# Spring Fundamentals Exam Preparation 18 June 2024

# Bon App-étit

Exam for the ["Spring Fundamentals" course @ SoftUni](https://judge.softuni.org/Contests/4469/Spring-Fundamentals-Retake-Exam-14-December-2023).

*Introducing the* ***Bon App-étit****, your culinary companion for exploring new tastes and connecting with  
fellow food enthusiasts. This app is all about taking your cooking experience to the next level by offering a  
wide range of recipes to suit your palate. Browse through hundreds of innovative recipes created by passionate  
chefs and home cooks. You can add your favorite dishes to your personal recipe collection, share your own  
culinary creations, and discover exciting flavor combinations. Unleash your inner chef and embark on a  
gastronomic journey with Bon App-étit!*

***Start -> Entity ->***

*User*

*Ако имаме* javax, а не jacarta, отиваме в интернет в mavenrepository -> изписваме това –

spring-boot-starter-parent, и избираме от всички именно това и 3.3.0 версия

и я пишем тук –

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>3.3.0</version>  
 <relativePath/> *<!-- lookup parent from repository -->*</parent>

Ако излезе такъв надпис в конзолата ни - Could not find artifact mysql:mysql-connector-java:pom:unknown in central (<https://repo.maven.apache.org/maven2>)

Сменяме с по-нова версия отново от mavenrepository – търсим - mysql-connector-j – и избираме или виждаме как е изписана последната там версия – примерно има com. Преди mysql, така че заменяме старата с това:

- <dependency>  
 <groupId>com.mysql</groupId>  
 <artifactId>mysql-connector-j</artifactId>  
</dependency>

И версията на Java я пишем 17

<properties>  
 <java.version>17</java.version>  
</properties>

Вече трябва да виждаме това –

import jakarta.persistence.Entity;  
  
@Entity  
public class User {  
}

There are several requirements you must follow in the implementation:

## Database Requirements

The **Database** of the **Bon App-étit** application needs to support **3 entities**:

### User

* Has an **Id – "UUID-String" or Long -> long**
* Has a Username (unique, not null)
  + Username length must be between 3 and 20 characters (inclusive of 3 and 20).
* Has a Password (not null)
  + Password length must be between 3 and 20 characters (inclusive of 3 and 20).
* Has an Email (unique, not null)
  + Must contain '@'.
* Has an addedRecipes -> mapped with addedBy
  + A collection that contains all recipes that the user added. One user may have many recipes and one recipe can be added by only one user.
* Has a favouriteRecipes
  + A collection that contains recipes that the user marked as favourite. One user may have many recipes and one recipe can be marked as favourite by many users.

Next ->

### Recipe

* Has an **Id – "UUID-String" or Long**
* Has a Name (not null)
  + Name length must be between 2 and 40 characters (inclusive of 2 and 40).
* Has Ingredients
  + Translation length must be between 2 and 150 characters (inclusive of 2 and 150).
* Has a Category (not null) – има си определени категории!!!
  + One recipe has one caegory and one category can have many recipes.
* Has an addedBy
  + The user who has added the recipe in the app. One recipe has one user, but one user may have added many recipe.

### Category

* Has an **Id** – **"UUID-String" or Long**
* Has a **Category name** (unique, not null)
  + an option between (MAIN\_DISH, DESSERT, COCKTAIL)
* Has a Description
  + For MAIN\_DISH - "Heart of the meal, substantial and satisfying; main dish delights taste buds."
  + For DESSERT – "Sweet finale, indulgent and delightful; dessert crowns the dining experience with joy."
  + For COCKTAIL – "Sip of sophistication, cocktails blend flavors, creating a spirited symphony in every glass."
* Has collection of **Recipes**
  + One category may **have many** recipes, but one recipe has **only one** category.

Implement the entities with the **correct data types** and implement **repositories** for them.

Да не забравим в конструкторите да сложим нови листове или сетове, ако имаме колекции –

public User() {  
 this.addedRecipes = new ArrayList<>();  
 this.favouriteRecipes = new ArrayList<>();  
}

И гетъри и сетъри!!!

Next -> in pom

password: the best23  
username: root

## Initialize categories

* Implement a method that checks (when the app started) if the database does not have **categories** and   
  initialize them
  + **You are free to do it in different ways.**

### CommandLineRunner

* Това е интерфейс на Spring Boot, който позволява **изпълнение на код при стартиране на приложението**.
* Методът run(...) се извиква веднага след като Spring Boot стартира.

Next ->

@Component  
public class InitService implements CommandLineRunner {  
  
 private final Map<CategoryName, String> descriptions = Map.*of*(  
 CategoryName.*MAIN\_DISH*, "Heart of the meal, substantial and satisfying; main dish delights taste buds.",  
 CategoryName.*DESSERT*, "Sweet finale, indulgent and delightful; dessert crowns the dining experience with joy.",  
 CategoryName.*COCKTAIL*, "Sip of sophistication, cocktails blend flavors, creating a spirited symphony in every glass.");  
  
 private final CategoryRepository categoryRepository;  
  
 public InitService(CategoryRepository categoryRepository) {  
 this.categoryRepository = categoryRepository;  
 }  
  
 @Override  
 public void run(String... args) throws Exception {  
 long count = this.categoryRepository.count();  
  
 if (count > 0) {  
 return;  
 }  
  
  
 List<Category> toInsert = Arrays.*stream*(CategoryName.*values*())  
 .map(cat -> new Category(cat, descriptions.get(cat)))  
 .toList();  
  
 this.categoryRepository.saveAll(toInsert);  
  
 }  
}

Next ->

@Repository  
public interface CategoryRepository extends JpaRepository<Category, Long> {  
  
}

Next -> add

@OneToMany(mappedBy = "category")  
private List<Recipe> recipes;  
  
  
  
  
public Category() {  
 this.recipes = new ArrayList<>();  
}  
  
public Category(CategoryName name, String description) {  
 this();  
  
 this.name = name;  
 this.description = description;  
}

And RUN

And check in Mysql tables

And again run to see wether we have only 3 info in table category. IF - Its ok.

8080

### / Защо се прави всичко това?

* **Инициализация на данни**: Когато стартирате приложението за първи път или когато базата данни е празна, този код добавя основни данни в базата (например категориите). Това е удобен начин да се уверите, че имате нужните данни, без да се налага да ги добавяте ръчно.
* **Проверка за дублиране**: Чрез проверката дали категорията вече съществува, този код предотвратява дублирането на записи при многократно стартиране на приложението. Това е полезно в ситуации, когато базата данни вече е била попълнена веднъж, и не е нужно да се изпълнява отново инициализацията./

Представи си, че имаш приложение, което трябва да предлага рецепти и тези рецепти трябва да бъдат категоризирани. Ако стартираш приложението за първи път, този код ще попълни основните категории като "Основни ястия", "Десерти" и "Коктейли". След като категориите са записани веднъж, те няма да бъдат записвани отново при следващи стартирания.

Това е един от начините да гарантираш, че базата данни съдържа всички нужни начални данни, без да се налага да ги въвеждаш ръчно всеки път.

## Page Requirements

Next ->

@Controller  
public class HomeController {  
  
  
 @GetMapping("/")  
 public String nonLoggedIndex(){  
 return "index";  
 }  
  
 @GetMapping("/home")  
 public String loggedInIndex(){  
 return "home";  
 }  
}

**@Controller** – Това е анотация на Spring, която маркира класа като контролер в уеб приложението. Това означава, че този клас ще обработва HTTP заявки и ще връща изгледи (страници).

Този контролер определя две различни начални страници в зависимост от URL:

* Ако потребителят отиде на /, ще се зареди **index.html**.
* Ако потребителят отиде на /home, ще се зареди **home.html**.

Обикновено това се използва за разделяне на съдържанието за **гости** и **влезли потребители**. Например, / може да бъде начална страница за неидентифицирани потребители, а /home – начална страница за тези, които са влезли в системата.

Next->

@Controller  
public class UserController {  
  
 @GetMapping("/register")  
 public String register(){  
 return "/register";  
 }  
  
 @PostMapping("/register")  
 public String doRegister(){  
 return "redirect:/register";  
 }  
}

And RUN to se page register

## ****Малко обобщение за следващите стъпки****

✔ **Всички въведени от потребителя данни минават през контролера**.  
✔ **Spring Boot автоматично свързва тези данни с HTML полетата**.  
✔ **Данните се подават на Java обект (DTO), който контролерът обработва**.  
✔ **След това можем да запишем данните в база или да направим други действия**.

Next -> in register html

<form class="mx-auto w-25 bg-blur rounded"  
 th:method="POST"  
 th:action="@{/register}"  
th:object="${registerData}">  
  
 <div class="form-group">  
 <div class="label-holder textCol d-flex justify-content-center">  
 <label for="username" class="h4 mb-2 text-white">Username</label>  
 </div>  
  
 *<!-- Example: Render errors messages with th:errorclass "is-invalid" and class "invalid-feedback" - you can make it differently. -->  
 <!-- <input th:field="\*{username}" th:errorclass="'is-invalid alert-danger'" type="text" class="form-control"-->  
 <!-- id="username" name="username"/>-->  
 <!-- <small class="invalid-feedback text-danger">Username error message</small>-->* <input type="text" class="form-control" id="username" name="username" th:field="\*{username}"/>  
 <small class="text-danger">Username error message</small>  
 </div>  
  
 <div class="form-group">  
 <div class="label-holder text-white textCol d-flex justify-content-center">  
 <label for="email" class="h4 mb-2">Email</label>  
 </div>  
 <input type="email" class="form-control" id="email" name="email" th:field="\*{email}"/>  
 <small class="text-danger">Email error message</small>  
 </div>

За паролите няма нужда!

Next->

@Controller  
public class UserController {  
  
 @ModelAttribute("registerData")  
 public UserRegisterDTO registerDTO(){  
 return new UserRegisterDTO();  
 }  
  
 @GetMapping("/register")  
 public String register(){  
 return "/register";  
 }  
  
 @PostMapping("/register")  
 public String doRegister(){  
 return "redirect:/register";  
 }  
}

Next->

public class UserRegisterDTO {  
  
 @Size(min = 3, max = 20)  
 @NotBlank  
 private String username;  
 @NotBlank  
 @Email  
 private String email;  
  
 @Size(min = 3, max = 20)  
 @NotBlank  
 private String password;  
  
   
 private String confirmPassword;

And contsructor and getter and setter!!!

Next-> here ->

@PostMapping("/register")  
public String doRegister(@Valid UserRegisterDTO data){  
 return "redirect:/register";  
}

And debug on point return "redirect:/register";

And have to see in console some data for all

Next ->

@PostMapping("/register")  
 public String doRegister(@Valid UserRegisterDTO data, BindingResult bindingResult, RedirectAttributes redirectAttributes){  
  
 *//Validate Data  
 //Return Errors  
 //Register user* if (bindingResult.hasErrors() || !data.getPassword().equals(data.getConfirmPassword()

){  
 *// пишем да се визуализира какво е въвел потребителя* redirectAttributes.addFlashAttribute("registerData", data);  
 redirectAttributes.addFlashAttribute(  
 "org.springframework.validation.BindingResult.registerData", bindingResult);  
 return "redirect:/register";  
 }  
  
  
 return "redirect:/login";  
 }  
}

And debug on points return "redirect:/register"; return "redirect:/login";

And have to see in console some error

And play in console on left button play there

And see in page register wrong username and just email but not password and next field.Its ok!

Next-> Put for all – name, email, password, comfirm password

<input type="text" class="form-control" id="username" name="username" th:field="\*{username}" th:errorclass="'is-invalid alert-danger'"/>

Next-> Put for – name, email, password

<input type="text" class="form-control" id="username" name="username" th:field="\*{username}" th:errorclass="'is-invalid alert-danger'"/>  
<small class="invalid-feedback text-danger">Username error message</small>

And RUN with wrong username only

And see on page register wrong name in red color and some warning. Its OK.

Next ->

if (bindingResult.hasErrors() || !data.getPassword().equals(data.getConfirmPassword())){  
 *// пишем да се визуализира какво е въвел потребителя* redirectAttributes.addFlashAttribute("registerData", data);  
 redirectAttributes.addFlashAttribute(  
 "org.springframework.validation.BindingResult.registerData", bindingResult);  
 return "redirect:/register";  
 }  
  
 userService.register(data);  
  
 return "redirect:/login";  
 }  
}

Next ->

@Controller  
public class UserController {  
  
 private UserService userService;  
  
 public UserController(UserService userService) {  
 this.userService = userService;  
 }  
  
 @ModelAttribute("registerData")  
 public UserRegisterDTO registerDTO(){  
 return new UserRegisterDTO();  
 }

Next ->

@Service  
public class UserService {  
  
 private final UserRepository userRepository;  
  
 public UserService(UserRepository userRepository) {  
 this.userRepository = userRepository;  
 }  
  
 public boolean register(UserRegisterDTO data) {  
 return false;  
  
 }  
}

@Repository  
public interface UserRepository extends JpaRepository<User, Long> {  
  
  
}

Next ->

@Service  
public class UserService {  
  
 private final UserRepository userRepository;  
  
 private final PasswordEncoder passwordEncoder;  
  
 public UserService(UserRepository userRepository, PasswordEncoder passwordEncoder) {  
 this.userRepository = userRepository;  
 this.passwordEncoder = passwordEncoder;  
 }  
  
  
 public boolean register (UserRegisterDTO data) {  
  
 *//Проверка дали има такъв потребител или имейл* Optional<User> existingUser = userRepository.findByUsernameOrEmail(data.getUsername(), data.getEmail());  
  
 if (existingUser.isPresent()){  
 return false;  
 }  
  
 *//Ако няма такъв потребител, запиши го в репо заедно с данните му* User user = new User();  
  
 user.setUsername(data.getUsername());  
 user.setEmail(data.getEmail());  
 user.setPassword(passwordEncoder.encode(data.getPassword()));  
  
 this.userRepository.save(user);  
  
 return true;  
  
 }  
}

Next ->

@Configuration  
public class Config {  
  
 @Bean  
 public PasswordEncoder encoder(){  
 return new BCryptPasswordEncoder();  
 }  
}

And debug with correct data all

And see in mysql data. Its OK. Още нямаме страницата с login.

ИНФО –

Методът **passwordEncoder.encode(data.getPassword())** се използва за криптиране (или хеширане) на паролата на потребителя преди да се запише в базата данни. Това е важна стъпка за сигурността, защото не трябва да съхраняваме паролите в тяхната открита форма, а трябва да използваме криптиране, за да ги направим неразпознаваеми за всеки, който има достъп до базата данни.

### Разбивка на кода:

#### 1. passwordEncoder.encode(data.getPassword()):

* **passwordEncoder** е обикновено инстанция на клас, който имплементира интерфейса за криптиране на пароли, например BCryptPasswordEncoder от Spring Security.
* **.encode(data.getPassword())** извършва криптиране на паролата, която потребителят е въвел (получена чрез **data.getPassword()**).
* След като паролата бъде криптирана, тя ще бъде записана в базата данни като хеширана стойност (не като обикновен текст), така че дори ако някой получи достъп до базата данни, няма да може да види реалната парола на потребителя.

#### 2. user.setPassword(passwordEncoder.encode(data.getPassword())):

* Това задава криптираната версия на паролата в обекта **user**, който ще бъде записан в базата данни.

### Как работи това в контекста на Spring Security?

В **Spring Security** има различни имплементации за криптиране на пароли, като **BCryptPasswordEncoder** е една от най-популярните. Ето как може да изглежда конфигурацията за passwordEncoder:

#### Пример за конфигуриране на BCryptPasswordEncoder в Spring:

java

CopyEdit

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder(); // Може да използваш и друга имплементация, ако предпочиташ

}

С тази конфигурация passwordEncoder ще бъде инжектиран в твоите класове, като например **UserService**.

### Защо е важно да използваш passwordEncoder.encode()?

1. **Сигурност**:
   * Никога не трябва да съхраняваш пароли като обикновен текст (plain text) в базата данни. Ако базата данни бъде компрометирана, паролите на потребителите ще бъдат изложени на риск.
   * Хеширането на паролите е безопасен начин да ги съхраняваш. Хеширането е еднопосочен процес, което означава, че не можеш да възстановиш оригиналната парола от нейния хеш. Това прави паролата безопасна за съхранение.
2. **Бързо и лесно сравняване**:
   * Когато потребителят се опита да влезе в системата, системата ще извърши същото криптиране на паролата, която той е въвел, и ще сравни резултата с хешираната версия в базата данни.
   * Ако паролите съвпадат, тогава потребителят ще бъде автентифициран.

Next -> in UserController ->

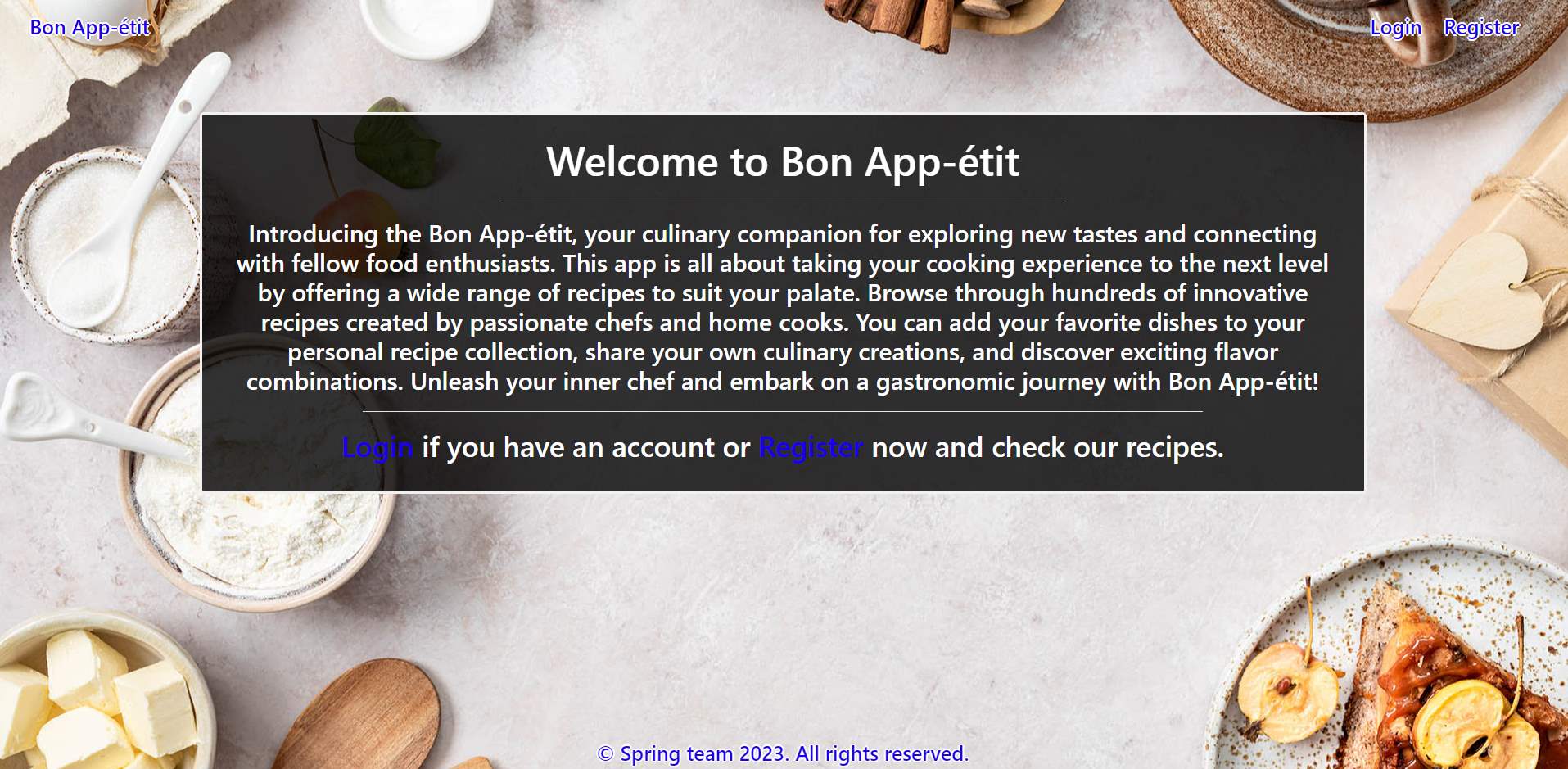
@GetMapping("/login")  
public String login(){  
  
 return "login";  
}  
  
@PostMapping("/login")  
*//Анотацията @Valid активира автоматична валидация според анотациите в UserLoginDTO (например @NotBlank или @Size*public String doLogin(@Valid UserLoginDTO data, BindingResult bindingResult, RedirectAttributes redirectAttributes) {  
  
 if (bindingResult.hasErrors()) {  
  
 redirectAttributes.addFlashAttribute("loginData", data);  
 redirectAttributes.addFlashAttribute(  
 "org.springframework.validation.BindingResult.loginData", bindingResult);  
  
 return "redirect:/login";  
 }  
  
 boolean success = userService.login(data);  
 if (!success){  
 redirectAttributes.addFlashAttribute("loginError", true);  
  
 return "redirect:/login";  
 }  
  
 return "redirect:/home";  
  
}

Next-> in login html

<form class="mx-auto w-25 bg-blur"  
 th:action="@{/login}" th:method="POST" th:object="${loginData}"

<div th:if="${loginError}" class="text-danger">Error messages</div>  
<div th:if="${loginError}" class="text-danger">Incorrect username or password!</div>  
  
<div class="form-group">  
 <div class="label-holder text-white d-flex justify-content-center">  
 <label for="username" class="h3 mb-2">Username</label>  
 </div>  
 <input type="text" class="form-control" id="username" name="username" th:field="\*{username}"/>  
</div>

### Index Page (logged out user)



Next -> in UserService

public boolean login(UserLoginDTO data) {  
  
 Optional<User> byUsername = userRepository.findByUsername(data.getUsername());  
  
 if (byUsername.isEmpty()){  
 return false;  
 }  
  
 boolean passMatch = passwordEncoder.matches(data.getPassword(), byUsername.get().getPassword());  
  
 if (!passMatch){  
 return false;  
 }  
  
 userSession.login(byUsername.get().getId(), data.getUsername());  
  
 return true;  
}

Next ->

package com.bonappetit.config;  
  
import org.springframework.stereotype.Component;  
import org.springframework.web.context.annotation.SessionScope;

@Component  
@SessionScope  
public class UserSession {  
  
 private long id;  
 private String username;  
  
 public void login(long id, String username){  
 this.id = id;  
 this.username = username;  
  
 }  
}

Next->

public class UserLoginDTO {  
  
  
 @NotBlank  
 @Size(min = 3, max = 20)  
 private String username;  
  
 private String password;

And t.n

Next ->

@Repository  
public interface UserRepository extends JpaRepository<User, Long> {  
  
  
 Optional<User> findByUsernameOrEmail(String username, String email);  
  
 Optional<User> findByUsername(String username);  
  
}

And RUN with correct name and password and we see page home

ИНФО-

## ****Защо ни е нужна**** UserSession****?****

1️⃣ **Запазва информация за текущия потребител**

* След като потребителят се логне, UserSession съхранява неговите данни.
* Това позволява други части от приложението да знаят кой е в момента вписан.

2️⃣ **Не е нужно да правим заявки към базата за всеки HTTP request**

* Без UserSession трябваше при всяка заявка да проверяваме в базата кой потребител е вписан.
* С @SessionScope, запазваме тази информация в паметта на сървъра и ускоряваме работата на приложението.

3️⃣ **Лесно проверяваме дали потребителят е влязъл**

* Ако **id == 0** и **username == null**, значи **няма вписан потребител**.
* Ако id и username са зададени, знаем, че има активен потребител.

Този метод **съхранява** ID-то и потребителското име в обекта userSession.  
➡️ Това означава, че след успешно влизане, UserSession съдържа информация за текущия потребител и може да се използва в цялото приложение.

* По подразбиране – сесията приключва при затваряне на браузъра
* Може да приключи и при неактивност (timeout)

server.servlet.session.timeout=30m

Кой го задава това и къде?

Това се задава в **конфигурационния файл на Spring Boot**. Можеш да го сложиш в:

### ****В**** application.properties ****(ако използваш properties формат)****

server.servlet.session.timeout=30m

* Явно прекратяване на сесията (logout)

Защо и id взимаме, нали имаме името му от базата данни, което сме намерили?

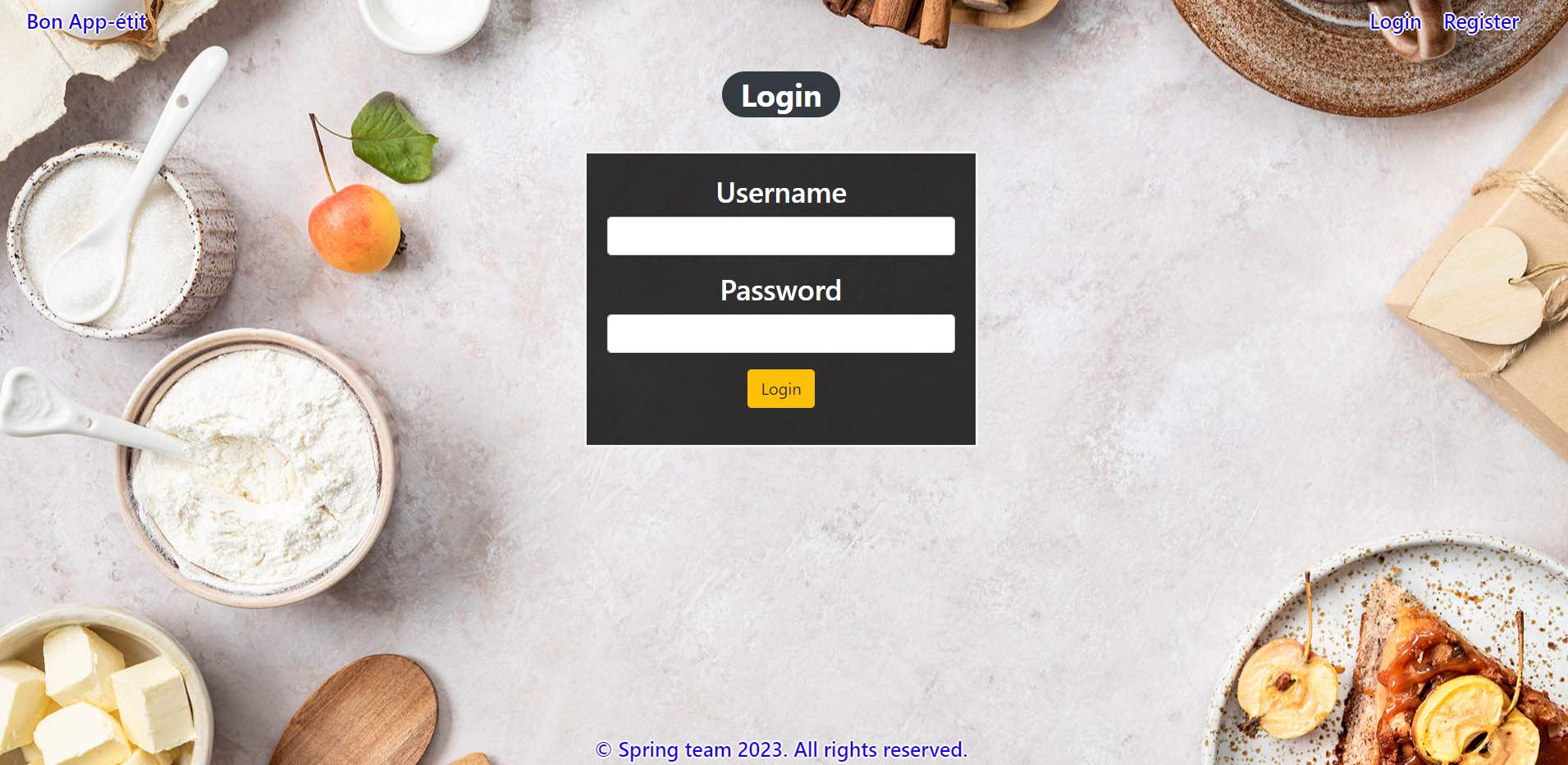
### ****ID-то е уникален идентификатор****

* В базата данни потребителското име **може да не е уникално** (в зависимост от конфигурацията), но **ID-то винаги е уникално**.
* Ако в бъдеще позволим промяна на потребителското име, **ID-то няма да се промени**, така че можем да го използваме като **стабилен начин за идентификация**.

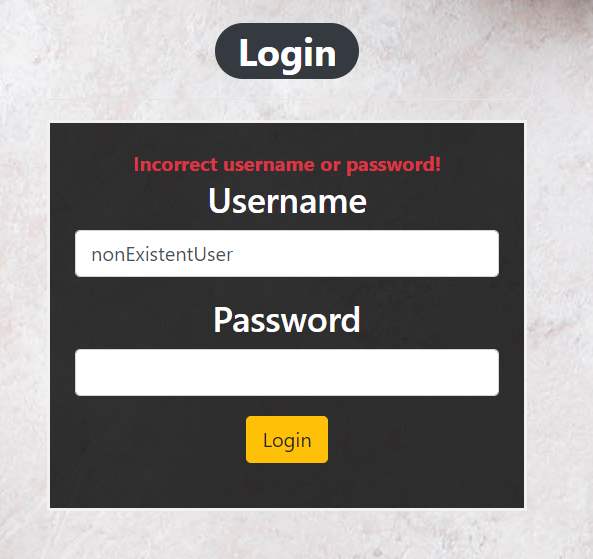
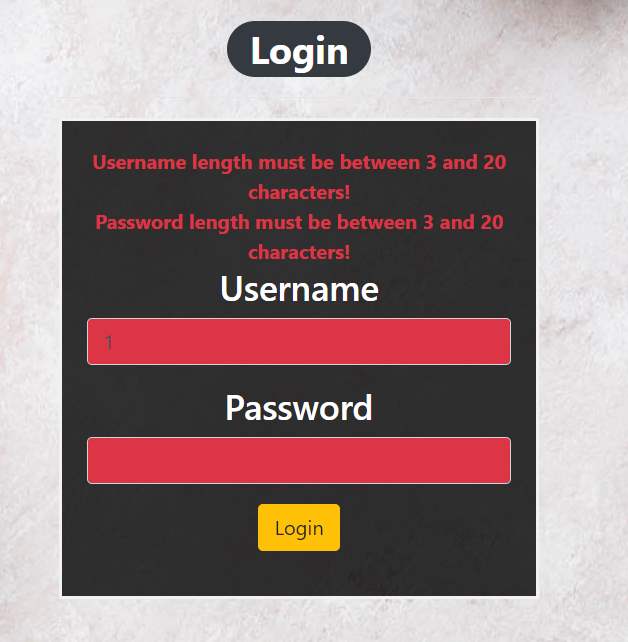
### ****2️⃣ Лесен достъп до потребителя в базата****

* Ако искаме да извлечем данни за текущия потребител в други части на приложението, можем да използваме userSession.getId(), за да го намерим по **ID**, което е много по-бързо и ефективно.

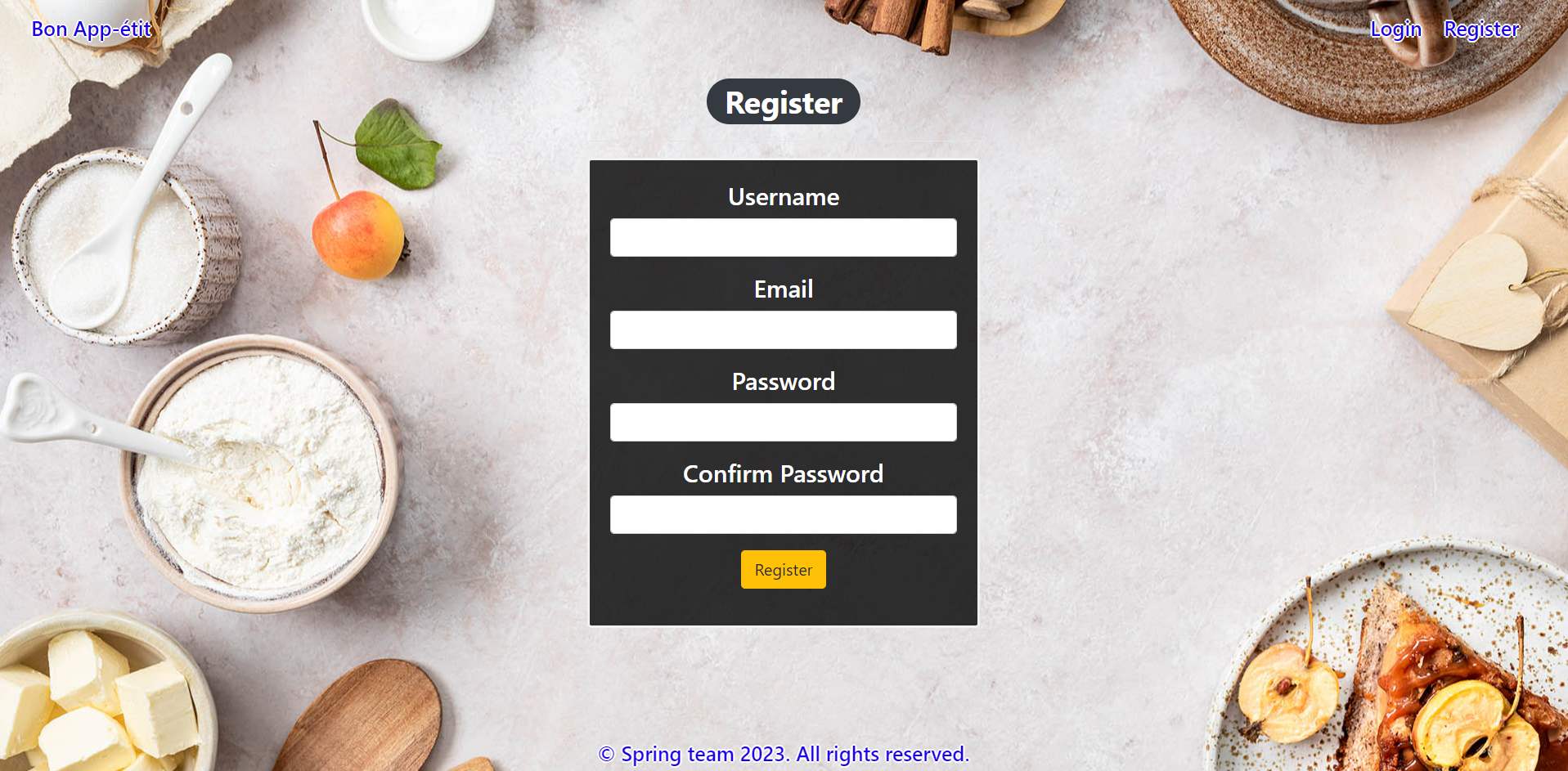
### Login Page (logged out user)



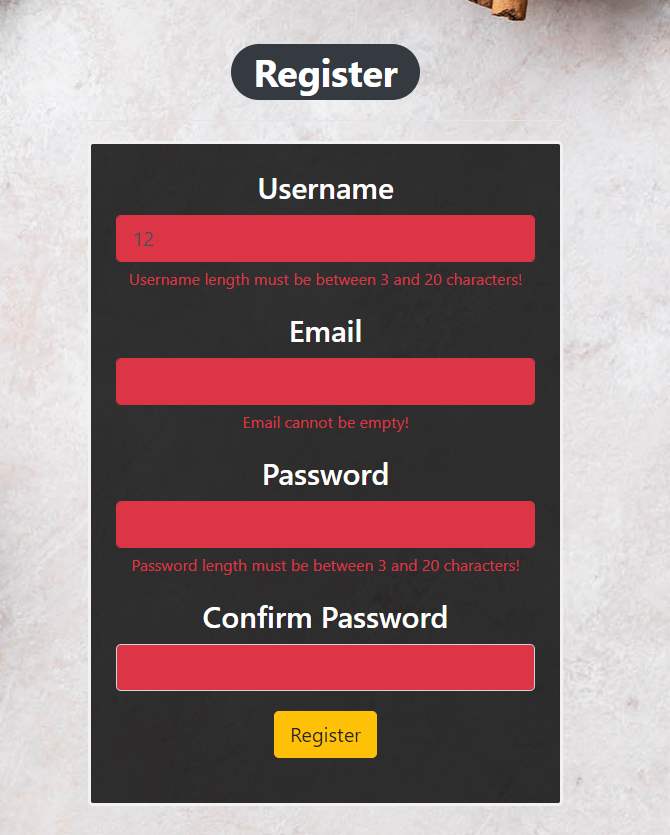
### Login Page validations

### Register Page (logged out user)



### Register Page validations

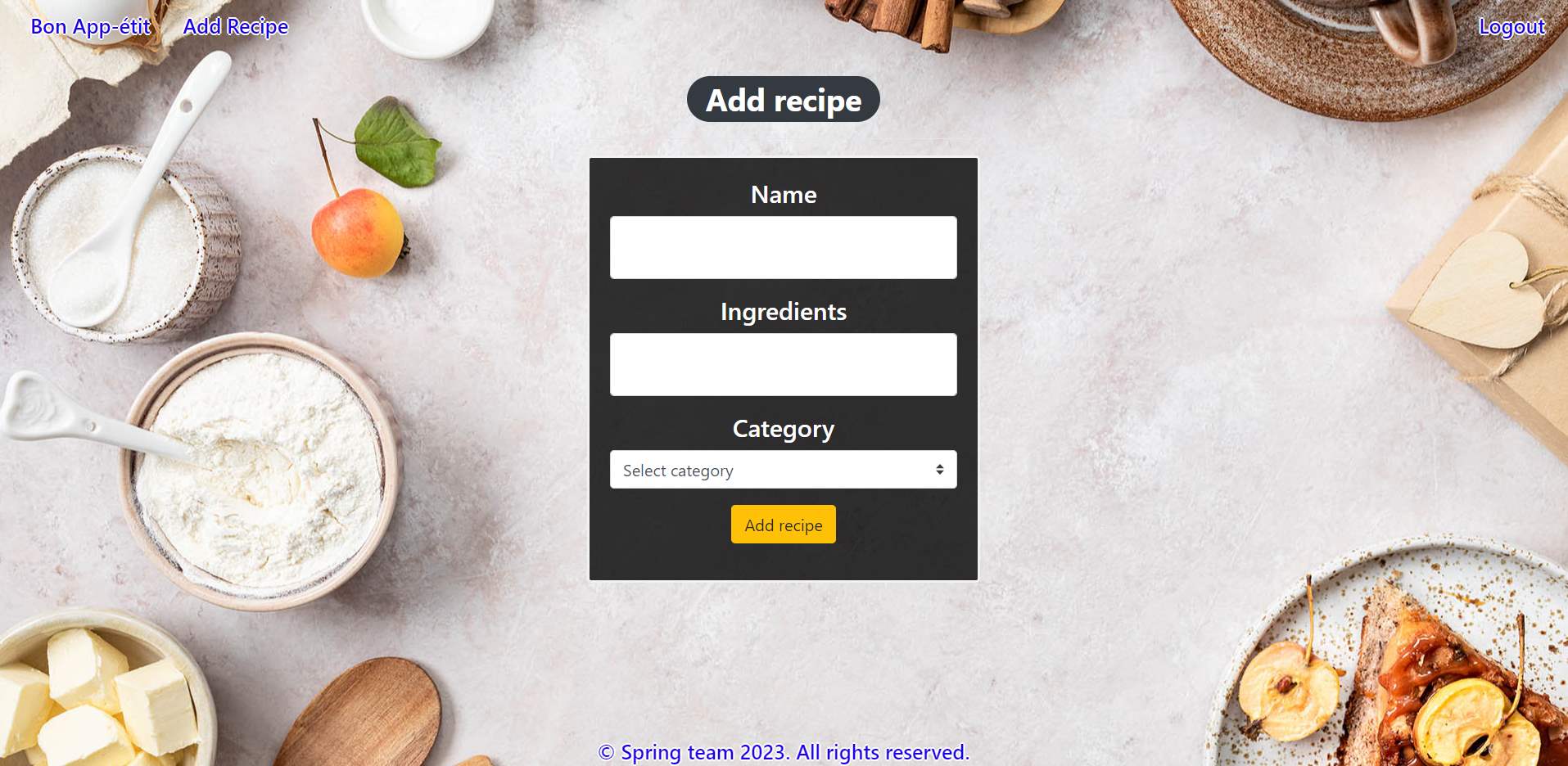


### Home Page (without having any recipes)

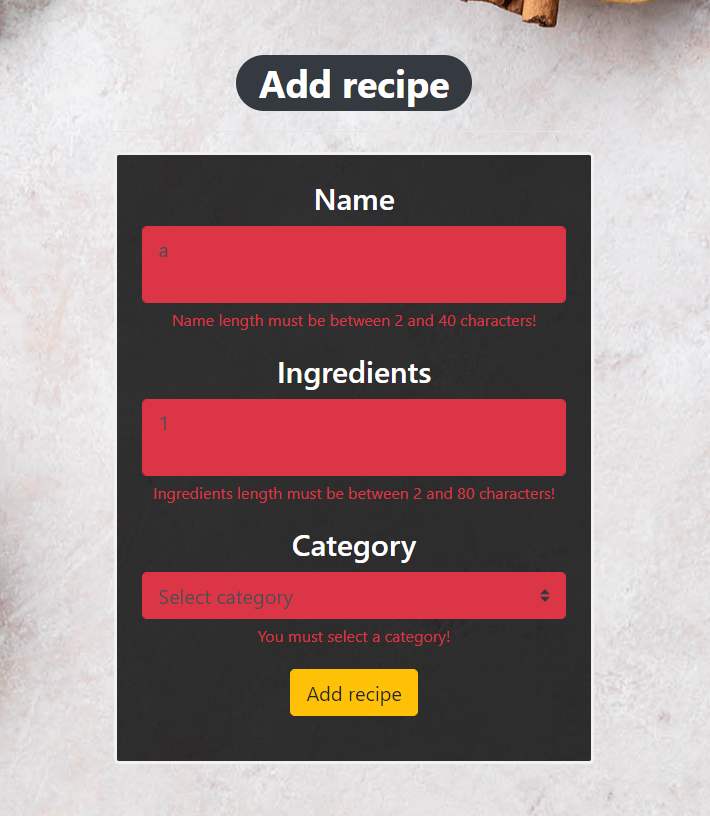
* The upper left section (Desserts) of the page should display **all of the added recipes for Desserts** from the database from all users.
* The upper right section (Main Dishes) of the page should display **all of the added recipes for Main Dishes** from the database from all users.
* The bottom left section (Cocktails) of the page should display **all of the added recipes for Cocktails** from the database from all users.
* The bottom right section (Favourites) of the page should display **all recipes marked as favorites** in the database by users.



### Add Recipe



### Add Recipe validation



### Home Page (with recipes)



The templates have been given to you in the application skeleton, so make sure you implement the pages correctly.

**NOTE**: The templates should look **EXACTLY** as shown above.

**NOTE**: The templates do **NOT** **require** **additional** **CSS** for you to write. Only **bootstrap** and the **given CSS** are enough.

## Functional Requirements

The Functionality Requirements describe the functionality that the application must support.

The application should provide **Guest** (not logged in) users with the functionality to log in, register and view the Index page.

The application should provide **Users** (logged in) with the functionality to **log out, add a new recipe (Add Recipe page), view all recipes (Home page) and add recipes to Favourites section.**

Bon App-étit in navbar should **redirect** to the appropriate URL depending on that if the **user is logged in**.

The application should provide functionality for **adding recipes** with category of **Main Dish, Dessert or Cocktail**.

On the **Home Page**, recipes should be **organized** into their respective categories: main dish recipes in the **Main Dishes** section, dessert recipes in the **Desserts** section, cocktail recipes in the **Cocktails** section, and favorite recipes in the **Favorites** section.

**Above** these sections, a **greeting text** displaying the current user's username is located: "Welcome, {username}".

Next to the name of each section on the **Home page** is a **counter** indicating the **number of recipes** in the current section.

Each recipe is accompanied by a **star with a plus icon**. When **clicked**, the selected recipe stays in the current section but is also **added** to the current user's **favourite recipes** collection. If the recipe is already present in the favorites collection, no action should be taken.

The application should store its data in a MySQL database.

## Security Requirements

Next-> in commons html ->

<header>  
 <nav class="navbar navbar-expand-lg bg-nav rounded">  
 <a class="nav-link text-primary active h5 text-stroke" th:href="@{/}">Bon App-étit</a>  
 <a class="nav-link text-primary active h5 text-stroke" th:href="@{/add-recipe}">Add Recipe</a>  
  
 <div class="collapse navbar-collapse d-flex justify-content-end" id="navbarNav">  
 <ul class="navbar-nav row">  
 <li class="nav-item col-md-4">  
 <a class="nav-link text-primary active h5 text-stroke" th:href="@{/login}">Login</a>  
 </li>  
  
 <li class="nav-item col-md-4">  
 <a class="nav-link text-primary active h5 text-stroke" th:href="@{/register}">Register</a>  
 </li>  
  
 <li class="nav-item col-md-4">  
 <a class="nav-link text-primary active h5 text-stroke" href="/">Logout</a>  
 </li>  
 </ul>  
 </div>  
 </nav>  
</header>

Сега ще укажем кои страници да се показват в зависимост дали е логнат или не потребителя –

<header>  
 <nav class="navbar navbar-expand-lg bg-nav rounded">  
 <a class="nav-link text-primary active h5 text-stroke" th:href="@{/}">Bon App-étit</a>  
 <a class="nav-link text-primary active h5 text-stroke" th:href="@{/add-recipe}"  
 th:if="${@userSession.isLoggedIn()}"  
 >Add Recipe  
 </a>  
  
 <div class="collapse navbar-collapse d-flex justify-content-end" id="navbarNav">  
 <ul class="navbar-nav row">  
 <li class="nav-item col-md-4"  
 th:unless="${@userSession.isLoggedIn()}"  
 >  
 <a class="nav-link text-primary active h5 text-stroke" th:href="@{/login}">Login</a>  
 </li>  
  
 <li class="nav-item col-md-4"  
 th:unless="${@userSession.isLoggedIn()}"  
 >  
 <a class="nav-link text-primary active h5 text-stroke" th:href="@{/register}">Register</a>  
 </li>  
  
 <li th:if="${@userSession.isLoggedIn()}" class="nav-item col-md-4">  
 <a class="nav-link text-primary active h5 text-stroke" href="/">Logout</a>  
 </li>  
 </ul>  
 </div>  
 </nav>  
</header>

Next->

@Component  
@SessionScope  
public class UserSession {  
  
 private long id;  
 private String username;  
  
 public void login(long id, String username){  
 this.id = id;  
 this.username = username;  
  
 }  
  
 public boolean isLoggedIn(){  
 return id > 0;  
 }  
}

And RUN and login with data and tick button login, and see page home. There see button logut, bonapeti, add-recipe.

The Security Requirements are mainly access requirements. Configurations about which users can access specific functionalities and pages.

* Guest (not logged in) users can access the Index page.
* Guest (not logged in) users can access the Login page.
* Guest (not logged in) users can access the Register page.
* Users (logged in) can access the Home page.
* Users (logged in) can access Add Recipe page.
* Users (logged in) can access Logout functionality.

Но все още нямаме page add recipe!!!

Next ->

@Controller  
public class RecipeController {  
  
  
 @GetMapping("/add-recipe")  
 public String addRecipe(){  
 return "/recipe-add";  
 }  
  
 @PostMapping("/add-recipe")  
 public String doAddRecipe(@Valid AddRecipeDTO data, BindingResult bindingResult, RedirectAttributes redirectAttributes) {  
  
  
 if (bindingResult.hasErrors()) {  
  
 redirectAttributes.addFlashAttribute("recipeData", data);  
 redirectAttributes.addFlashAttribute(  
 "org.springframework.validation.BindingResult.recipeData", bindingResult);  
 return "redirect:/add-recipe";  
 }  
  
 return "redirect:/home";  
 }  
}

Next -> in recipe-add html

<header th:replace="fragments/commons::nav"></header>  
<main>  
 <h1 class="text-center mt-2"><span class="badge badge-pill badge-dark">Add recipe</span></h1>  
 <hr class="hr-3 w-25 bg-white"/>  
  
 <form class="mx-auto w-25 bg-blur rounded" th:action="@{/add-recipe}" th:method="POST" th:object="${recipeData}">  
  
 <div class="form-group">  
 <div class="label-holder text-white textCol d-flex justify-content-center">  
 <label for="name" class="h4 mb-2">Name</label>  
 </div>  
 <textarea type="text" class="form-control" id="name" name="name" th:field="\*{name}"></textarea>  
 <small class="text-danger">Name error message</small>  
 </div>  
  
 <div class="form-group">  
 <div class="label-holder text-white textCol d-flex justify-content-center">  
 <label for="ingredients" class="h4 mb-2">Ingredients</label>  
 </div>  
 <textarea type="text" class="form-control" id="ingredients" name="ingredients" th:field="\*{ingredients}"></textarea>  
 <small class="text-danger">Ingredients error message</small>  
 </div>  
  
 <div class="form-group">  
 <div class="text-white label-holder d-flex justify-content-center">  
 <label for="category" class="h4 mb-2">Category</label>  
 </div>  
 <select class="browser-default custom-select" id="category" name="category" th:field="\*{category}">  
 <option value="">Select category</option>  
 <option value="MAIN\_DISH">Main Dish</option>  
 <option value="DESSERT">Dessert</option>  
 <option value="COCKTAIL">Cocktail</option>  
 </select>  
  
 <small class="text-danger">Category error message</small>  
 </div>  
  
 <div class="button-holder d-flex justify-content-center">  
 <button type="submit" class="btn btn-warning mb-3" value="Register">Add recipe</button>  
 </div>  
  
 </form>

Next ->

@Controller  
public class RecipeController {  
  
 @ModelAttribute("recipeData")  
 public AddRecipeDTO recipeData (){  
  
 return new AddRecipeDTO();  
 }

Next ->

public class AddRecipeDTO {  
  
 @NotNull  
 @Size(min = 2, max = 40)  
 private String name;  
  
 @NotNull  
 @Size(min = 2, max = 150)  
 private String ingredients;  
  
 @NotNull  
 private CategoryName category;

And DEBUG here return "redirect:/add-recipe";  
 }  
}

And write correct data and see in console all data.

Next -> in RecipeController

private final RecipeService recipeService;  
  
public RecipeController(RecipeService recipeService) {  
 this.recipeService = recipeService;  
}

recipeService.create(data);  
  
 return "redirect:/add-recipe";  
 }  
}

Next ->

@Service  
public class RecipeService {  
  
 private final RecipeRepository recipeRepository;

private final UserSession userSession;  
  
 public RecipeService(RecipeRepository recipeRepository, UserSession userSession) {  
 this.recipeRepository = recipeRepository;  
 this.userSession = userSession;  
 }  
  
  
 public boolean create(AddRecipeDTO data) {  
 return false;  
 }

Next ->

@Repository  
public interface RecipeRepository extends JpaRepository<Recipe, Long> {  
  
  
}

Next -> we put errors messages ->

<textarea type="text" class="form-control" id="name" name="name" th:field="\*{name}" th:errorclass="'is-invalid alert-danger'"></textarea>  
 <small class="invalid-feedback text-danger">Name error message</small>  
</div>

<textarea type="text" class="form-control" id="ingredients" name="ingredients" th:field="\*{ingredients}" th:errorclass="'is-invalid alert-danger'"></textarea>  
 <small class="invalid-feedback text-danger">Ingredients error message</small>  
</div>

<select class="browser-default custom-select" id="category" name="category" th:field="\*{category}" th:errorclass="'is-invalid alert-danger'">  
 <option value="">Select category</option>  
 <option value="MAIN\_DISH">Main Dish</option>  
 <option value="DESSERT">Dessert</option>  
 <option value="COCKTAIL">Cocktail</option>  
 </select>  
  
 <small class="invalid-feedback text-danger">Category error message</small>  
</div>

Next ->

@Service  
public class RecipeService {  
  
 private final RecipeRepository recipeRepository;  
 private final UserSession userSession;  
  
 private final UserRepository userRepository;  
  
 private final CategoryRepository categoryRepository;

public boolean create(AddRecipeDTO data) {  
  
 if (!userSession.isLoggedIn()){  
 return false;  
  
 }  
  
 Optional<User> byId = userRepository.findById(userSession.id());  
  
 if (byId.isEmpty()){  
 return false;  
 }  
  
 Optional<Category> byName = categoryRepository.findByName(data.getCategory());  
  
 if (byName.isEmpty()){  
 return false;  
 }  
  
 Recipe recipe = new Recipe();  
 recipe.setName(data.getName());  
 recipe.setIngredients(data.getIngredients());  
 recipe.setCategory(byName.get());  
  
 recipe.setAddedBy(byId.get());  
  
 recipeRepository.save(recipe);  
  
 return true;  
 }  
}

Next->

@Repository  
public interface CategoryRepository extends JpaRepository<Category, Long> {  
  
 Optional<Category> findByName(CategoryName name);  
}

Next->

@Component  
@SessionScope  
public class UserSession {  
  
 private long id;  
 private String username;  
  
 public void login(long id, String username){  
 this.id = id;  
 this.username = username;  
  
 }  
  
 public Long id() {  
 return id;  
 }  
  
 public String username(){  
 return username;  
 }  
  
 public boolean isLoggedIn(){  
 return id > 0;  
 }

And RUN – логваме се, въвеждаме нова рецепта. Ще ни отведе на страница home, но все още няма да виждаме рецептата. Може обаче да видим в mysql данните дали са пристигнали. Трябва да са там.

Next ->

@Controller  
public class RecipeController {  
  
 private final RecipeService recipeService;  
 private final UserSession userSession;

@GetMapping("/add-recipe")  
public String addRecipe(){  
 if (!userSession.isLoggedIn()){  
 return "redirect:/";  
 }  
 return "/recipe-add";  
}  
  
@PostMapping("/add-recipe")  
public String doAddRecipe(@Valid AddRecipeDTO data, BindingResult bindingResult, RedirectAttributes redirectAttributes) {  
  
 if (!userSession.isLoggedIn()){  
 return "redirect:/";  
 }  
  
  
 if (bindingResult.hasErrors()) {  
  
 redirectAttributes.addFlashAttribute("recipeData", data);  
 redirectAttributes.addFlashAttribute(  
 "org.springframework.validation.BindingResult.recipeData", bindingResult);  
 return "redirect:/add-recipe";  
 }

Next->

@Controller  
public class UserController {  
  
 private final UserService userService;  
 private final UserSession userSession;

@GetMapping("/register")  
public String register(){  
  
 if (userSession.isLoggedIn()){  
 return "redirect:/home";  
 }  
  
 return "/register";  
 }  
  
 @PostMapping("/register")  
 public String doRegister(@Valid UserRegisterDTO data, BindingResult bindingResult, RedirectAttributes redirectAttributes){  
  
 if (userSession.isLoggedIn()){  
 return "redirect:/home";  
 }  
  
 *//Validate Data  
 //Return Errors  
 //Register user* if (bindingResult.hasErrors() || !data.getPassword().equals(data.getConfirmPassword())){  
 *// пишем да се визуализира какво е въвел потребителя* redirectAttributes.addFlashAttribute("registerData", data);  
 redirectAttributes.addFlashAttribute(  
 "org.springframework.validation.BindingResult.registerData", bindingResult);  
 return "redirect:/register";  
 }  
  
 boolean success = userService.register(data);  
  
 if (!success){  
 return "redirect:/register";  
 }  
  
 return "redirect:/login";  
 }  
  
 @GetMapping("/login")  
 public String login(){  
  
 if (userSession.isLoggedIn()){  
 return "redirect:/home";  
 }  
  
 return "login";  
 }  
  
 @PostMapping("/login")  
 *//Анотацията @Valid активира автоматична валидация според анотациите в UserLoginDTO (например @NotBlank или @Size* public String doLogin(@Valid UserLoginDTO data, BindingResult bindingResult, RedirectAttributes redirectAttributes) {  
  
 if (userSession.isLoggedIn()){  
 return "redirect:/home";  
 }  
  
 if (bindingResult.hasErrors()) {  
  
 redirectAttributes.addFlashAttribute("loginData", data);  
 redirectAttributes.addFlashAttribute(  
 "org.springframework.validation.BindingResult.loginData", bindingResult);  
  
 return "redirect:/login";  
 }  
  
 boolean success = userService.login(data);  
 if (!success){  
 redirectAttributes.addFlashAttribute("loginError", true);  
  
 return "redirect:/login";  
 }  
  
 return "redirect:/home";  
  
 }  
}

And Run и правим проверки на тези неща – имаме ли достъп и кога до всяка страница според изискванията –

* Guest (not logged in) users can access the Index page.
* Guest (not logged in) users can access the Login page.
* Guest (not logged in) users can access the Register page.
* Users (logged in) can access the Home page.
* Users (logged in) can access Add Recipe page.
* Users (logged in) can access Logout functionality.

Next ->

<header th:replace="fragments/commons::nav"></header>  
<main>  
 <div class="container bg-blur rounded mt-1 mb-4">  
 <h6 class="bg-gray text-dark mb-0 mt-0">Welcome, <span th:text="${@userSession.username()}"></span> </h6>  
 <div class="row mt-1 mb-2">  
 <div class="col-6 mt-1 mb-1">

Next -> in Service

public Map<CategoryName, List<Recipe>> findAllByCategory() {  
 return null;  
}

Next -> HomeController

private final RecipeService recipeService;

@GetMapping("/home")  
 public String loggedInIndex(Model model){  
  
 if (!userSession.isLoggedIn()){  
 return "redirect:/";  
 }  
  
 Map<CategoryName, List<Recipe>> allRecipes = recipeService.findAllByCategory();  
  
model.addAttribute("cocktailsData", allRecipes.get(CategoryName.*COCKTAIL*));

*// recipeService.findFavourities();* return "home";  
 }  
}

Next -> RecipeService

public Map<CategoryName, List<Recipe>> findAllByCategory() {  
  
 Map<CategoryName, List<Recipe>> result = new HashMap<>();  
  
 List<Category> allCategories = categoryRepository.findAll();  
  
 for (Category cat: allCategories) {  
 List<Recipe> recipes = recipeRepository.findAllByCategory(cat);  
  
  
 result.put(cat.getName(), recipes);  
 }  
  
 return result;  
}

Next ->

@Repository  
public interface RecipeRepository extends JpaRepository<Recipe, Long> {  
  
  
 List<Recipe> findAllByCategory(Category category);  
  
}

Next -> in home html – правим да се показва бройката въведени рецепти в дадената категория

<div>  
 <h3 class="mySticky bg-gray text-dark mb-0 mt-2">Cocktails (<span th:text="${cocktailsData.size()}"></span>)</h3>

And RUN – като се логнем и сме на хоум страницата трябва да виждаме брой 1 в категория коктейли, защото сме вече въвели коктейл мохито!

Next -> сега трябва да изпишем инфото на коктейла в тази категория

<div>  
 <h3 class="mySticky bg-gray text-dark mb-0 mt-2">Cocktails (<span th:text="${cocktailsData.size()}"></span>)</h3>  
 <div class="bg-dark rounded m-auto" style="height: 31vh; overflow: auto">  
 <table class="table table-striped table-dark">  
 <tr th:each="cocktail: ${cocktailsData}">  
 <td class="bg-success" scope="row" style="width: 5%">  
 <img class="mr-2 " src="/img/cocktail.png" alt="Cocktail" width="35px"  
 height="35px">  
 </td>  
 <td th:text="${cocktail.getName()}"></td>  
 <td th:text="${cocktail.getIngredients()}"></td>  
 <td><a href="/"><img class="mr-2 " src="/img/add-to-favourites.png" alt="Cocktail" width="35px" height="35px"></a></td>  
 </tr>  
 </table>  
 </div>  
 </div>  
</div>

And RUN – сега трябва да се вижда инфо за коктейла на страницата хоум

Вписваме още един коктейл с инфо в интернет и виждаме , че се добавя и бройката става 2

Next -> правим и бутон за любими рецепти срещу всяка рецепта, като все още няма заявка към него тук

</td>  
 <td th:text="${cocktail.getName()}"></td>  
 <td th:text="${cocktail.getIngredients()}"></td>  
 <td>  
 <form>  
 <button type="submit">  
 <img class="mr-2 " src="/img/add-to-favourites.png" alt="Cocktail" width="35px" height="35px">  
 </button>  
 </form>  
 </td>  
</tr>

Next -> пътят, който точно ни отвежда до id на избраната рецепта

<td>  
 <form th:method="POST" th:action="@{/add-to-favourites/{id}(id=${cocktail.getId()})}">

<button type="submit">  
 <img class="mr-2 " src="/img/add-to-favourites.png" alt="Cocktail" width="35px" height="35px">  
 </button>  
 </form>  
</td>

Next -> RecipeController

@PostMapping("/add-to-favourites/{recipeId}")  
public String addToFavourites(@PathVariable long recipeId){  
   
 *//@PathVariable е анотация, която показва,  
 // че параметърът id трябва да бъде взет от пътната променлива в URL-то.  
 //В този случай, ако потребителят посети URL /add-to-favorites/123,  
 // тогава стойността 123 ще бъде присвоена на параметъра id и методът ще бъде извикан с тази стойност.* return "redirect:/home";  
  
}

Next -> recipeService

public void addToFavourites(Long id, long recipeId) {  
 Optional<User> userOpt = userRepository.findById(recipeId);  
  
 if (userOpt.isEmpty()){  
 return;  
 }  
  
 Optional<Recipe> recipeOpt = recipeRepository.findById(recipeId);  
 if (recipeOpt.isEmpty()){  
 return;  
 }  
  
 userOpt.get().addFavourite(recipeOpt.get());  
  
 userRepository.save(userOpt.get());  
}

Next -> RecipeController

@PostMapping("/add-to-favourites/{recipeId}")  
public String addToFavourites(@PathVariable long recipeId){  
  
 *//@PathVariable е анотация, която показва,  
 // че параметърът id трябва да бъде взет от пътната променлива в URL-то.  
 //В този случай, ако потребителят посети URL /add-to-favorites/123,  
 // тогава стойността 123 ще бъде присвоена на параметъра id и методът ще бъде извикан с тази стойност.* if (!userSession.isLoggedIn()){  
 return "redirect:/";  
 }  
  
 recipeService.addToFavourites(userSession.id(), recipeId);  
  
 return "redirect:/home";  
  
}

Next -> User

public void addFavourite(Recipe recipe) {  
 this.favouriteRecipes.add(recipe);  
  
}

Next -> home Controller

private final UserService userService;

@GetMapping("/home")  
 @Transactional  
 public String loggedInIndex(Model model){  
  
 if (!userSession.isLoggedIn()){  
 return "redirect:/";  
 }  
  
 Map<CategoryName, List<Recipe>> allRecipes = recipeService.findAllByCategory();  
  
  
 List<RecipeInfoDTO> favourites = userService.findFavourites(userSession.id())  
 .stream().map(RecipeInfoDTO::new).toList();  
  
 List<RecipeInfoDTO> cocktails = allRecipes.get(CategoryName.*COCKTAIL*).stream().map(RecipeInfoDTO::new).toList();  
  
 model.addAttribute("cocktailsData", cocktails);  
 model.addAttribute("favouritesData", favourites);  
  
  
 return "home";  
 }  
}

Next -> UserService

//@Transactional

public List<Recipe> findFavourites(Long id) {

*// Потърсете потребителя по id* return userRepository.findById(id)  
 .map(User::getFavouriteRecipes) *// Ако потребителят съществува, вземете любимите рецепти* .orElseGet(ArrayList::new); *// Ако няма потребител, върнете празен списък*}  
  
   
}

Next -> RecipeService

@Transactional  
public void addToFavourites(Long id, long recipeId) {  
 Optional<User> userOpt = userRepository.findById(id);  
  
 if (userOpt.isEmpty()){  
 return;  
 }  
  
 Optional<Recipe> recipeOpt = recipeRepository.findById(recipeId);  
 if (recipeOpt.isEmpty()){  
 return;  
 }  
  
 userOpt.get().addFavourite(recipeOpt.get());  
  
 userRepository.save(userOpt.get());  
}

Next ->

public class RecipeInfoDTO {  
  
 private long id;  
 private String name;  
 private String ingredients;  
  
  
 public RecipeInfoDTO(Recipe recipe) {  
 this.id = recipe.getId();  
 this.name = recipe.getName();  
 this.ingredients = recipe.getIngredients();  
 }

and t.n

Next -> поставяме тифове и на останалите категории, сменяйки само името на категорията и дейтата

<div>  
 <h3 class="mySticky bg-gray text-dark mb-0 mt-2">Desserts (<span th:text="${dessertsData.size()}"></span>)</h3>  
 <div class="bg-dark rounded m-auto" style="height: 31vh; overflow: auto">  
 <table class="table table-striped table-dark">  
 <tr th:each="dessert: ${dessertsData}">  
 <td class="bg-warning" scope="row" style="width: 5%">  
 <img class="mr-2 " src="/img/dessert.png" alt="Dessert" width="35px" height="35px">  
 </td>  
 <td th:text="${dessert.getName()}"></td>  
 <td th:text="${dessert.getIngredients()}"></td>  
 <td>  
 <form th:method="POST" th:action="@{/add-to-favourites/{id}(id=${dessert.getId()})}">  
 <button type="submit">  
 <img class="mr-2 " src="/img/add-to-favourites.png" alt="Cocktail" width="35px" height="35px">  
 </button>  
 </form>  
 </td>  
 </tr>  
 </table>  
 </div>

Next -> homeController

@GetMapping("/home")  
@Transactional  
public String loggedInIndex(Model model){  
  
 if (!userSession.isLoggedIn()){  
 return "redirect:/";  
 }  
  
 Map<CategoryName, List<Recipe>> allRecipes = recipeService.findAllByCategory();  
  
  
 List<RecipeInfoDTO> favourites = userService.findFavourites(userSession.id())  
 .stream().map(RecipeInfoDTO::new).toList();  
  
 List<RecipeInfoDTO> cocktails = allRecipes.get(CategoryName.*COCKTAIL*).stream().map(RecipeInfoDTO::new).toList();  
  
 List<RecipeInfoDTO> mainDishes = allRecipes.get(CategoryName.*MAIN\_DISH*).stream().map(RecipeInfoDTO::new).toList();  
  
 List<RecipeInfoDTO> desserts = allRecipes.get(CategoryName.*DESSERT*).stream().map(RecipeInfoDTO::new).toList();  
  
 model.addAttribute("cocktailsData", cocktails);  
 model.addAttribute("mainDishesData", mainDishes);  
 model.addAttribute("dessertsData", desserts);  
 model.addAttribute("favouritesData", favourites);  
  
  
 return "home";  
}

Next -> in commons

<li th:if="${@userSession.isLoggedIn()}" class="nav-item col-md-4">  
 <a class="nav-link text-primary active h5 text-stroke" th:href="@{/logout}">Logout</a>  
</li>

Next -> User Controller

@GetMapping("/logout")  
 public String logout(){  
  
 if (!userSession.isLoggedIn()){  
 return "redirect:/";  
 }  
  
 userSession.logout();  
  
 return "redirect:/";  
 }  
}

Next -> In User Session

public void logout() {  
  
 id = 0;  
 username = null;  
}

## Scoring

### Database – 10 points.

### Pages – 25 points.

### Functionality – 35 points.

### Security – 5 points.

### Validations – 15 points.

### Code Quality – 10 points.