forwarded to the detail page where a download function is available to get the showing picture.

4.) Minimum of 2 MYSQL tables for data:

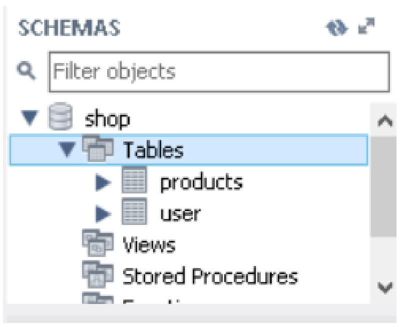
The MYSQL database has 2 tables which are user and products. The user contains the email addresses, the hashed password and the role. The products contain more information about the product with the URL to the picture.



id	version	email	password	role
1	0	MHAFEZI@uclan	$2a$10$pKO9H9xt6tdI/29RTQv8c.QTvAbZzN7.TpNmwM1AZzeXfCHhkWMp6	ROLE_ADMIN
2	0	Hello@world	$2a$10$RP1nxrDEl8553nB32YTDseurOks.ENwRYmW7QyayELdJbE0NVY592	ROLE_USER
3	0	mojtaba@admin.at	$2a$10$lopfdyqOK.A5p3VW3kSeFe2nxegWAuSTIbyr22dhm/VaYB3Koq8jO	ROLE_USER
NULL	NULL	NULL	NULL	NULL

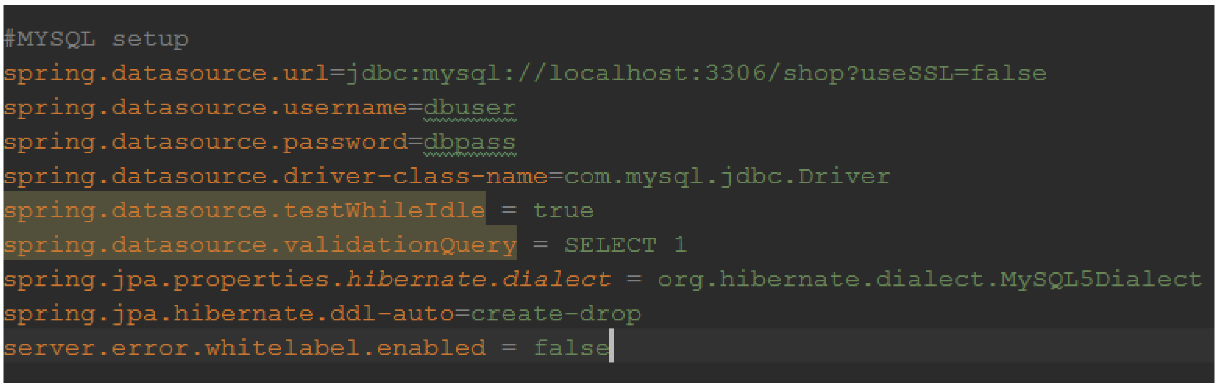


id	version	description	name	price	producer	stock	url
1	0	Platformer	Fallien	0	Hafezi	100	default.png
2	0	Strategy	Knights to arms	0	Hafezi	100	tomatoes.jpg
3	0	Sports game 3d	Bowling	0	Even	100	bowl.jpg
4	0	Sports game 2d	Basketball	0	More	100	basketball.png
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL



5.) No deprecated methods for connections

By using the spring boot application with correct configuration, the connection to many databases is possible be it SQL or NOSQL databases.

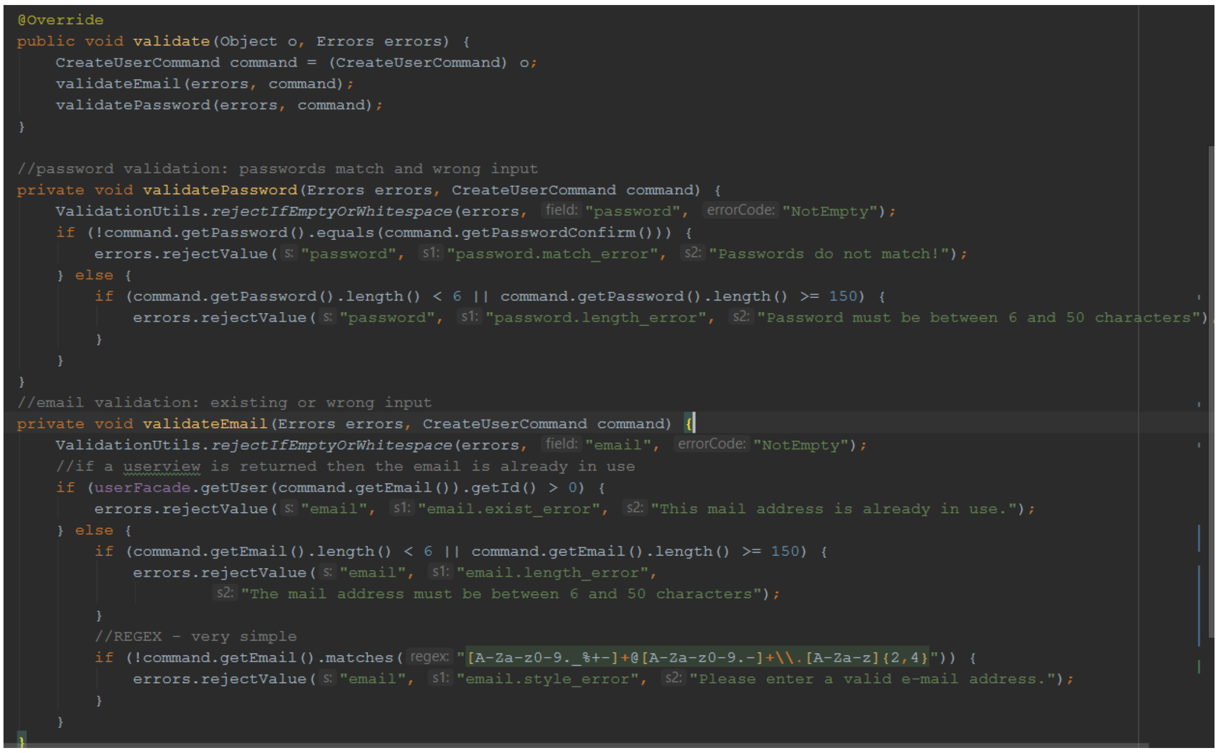


```
#MYSQL setup
spring.datasource.url=jdbc:mysql://localhost:3306/shop?useSSL=false
spring.datasource.username=dbuser
spring.datasource.password=dbpass
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.testWhileIdle = true
spring.datasource.validationQuery = SELECT 1
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
spring.jpa.hibernate.ddl-auto=create-drop
server.error.whitelabel.enabled = false
```

The dependencies must be configured correctly too. As one can see, the SSL is disabled for development process. Furthermore the username and password chosen are not safe.

6.) Server and client side validation:

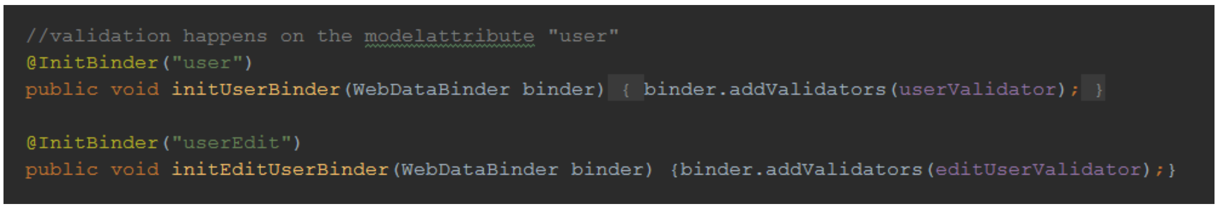
Server validation: By extending the Spring Security validators and adding additional checks the server validation works without problems.



```
@Override
public void validate(Object o, Errors errors) {
    CreateUserCommand command = (CreateUserCommand) o;
    validateEmail(errors, command);
    validatePassword(errors, command);
}

//password validation: passwords match and wrong input
private void validatePassword(Errors errors, CreateUserCommand command) {
    ValidationUtils.rejectIfEmptyOrWhitespace(errors, field: "password", errorCode: "NotEmpty");
    if (!command.getPassword().equals(command.getPasswordConfirm())) {
        errors.rejectValue( s: "password", s1: "password.match_error", s2: "Passwords do not match!");
    } else {
        if (command.getPassword().length() < 6 || command.getPassword().length() >= 150) {
            errors.rejectValue( s: "password", s1: "password.length_error", s2: "Password must be between 6 and 50 characters")
        }
    }
}
//email validation: existing or wrong input
private void validateEmail(Errors errors, CreateUserCommand command) {
    ValidationUtils.rejectIfEmptyOrWhitespace(errors, field: "email", errorCode: "NotEmpty");
    //if a userview is returned then the email is already in use
    if (userFacade.getUser(command.getEmail()).getId() > 0) {
        errors.rejectValue( s: "email", s1: "email.exist_error", s2: "This mail address is already in use.");
    } else {
        if (command.getEmail().length() < 6 || command.getEmail().length() >= 150) {
            errors.rejectValue( s: "email", s1: "email.length_error",
                    s2: "The mail address must be between 6 and 50 characters");
        }
        //REGEX - very simple
        if (!command.getEmail().matches( regex: "[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,4}")) {
            errors.rejectValue( s: "email", s1: "email.style_error", s2: "Please enter a valid e-mail address.");
        }
    }
}
```

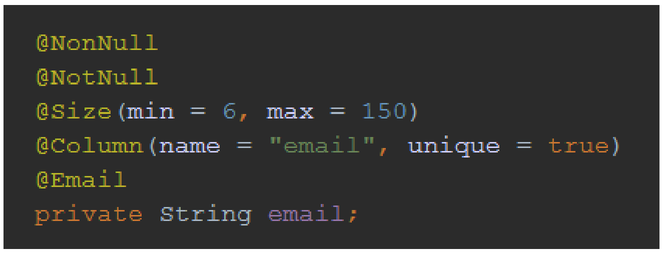
To get the validators working in the controller classes they need to be declared correctly:



```
//validation happens on the modelattribute "user"
@InitBinder("user")
public void initUserBinder(WebDataBinder binder) { binder.addValidators(userValidator); }

@InitBinder("userEdit")
public void initEditUserBinder(WebDataBinder binder) {binder.addValidators(editUserValidator);}
```

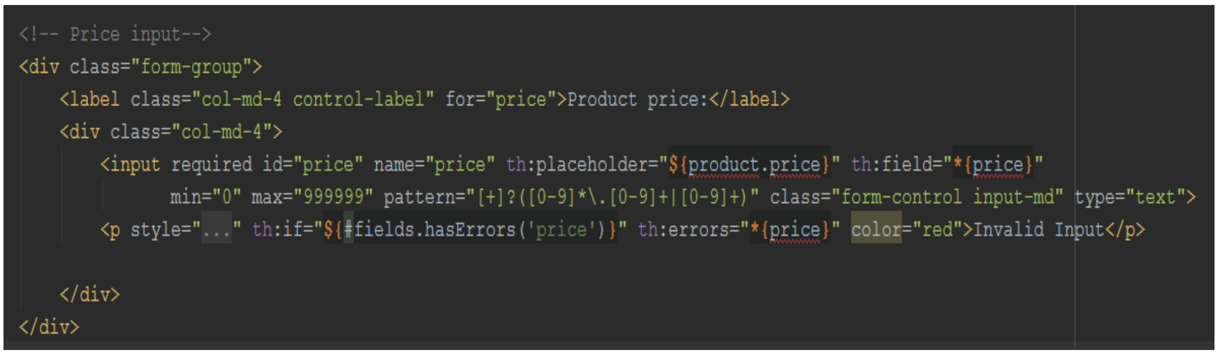
This way the Model attributes “user” or “userEdit” have their corresponding validation. With this approach, custom validation can be added to the already included domain validations:



```
@NonNull
@NotNull
@Size(min = 6, max = 150)
@Column(name = "email", unique = true)
@Email
private String email;
```

This code catches all problems occurring and forwards it to the BindingResult to provide the information on the views. Different Validators are used for different validation on users and products.

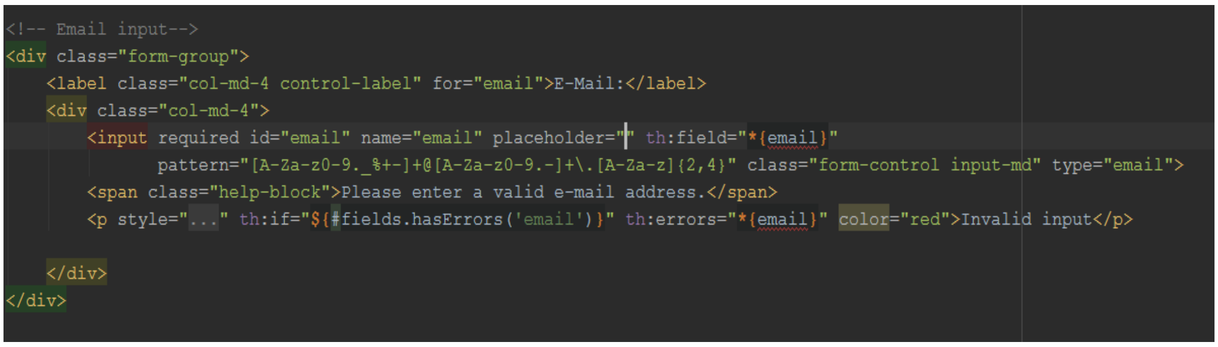
On the client side, I decided to use HTML-validation. Next to auto-populating by giving the “th:placeholer” the current attribute of the field, I also set the min and max values in addition to a simple regular expression for a field of the type DOUBLE.



```
<!-- Price input-->
<div class="form-group">
    <label class="col-md-4 control-label" for="price">Product price:</label>
    <div class="col-md-4">
        <input required id="price" name="price" th:placeholder="${product.price}" th:field="*{price}"
               min="0" max="999999" pattern="[+]?([0-9]*\.[0-9]+|[0-9]+)" class="form-control input-md" type="text">
        <p style="..." th:if="${#fields.hasErrors('price')}" th:errors="*{price}" color="red">Invalid Input</p>

    </div>
</div>
```

The required statement in the input demands information from the user and thus the form cannot be submitted without all required fields filled out. With the use of corresponding patterns the minimum length of characters for passwords and other product details are secured. The email has its own validation through a regular expression.



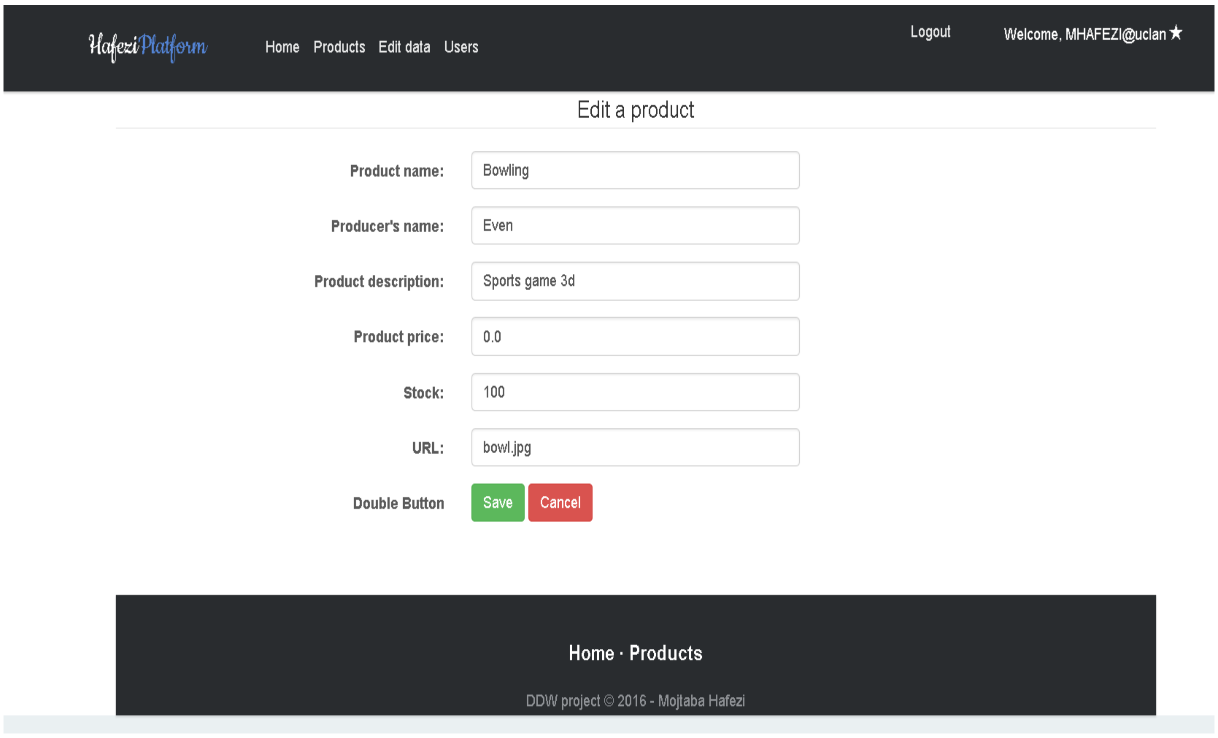
```
<!-- Email input-->
<div class="form-group">
    <label class="col-md-4 control-label" for="email">E-Mail:</label>
    <div class="col-md-4">
        <input required id="email" name="email" placeholder="" th:field="*{email}"
               pattern="[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}" class="form-control input-md" type="email">
        <span class="help-block">Please enter a valid e-mail address.</span>
        <p style="..." th:if="${#fields.hasErrors('email')}" th:errors="*{email}" color="red">Invalid input</p>

    </div>
</div>
```

Every form is validated with both approaches for client and server side validation.

Validations can be found in the domain classes, the validators and all views (HTMLs) with a form.

7.) CRUD operations on data:

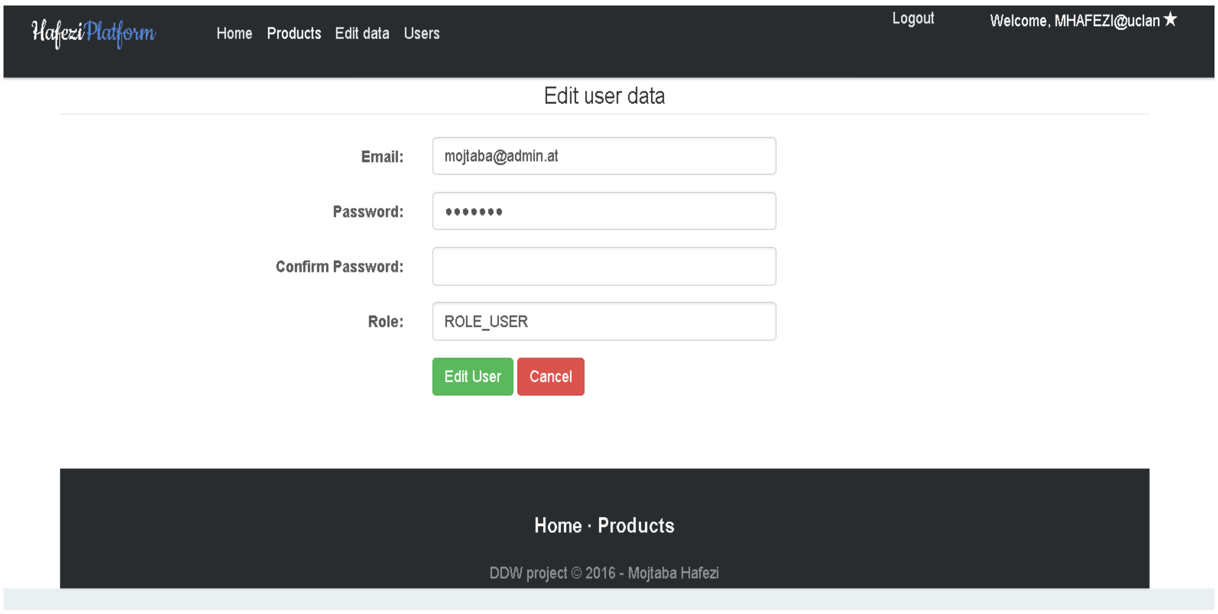

HafeziPlatform
Home Products Edit data Users
Logout
Welcome, MHAFEZI@uclan ★

Edit a product

Product name:	Bowling
Producer's name:	Even
Product description:	Sports game 3d
Product price:	0.0
Stock:	100
URL:	bowl.jpg
Double Button	Save Cancel

Home · Products
DDW project © 2016 - Mojtaba Hafezi

A **broad range** of information can be changed. The employees and administrators can operate all CRUD operations on the products, while the users can only be viewed and changed by the admin.


HafeziPlatform
Home Products Edit data Users
Logout
Welcome, MHAFEZI@uclan ★

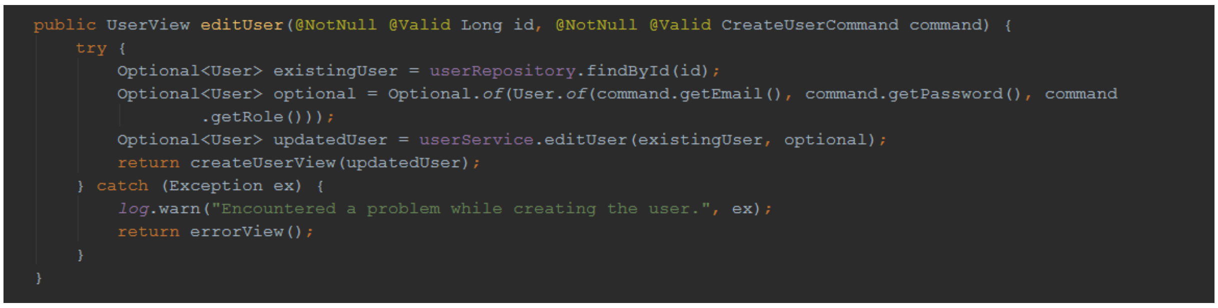
Edit user data

Email:	mojtaba@admin.at
Password:	•••••••
Confirm Password:	
Role:	ROLE_USER
	Edit User Cancel

Home · Products
DDW project © 2016 - Mojtaba Hafezi

Some of the fields get **auto-populated**. This is provided mostly through the server but also the html placeholder comes in use (for products for example). Especially in the edit forms the fields are auto-populated to reduce the workload on the users (changes on users or products alike).

The source code for the crud operations are all in the façade classes which make use of the services.

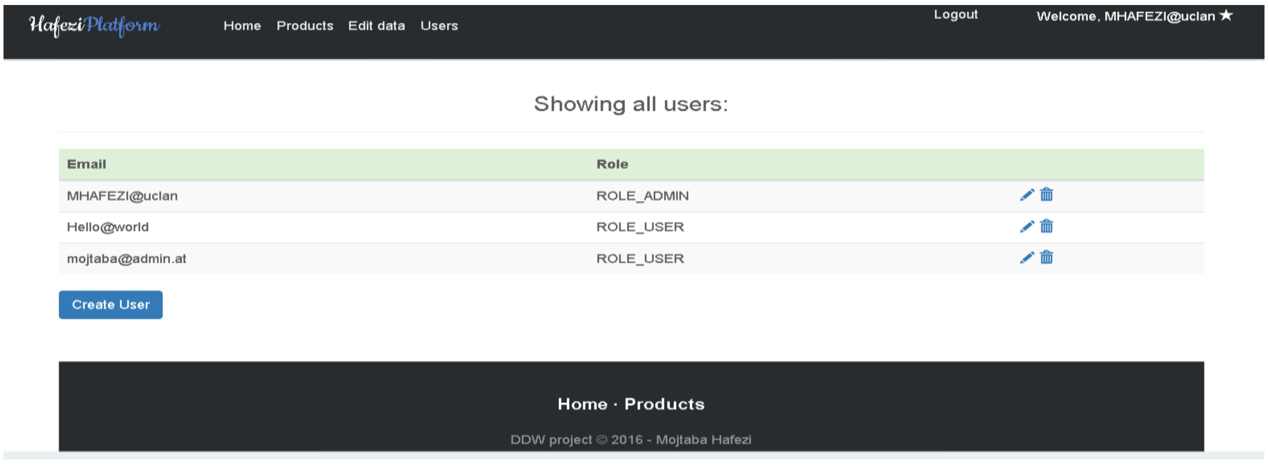


```
public UserView editUser(@NotNull @Valid Long id, @NotNull @Valid CreateUserCommand command) {
    try {
        Optional<User> existingUser = userRepository.findById(id);
        Optional<User> optional = Optional.of(User.of(command.getEmail(), command.getPassword(), command
                .getRole()));
        Optional<User> updatedUser = userService.editUser(existingUser, optional);
        return createUserView(updatedUser);
    } catch (Exception ex) {
        log.warn("Encountered a problem while creating the user.", ex);
        return errorView();
    }
}
```

8.) Different users with different rights/views:

To begin with I provide three roles: User, Admin and Employee. The employee can change product data and their own user data like password or email. A normal user has no CRUD rights on any data but his own login credentials. The admin has CRUD rights on all tables and can see the list of users etc.

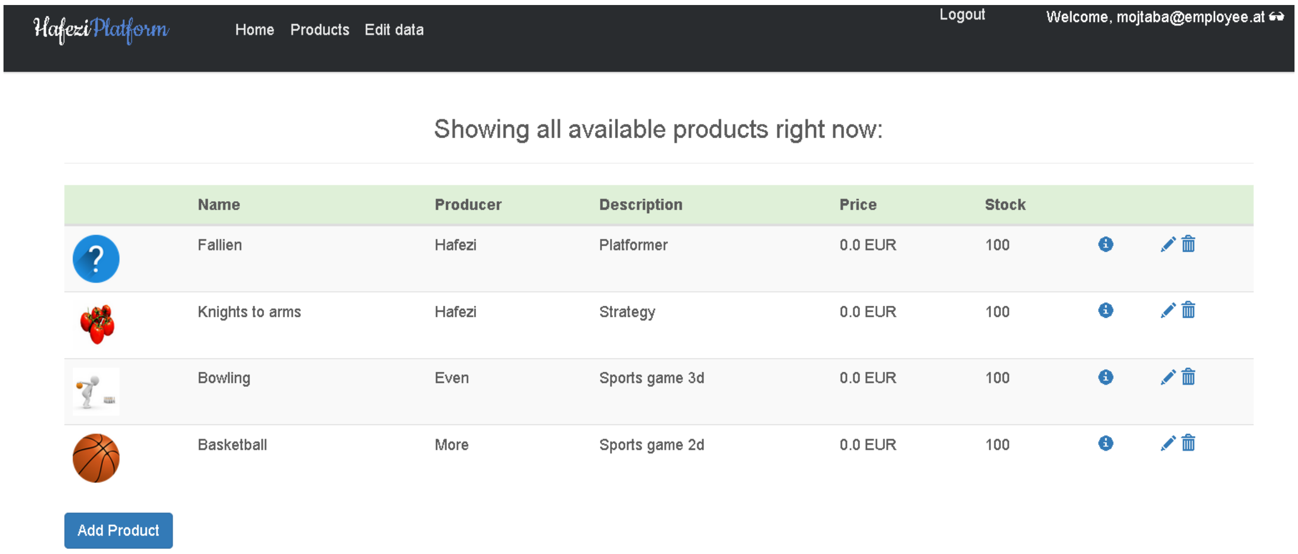
The admin is marked on the upper right corner by a star and the navigation bar shows more options:



Showing all users:

Email	Role	
MHAFEZI@uclan	ROLE_ADMIN	
Hello@world	ROLE_USER	
mojtaba@admin.at	ROLE_USER	

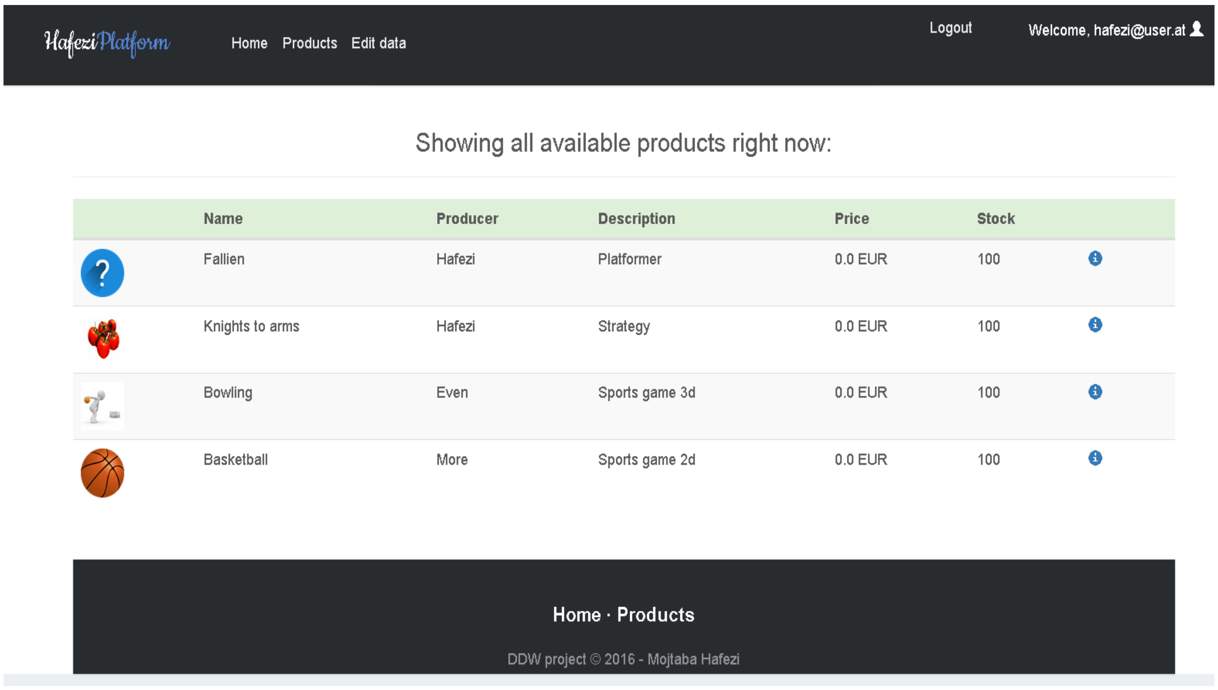
The employee is marked with the sunglasses and has fewer options for changes.



Showing all available products right now:

	Name	Producer	Description	Price	Stock		
	Fallien	Hafezi	Platformer	0.0 EUR	100		
	Knights to arms	Hafezi	Strategy	0.0 EUR	100		
	Bowling	Even	Sports game 3d	0.0 EUR	100		
	Basketball	More	Sports game 2d	0.0 EUR	100		

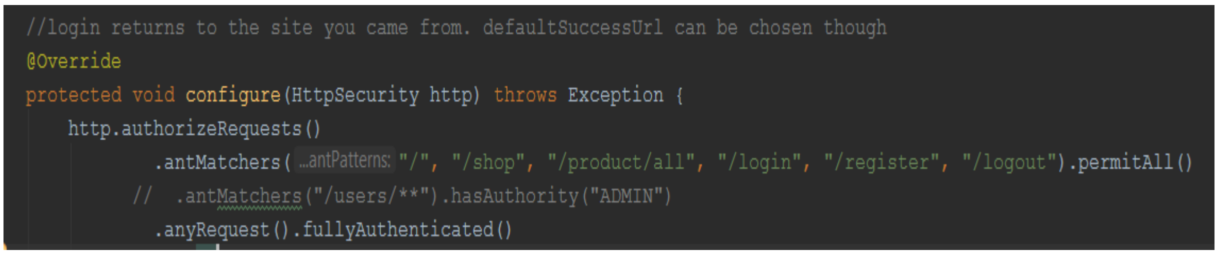
The user marked with a user icon, can only edit his/her own data and view/download the products.



Showing all available products right now:

	Name	Producer	Description	Price	Stock	
	Fallien	Hafezi	Platformer	0.0 EUR	100	ℹ
	Knights to arms	Hafezi	Strategy	0.0 EUR	100	ℹ
	Bowling	Even	Sports game 3d	0.0 EUR	100	ℹ
	Basketball	More	Sports game 2d	0.0 EUR	100	ℹ

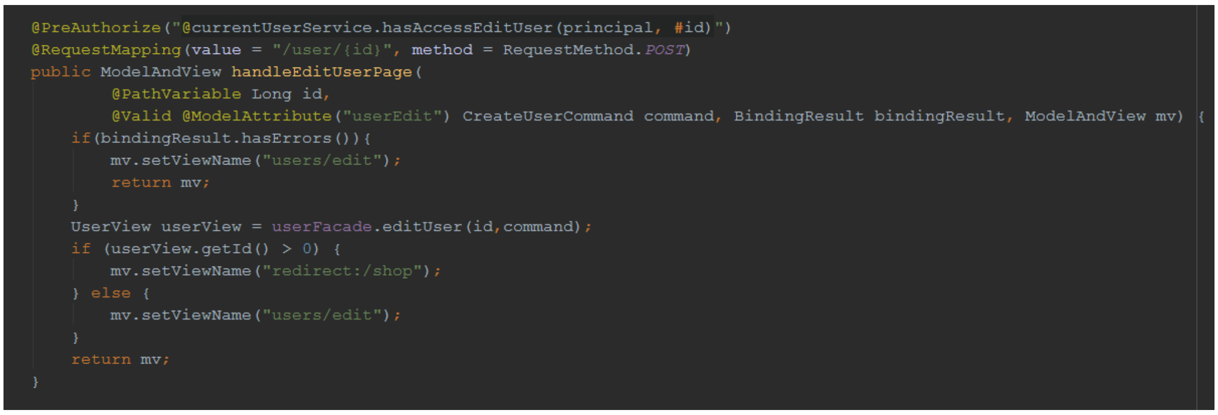
To make this possible different approaches were used. Not only the spring security authorization, which is configurable by telling spring which URLs are restricted for which users, but also Thymeleaf security and the method authorization has been implemented.



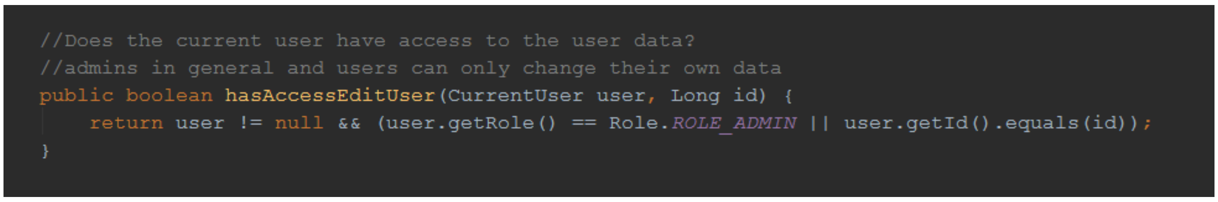
```
//login returns to the site you came from. defaultSuccessUrl can be chosen though
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
            .antMatchers( ...antPatterns: "/", "/shop", "/product/all", "/login", "/register", "/logout").permitAll()
        //  .antMatchers("/users/**").hasAuthority("ADMIN")
            .anyRequest().fullyAuthenticated()
```

With the commented ".antMatchers()" method one can add as many URLs with the corresponding authority.

In the situation, a user wants to edit his own credentials but is not an admin, the approach shown above wouldn't make it possible for a simple user to change anything. In this scenario, following approach solves the problem in an elegant way:



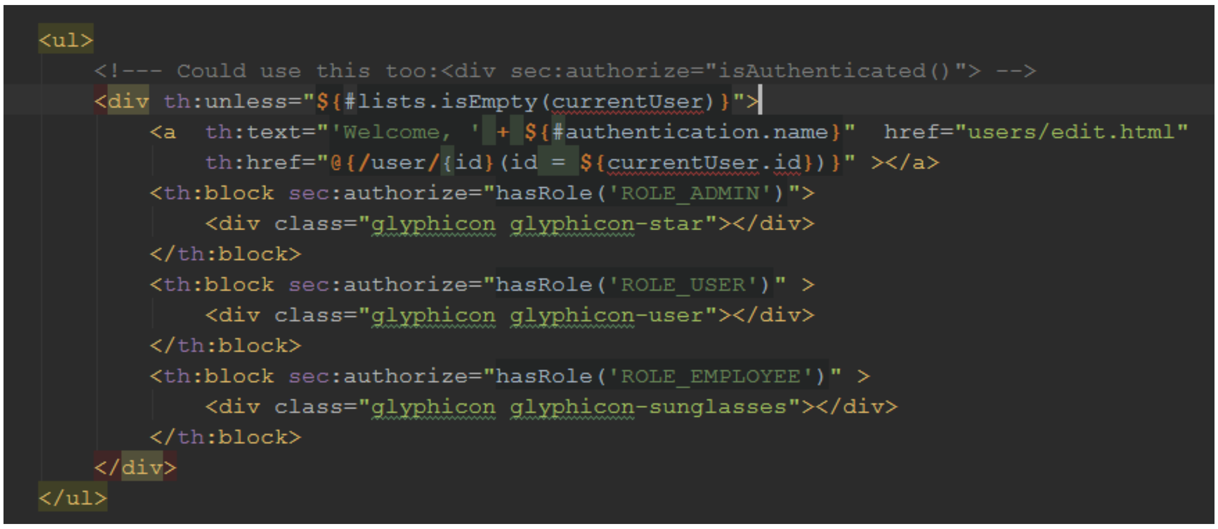
```
@PreAuthorize("@currentUserService.hasAccessEditUser(principal, #id)")
@RequestMapping(value = "/user/{id}", method = RequestMethod.POST)
public ModelAndView handleEditUserPage(
        @PathVariable Long id,
        @Valid @ModelAttribute("userEdit") CreateUserCommand command, BindingResult bindingResult, ModelAndView mv) {
    if(bindingResult.hasErrors()){
        mv.setViewName("users/edit");
        return mv;
    }
    UserView userView = userFacade.editUser(id,command);
    if (userView.getId() > 0) {
        mv.setViewName("redirect:/shop");
    } else {
        mv.setViewName("users/edit");
    }
    return mv;
}
```



```
//Does the current user have access to the user data?
//admins in general and users can only change their own data
public boolean hasAccessEditUser(CurrentUser user, Long id) {
    return user != null && (user.getRole() == Role.ROLE_ADMIN || user.getId().equals(id));
}
```

With the “PreAuthorize” I ask the spring security to check if the method “hasAccessEditUser” returns true or false in the CurrentUserService class (which needs to extend the UserDetailsService to get called with the annotation). So, when a normal user clicks on “Edit Data” on the website to change his own email address or password, his id is used for the get (afterwards post) request and if either the role fits or the id then the access is granted. The integration of spring security here is of importance as the principal is the currently authenticated user.

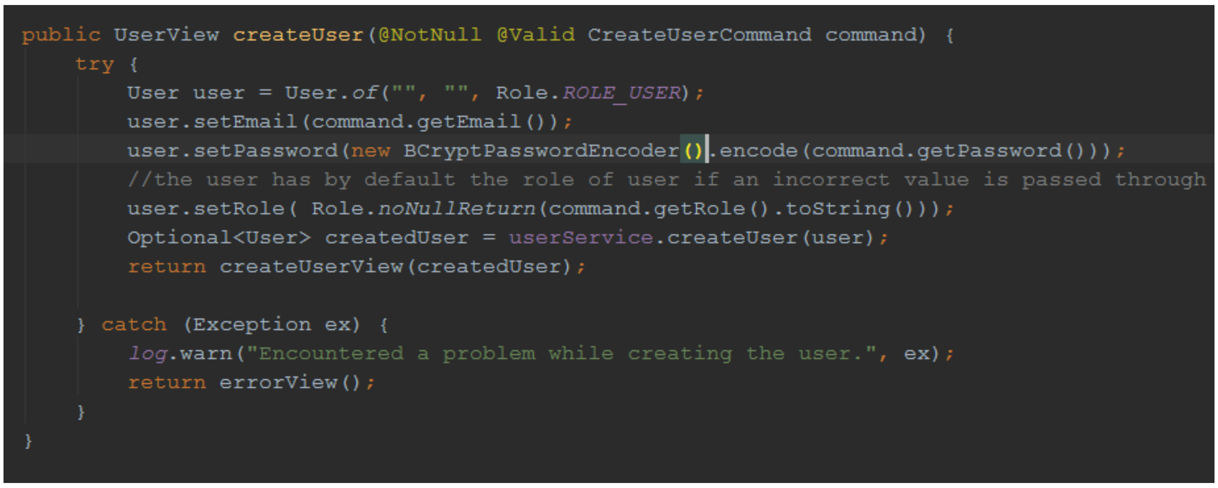
For providing a bespoke interface for each user the Thymeleaf security dialect is used in the views to provide dynamical checks on the user’s current role.



```
<ul>
    <!--- Could use this too:<div sec:authorize="isAuthenticated()"> -->
    <div th:unless="${#lists.isEmpty(currentUser)}">
        <a  th:text="'Welcome, ' + ${#authentication.name}"  href="users/edit.html"
            th:href="@{/user/{id}(id = ${currentUser.id})}" ></a>
        <th:block sec:authorize="hasRole('ROLE_ADMIN')">
            <div class="glyphicon glyphicon-star"></div>
        </th:block>
        <th:block sec:authorize="hasRole('ROLE_USER')" >
            <div class="glyphicon glyphicon-user"></div>
        </th:block>
        <th:block sec:authorize="hasRole('ROLE_EMPLOYEE')" >
            <div class="glyphicon glyphicon-sunglasses"></div>
        </th:block>
    </div>
</ul>
```

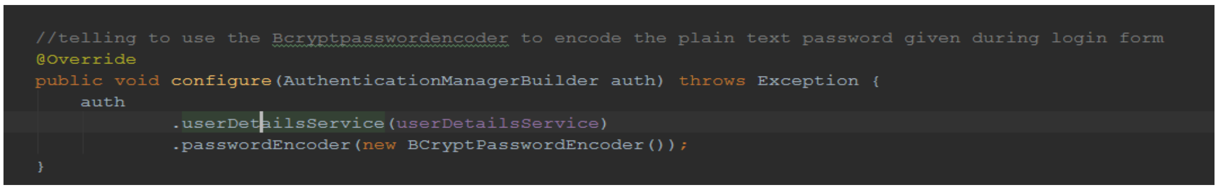
9.) Secure password hashing with additional salting:

After researching many hashing algorithms and possibilities in java I came to the conclusion to use the BCrypt hashing algorithm. It has a salting function, which makes the passwords very safe and is added dynamically. There is no way to decode the hashed algorithm so the match function gets used to check for correctness of the passwords. CSRF_Token was also enabled for better protection.



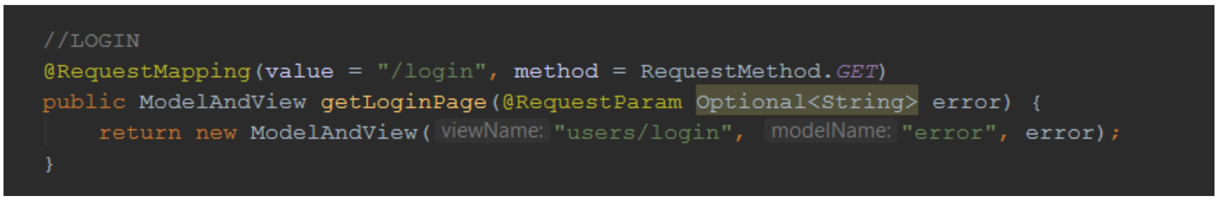
```
public UserView createUser(@NotNull @Valid CreateUserCommand command) {
    try {
        User user = User.of("", "", Role.ROLE_USER);
        user.setEmail(command.getEmail());
        user.setPassword(new BCryptPasswordEncoder().encode(command.getPassword()));
        //the user has by default the role of user if an incorrect value is passed through
        user.setRole( Role.noNullReturn(command.getRole().toString()));
        Optional<User> createdUser = userService.createUser(user);
        return createUserView(createdUser);

    } catch (Exception ex) {
        log.warn("Encountered a problem while creating the user.", ex);
        return errorView();
    }
}
```



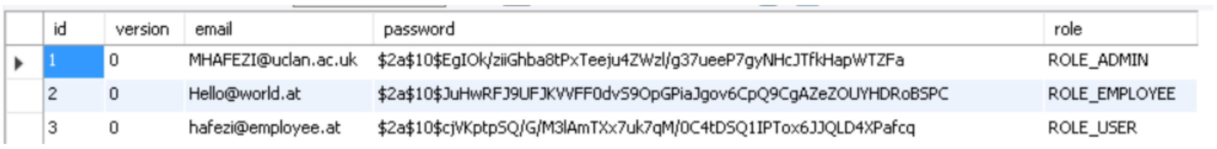
```
//telling to use the Bcryptpasswordencoder to encode the plain text password given during login form
@Override
public void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth
            .userDetailsService(userDetailsService)
            .passwordEncoder(new BCryptPasswordEncoder());
}
```

In the configure method I demand the BCryptPasswordEncoder to be used for encoding all passwords in the checks (Login).



```
//LOGIN
@RequestMapping(value = "/login", method = RequestMethod.GET)
public ModelAndView getLoginPage(@RequestParam Optional<String> error) {
    return new ModelAndView( viewName: "users/login",  modelName: "error", error);
}
```

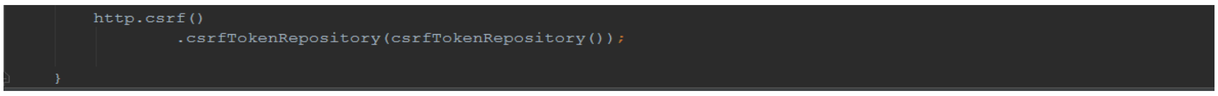
Login Process: The user opens the /login URL and fills up the form to submit it. Meanwhile the LoginController returned a view with the form and after submitting the data the loadUserByUsername() gets called by the UserDetailsService (which I implemented with the CurrentUserService class). The username (as in the configuration shown equals the email) is given to the load method and the CurrentUser is returned from the Service layer. Afterwards the getPassword() method gets called and the comparison with the saved hashed password happens with the matches() method of the BCryptPasswordEncoder class. If any problems occur, the URL is set to login?error where the message “Username or Email is wrong”. The actual problem is not visible to make it more secure against intruders. Further details can be viewed in the source code.



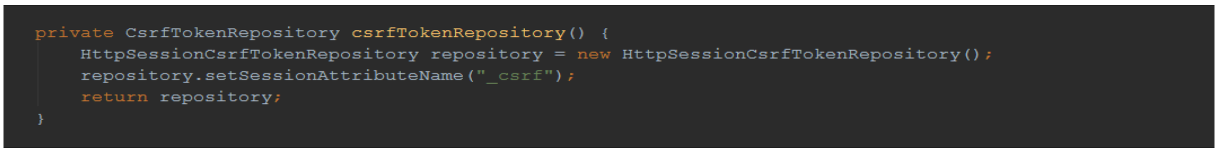
id	version	email	password	role
1	0	MHAFEZI@uclan.ac.uk	$2a$10$EgIOk/ziiGhba8tPxTeeju4ZWzl/g37ueeP7gyNHcJTfkHapWTZFa	ROLE_ADMIN
2	0	Hello@world.at	$2a$10$JuHwRFJ9UFJKVVFF0dvS9OpGPiaJgov6CpQ9CgAZeZOUYHDRoBSPC	ROLE_EMPLOYEE
3	0	hafezi@employee.at	$2a$10$cjVKptpSQ/G/M3lAmTXx7uk7qM/0C4tDSQ1IPTox6JJQLD4XPafcq	ROLE_USER

10.)Additional security features:

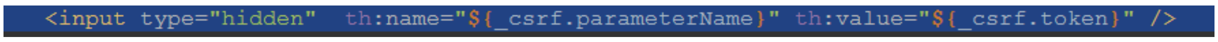
To prevent Cross-site request forgery a CSRF token is created in a cookie, which is deleted after the user logs out again. To make this possible, proper HTTP verbs like GET and POST are used through the complete website. Using the configure method in the WEBSECURITYCONFIG class the application ensures any request to the application requires authentication for specific URLS and allows form based login. Furthermore, the logout is handled by invalidating the session, deleting cookies and redirecting to the login page.



```
        http.csrf()
                .csrfTokenRepository(csrfTokenRepository());
    }
```



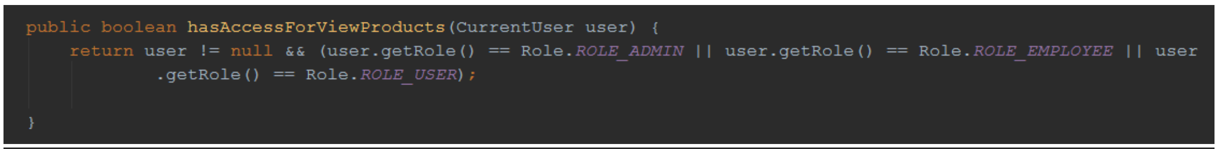
```
private CsrfTokenRepository csrfTokenRepository() {
    HttpSessionCsrfTokenRepository repository = new HttpSessionCsrfTokenRepository();
    repository.setSessionAttributeName("_csrf");
    return repository;
}
```



```
  <input type="hidden"  th:name="${_csrf.parameterName}" th:value="${_csrf.token}" />
```

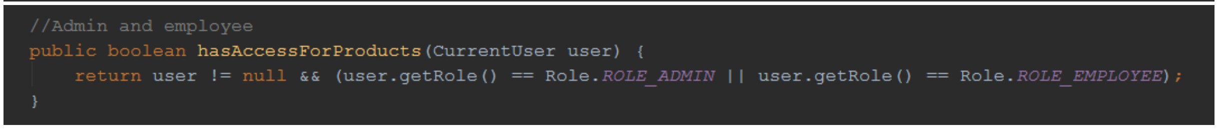
This hidden input field needs to be declared in each form to prevent Cross-site request forgery.

Also, the “@PreAuthorize” methods used for handling appropriate access in addition to Thymeleaf security dialect are features as mentioned already.

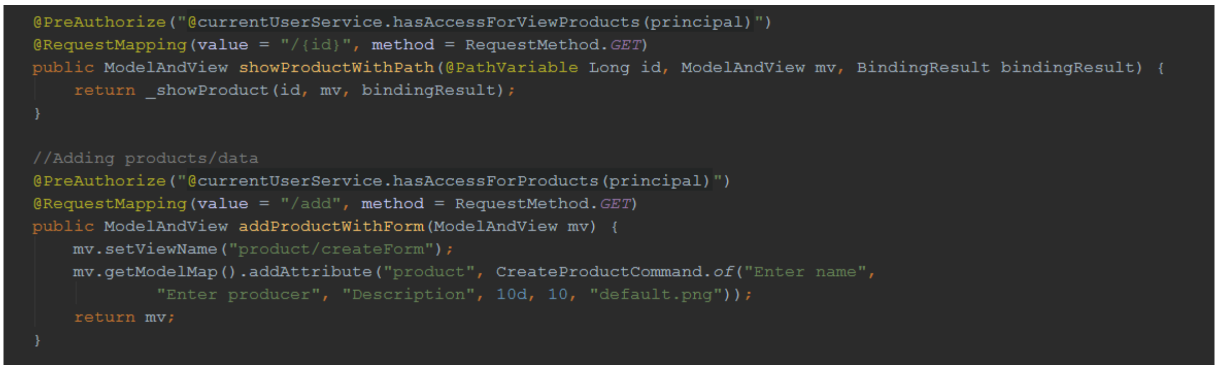


```
public boolean hasAccessForViewProducts(CurrentUser user) {
    return user != null && (user.getRole() == Role.ROLE_ADMIN || user.getRole() == Role.ROLE_EMPLOYEE || user
            .getRole() == Role.ROLE_USER);

}
```



```
//Admin and employee
public boolean hasAccessForProducts(CurrentUser user) {
    return user != null && (user.getRole() == Role.ROLE_ADMIN || user.getRole() == Role.ROLE_EMPLOYEE);
}
```

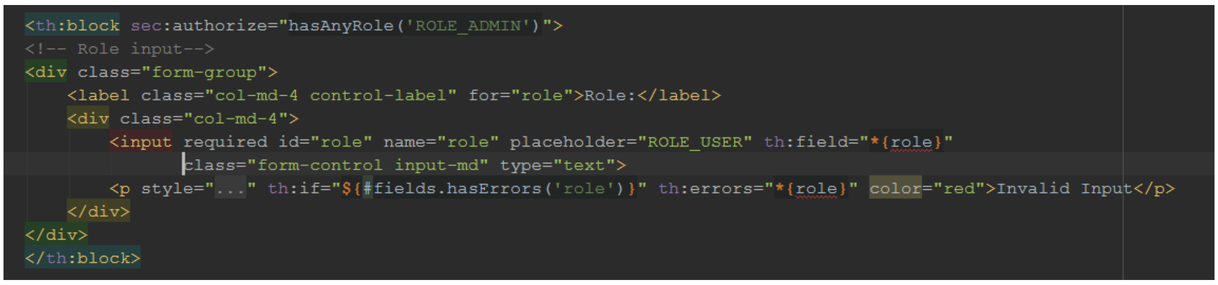


```
@PreAuthorize("@currentUserService.hasAccessForViewProducts(principal)")
@RequestMapping(value = "/{id}", method = RequestMethod.GET)
public ModelAndView showProductWithPath(@PathVariable Long id, ModelAndView mv, BindingResult bindingResult) {
    return _showProduct(id, mv, bindingResult);
}

//Adding products/data
@PreAuthorize("@currentUserService.hasAccessForProducts(principal)")
@RequestMapping(value = "/add", method = RequestMethod.GET)
public ModelAndView addProductWithForm(ModelAndView mv) {
    mv.setViewName("product/createForm");
    mv.getModelMap().addAttribute("product", CreateProductCommand.of("Enter name",
            "Enter producer", "Description", 10d, 10, "default.png"));
    return mv;
}
```

Found in the controller classes and the CurrentUserService class. The CSRF declarations are in all views with a form included.

The Thymeleaf security feature is used for preventing a user for example to change his own role from a normal user to admin or similar.

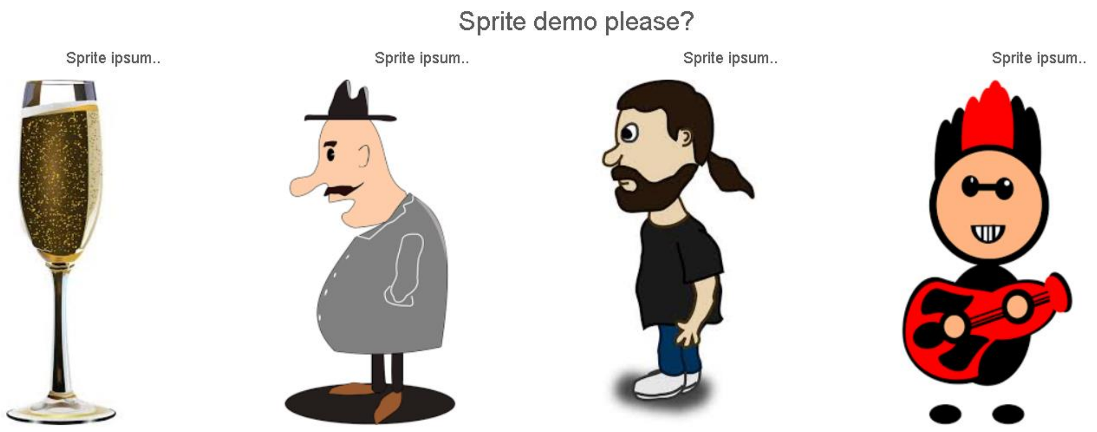


```
<th:block sec:authorize="hasAnyRole('ROLE_ADMIN')">
<!-- Role input-->
<div class="form-group">
    <label class="col-md-4 control-label" for="role">Role:</label>
    <div class="col-md-4">
        <input required id="role" name="role" placeholder="ROLE_USER" th:field="*{role}"
               class="form-control input-md" type="text">
        <p style="..." th:if="${#fields.hasErrors('role')}" th:errors="*{role}" color="red">Invalid Input</p>
    </div>
</div>
</th:block>
```

So, if the authenticated user has not the role of an admin, he won’t be able to see this part of the form. The source code can be found in the user/edit.html.

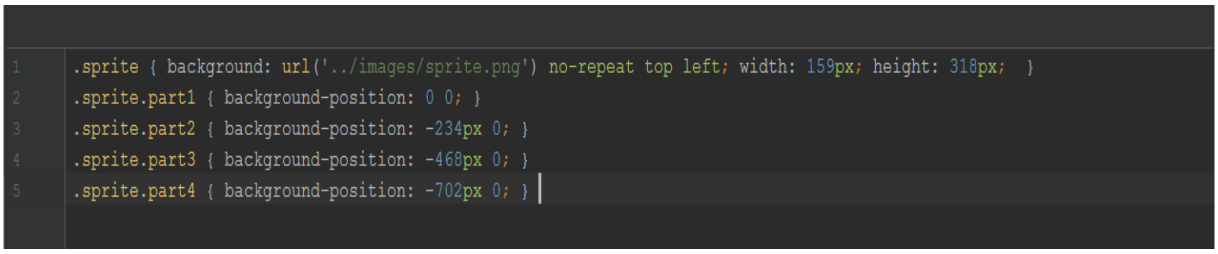
11.) Additional performance feature:

Caching enabled for specific methods of the repository (get all users for example etc.). Furthermore, by using sprites the performance can be increased a lot. I provided these for the pictures at the end of the website. By adding all the figures in one picture and using CSS to determine where each background starts (with the background position), not only are all four pictures loaded by one single HTTP call but also reduces the overhead and bandwidth used.



Depending on the images used and the sprite created the performance increase differs. In this example the sprite is about 120KB bigger in size but reduces the calls needed from four to only one.

This can be easily accomplished using CSS like in the presented demo or special tools which let you choose the pictures and makes a sprite with the corresponding stylesheet for you. I would rather recommend to create the sprites with compression first and let use a tool to determine the positons for a better optimisation. The autogenerating tool created a sprite with 200kb more in size than the compressed one with the positions determined manually.



```
.sprite { background: url('../images/sprite.png') no-repeat top left; width: 159px; height: 318px;  }
.sprite.part1 { background-position: 0 0; }
.sprite.part2 { background-position: -234px 0; }
.sprite.part3 { background-position: -468px 0; }
.sprite.part4 { background-position: -702px 0; }
```

One more features provided, is the adaption of the “Spring Devtools” for raising the developer’s productivity. After adding the dependency with Maven, the corresponding configurations in the IDE had to be made (Project settings and within the REGISTRY) to provide required features.

```
spring.thymeleaf.cache=false
security.basic.enabled=false
spring.thymeleaf.mode=LEGACYHTML5
```

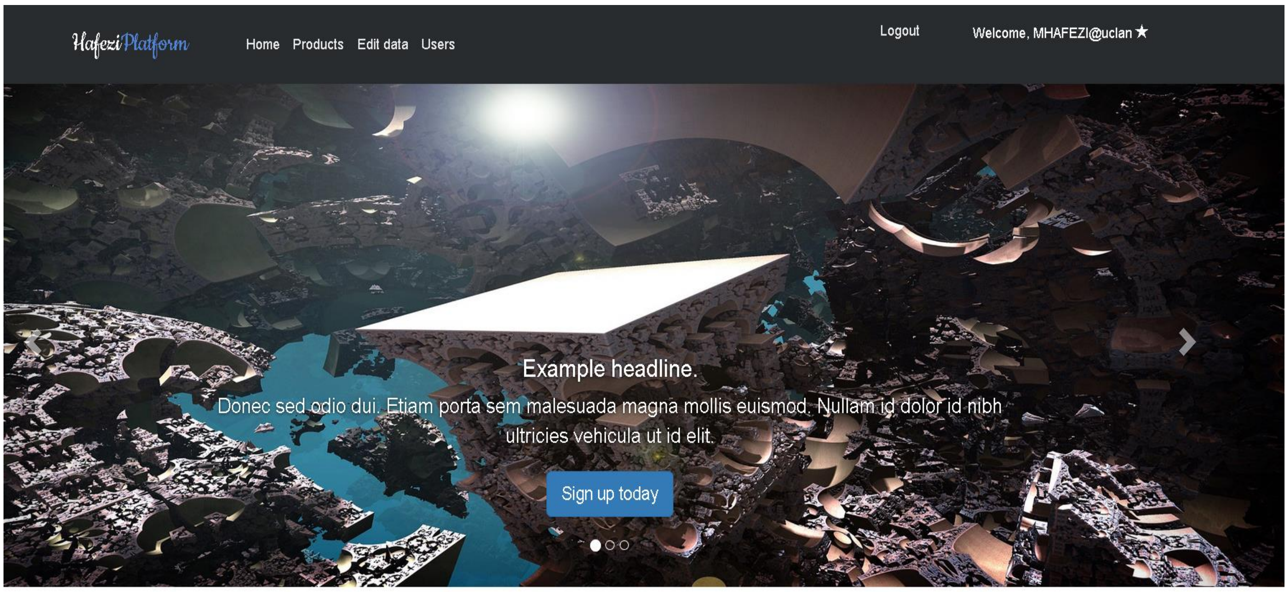
These properties are used to disable caching and allows updating pages without the need to restart the complete application. The mode is set to the LegacyHTML5 to work with the current HTML versions. Any changes made to the project are automatically adapted and only a “build project” is required to update everything (and a refresh on the browser as LiveReload was not installed). As every restart takes about ten to twenty seconds, every minor change a developer makes would result in annoying waiting time and therefore this feature is one of the most developer friendly ones.

```
: Started ShopApplication in 11.908 seconds (JVM running for 12.64)
```

Finally, by using Lombok the developer’s productivity rises even more as boilerplate code can be easily avoided which is seen mainly in the product domain class.

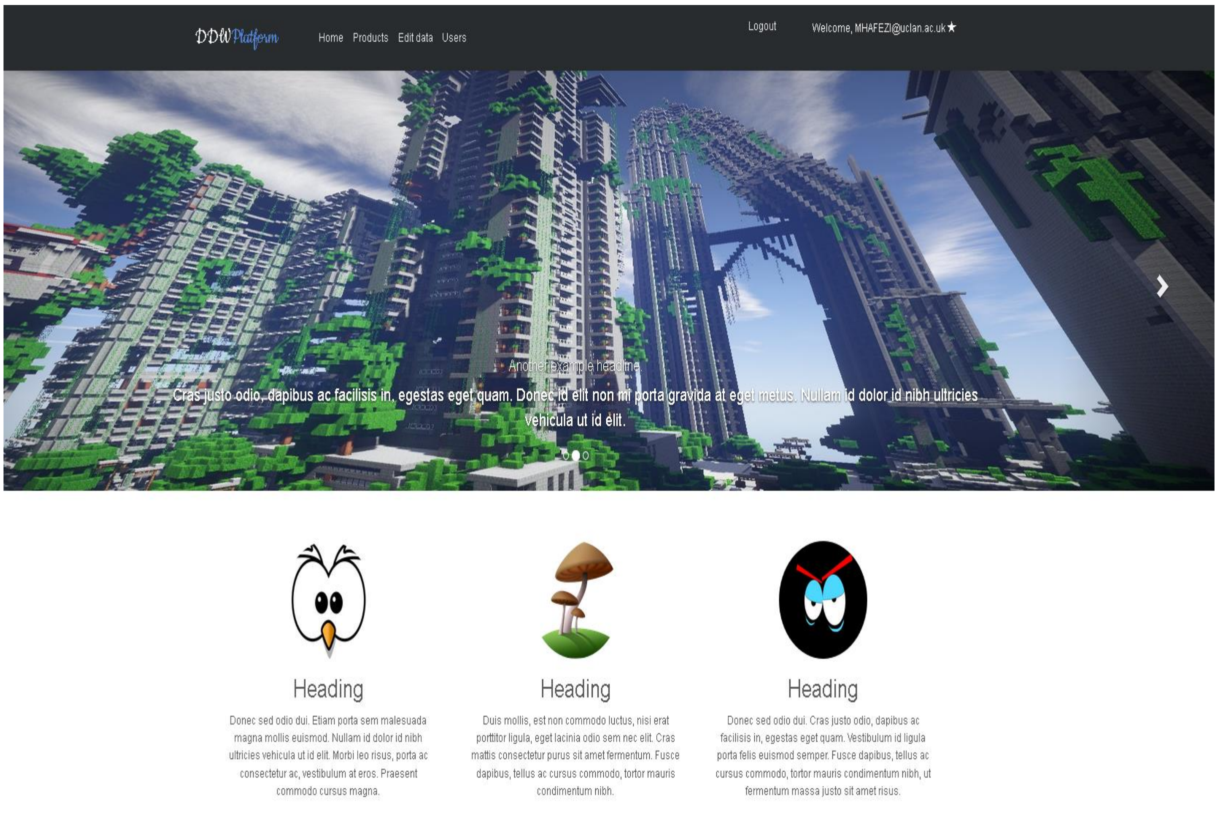
12.) Design, usability and layout:

A well-designed website is provided with responsiveness and usability. More pictures are added in the directory of this document.



Example headline.

Donec sed odio dui. Etiam porta sem malesuada magna mollis euismod. Nullam id dolor id nibh ultricies vehicula ut id elit.



Another example headline.

Cras justo odio, dapibus ac facilisis in, egestas eget quam. Donec id elit non mi porta gravida at eget metus. Nullam id dolor id nibh ultricies vehicula ut id elit.

Heading

Donec sed odio dui. Etiam porta sem malesuada magna mollis euismod. Nullam id dolor id nibh ultricies vehicula ut id elit. Morbi leo risus, porta ac consectetur ac, vestibulum at eros. Praesent commodo cursus magna.

Heading

Duis mollis, est non commodo luctus, nisi erat porttitor ligula, eget lacinia odio sem nec elit. Cras mattis consectetur purus sit amet fermentum. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh.

Heading

Donec sed odio dui. Cras justo odio, dapibus ac facilisis in, egestas eget quam. Vestibulum id ligula porta felis euismod semper. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus.