STUDENT GOTUJE - KOMENTARZE W KODZIE

Dokumentacja techniczna kodu

Promotor projektu:
Prof.dr.hab.inż Piotr Powroźnik.

—

Autorzy:
Karolina Kleciak,
Paweł Łaskarzewski,
Kornel Myczko

Dokumentacja została przygotowana na podstawie analizy kodu programu oraz zawartych w nim komentarzy

`src/main/java/org/example/uzgotuje/config/AppConfig.java`

```
@@ -1,14 +1,23 @@
package org.example.uzgotuje.config;

import org.springframework.boot.web.client.RestTemplateBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

+ /**
+  * Configuration class for application-specific beans.
+  */
@Configuration
public class AppConfig {

+     /**
+      * Creates and configures a {@link RestTemplate} bean.
+      *
+      * @param builder the {@link RestTemplateBuilder} used to build the {@link RestTemplate}
+      * @return a configured {@link RestTemplate} instance
+      */
      @Bean
      public RestTemplate restTemplate(RestTemplateBuilder builder) {
          return builder.build();
      }
```

src/main/java/org/example/uzgotuje/config/CorsConfig.jav

```
@@ -1,28 +1,36 @@
package org.example.uzgotuje.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;

/**
 * Configuration class for setting up CORS (Cross-Origin Resource Sharing) in the application.
 */
@Configuration
public class CorsConfig {

    /**
     * Creates and configures a {@link CorsConfigurationSource} bean.
     *
     * @return a configured {@link CorsConfigurationSource} instance
     */

    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.setAllowedOrigins(List.of("http://localhost:3002")); // Frontend URL
        configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE", "OPTIONS"));
        configuration.setAllowedHeaders(Arrays.asList("Content-Type", "Authorization"));
        configuration.setAllowCredentials(true); // Required for cookies or credentials

        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);
        return source;
    }
}
```

src/main/java/org/example/uzgotuje/config/PasswordEncoderConfig.java

```
@@ -1,13 +1,22 @@
package org.example.uzgotuje.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

+ /**
+  * Configuration class for setting up the password encoder bean.
+  */
@Configuration
public class PasswordEncoderConfig {
+
+     /**
+      * Creates and configures a {@link BCryptPasswordEncoder} bean.
+      *
+      * @return a configured {@link BCryptPasswordEncoder} instance
+      */
     @Bean
     public BCryptPasswordEncoder passwordEncoder() {
         return new BCryptPasswordEncoder();
     }
}
```

src/main/java/org/example/uzgotuje/config/WebSecurityConfig.java

```
@@ -1,51 +1,76 @@
package org.example.uzgotuje.config;

import lombok.AllArgsConstructor;
import org.example.uzgotuje.services.UserService;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

import java.util.Collections;

import static org.springframework.security.config.Customizer.withDefaults;

/**
 * Configuration class for setting up web security in the application.
 */
@Configuration
@AllArgsConstructor
@EnableWebSecurity
public class WebSecurityConfig {
```

```
public class WebSecurityConfig {

    private final UserService userService;
    private final PasswordEncoderConfig passwordEncoder;

    /**
     * Configures the security filter chain.
     *
     * @param http the {@link HttpSecurity} to modify
     * @return a configured {@link SecurityFilterChain} instance
     * @throws Exception if an error occurs while configuring the security filter chain
     */
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
                .csrf(csrf->csrf.disable())
                .authorizeRequests(authorizeRequests->authorizeRequests
                        .requestMatchers("/auth/**","/recipe/**","files/**")
                .csrf(csrf -> csrf.disable())
                .authorizeRequests(authorizeRequests -> authorizeRequests
                        .requestMatchers("/auth/**", "/recipe/**", "files/**")
                        .permitAll()
                        .anyRequest()
                        .authenticated()
                )
                .formLogin(withDefaults());
        return http.build();
    }
```

```
/**
 * Configures the authentication manager builder with a DAO authentication provider.
 *
 * @param auth the {@link AuthenticationManagerBuilder} to modify
 * @throws Exception if an error occurs while configuring the authentication manager builder
 */
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.authenticationProvider(daoAuthenticationProvider());
}

/**
 * Creates and configures a {@link DaoAuthenticationProvider} bean.
 *
 * @return a configured {@link DaoAuthenticationProvider} instance
 */
@Bean
public DaoAuthenticationProvider daoAuthenticationProvider() {
    DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
    provider.setPasswordEncoder(passwordEncoder.passwordEncoder());
    provider.setUserDetailsService(userService);
    return provider;
}
```

src/main/java/org/example/uzgotuje/controller/AuthenticationController.java

```
@@ -1,137 +1,194 @@
package org.example.uzgotuje.controller;

import jakarta.servlet.http.Cookie;
import jakarta.servlet.http.HttpServletResponse;
import lombok.AllArgsConstructor;
import org.example.uzgotuje.database.entity.auth.User;
import org.example.uzgotuje.services.authorization.*;
import org.example.uzgotuje.services.token.TokenResponse;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Objects;

+ /**
+  * REST controller for handling authentication-related requests.
+  */
@RestController
@RequestMapping(path = "/auth")
@AllArgsConstructor
@CrossOrigin(origins = "*")
public class AuthenticationController {

    private final AuthenticationService authenticationService;
    private final RecaptchaService recaptchaService;
```

```
/**
 * Registers a new user.
 *
 * @param request the registration request containing user details
 * @return a response entity with the registration response and HTTP status
 */
@PostMapping(path = "/register")
public ResponseEntity<RegistrationResponse> register(@RequestBody RegistrationRequest request) {
    RegistrationResponse response = authenticationService.register(request);
    if ("Success".equals(response.getMessage()) || "Send new Token".equals(response.getMessage())) {
        return new ResponseEntity<>(response, HttpStatus.OK);
    } else {
        return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
    }
}
```

```
/**
 * Verifies the reCAPTCHA token.
 *
 * @param token the reCAPTCHA token to verify
 * @return a response entity with the verification result and HTTP status
 */
@PostMapping("/verifyCaptcha")
public ResponseEntity<String> verifyCaptcha(@RequestParam String token) {
    boolean isValid = recaptchaService.verifyRecaptcha(token);

    if (isValid) {
        return new ResponseEntity<>("Success", HttpStatus.OK);
    } else {
```

```
    } else {
        return new ResponseEntity<>("Failure", HttpStatus.BAD_REQUEST);
    }
}
```

```
/**
 * Confirms the email verification token.
 *
 * @param token the email verification token
 * @return a response entity with the token response and HTTP status
 */
@GetMapping(path = "/confirm")
public ResponseEntity<TokenResponse> confirm(@RequestParam("token") String token) {
    TokenResponse response = authenticationService.confirmToken(token);
    if ("Email confirmed".equals(response.getMessage())) {
        return new ResponseEntity<>(response, HttpStatus.OK);
    } else {
        return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
    }
}
```

```
// Logging User and creating cookie on user side and cookie in database
/**
 * Logs in the user and creates a session cookie.
 *
 * @param loginRequest the login request containing user credentials
 * @param response the HTTP servlet response to add the cookie to
 * @return a response entity with the login result and HTTP status
 */
```

```
@PostMapping(path = "/login")
public ResponseEntity<String> login(@RequestBody LoginRequest loginRequest, HttpServletResponse response) {
        String cookieValue = authenticationService.login(loginRequest.getEmail(), loginRequest.getPassword());

        if(Objects.equals(cookieValue, "Invalid credentials")) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid credentials");
        }
        // Set the cookie in the response
        Cookie cookie = new Cookie("SESSION_ID", cookieValue);
        cookie.setHttpOnly(true); // Prevent client-side access to the cookie
        cookie.setPath("/");
        cookie.setMaxAge(2 * 60 * 60); // 2 hours

    System.out.println("Cookie created: " + cookie.getName() + " = " + cookie.getValue());
        response.addCookie(cookie);

        return ResponseEntity.ok("Login successful");
}
```

```
// Checking if cookie is valid
/**
 * Checks if the session cookie is valid.
 *
 * @param cookieValue the session cookie value
 * @return a response entity with the validation result and HTTP status
 */
@GetMapping("/check")
public ResponseEntity<String> checkCookie(@CookieValue(value = "SESSION_ID", required = false) String cookieValue) {
    if (cookieValue != null && authenticationService.validateCookie(cookieValue)) {
        return ResponseEntity.ok("Cookie is valid");
```

```
        if (cookieValue != null && authenticationService.validateCookie(cookieValue)) {
            return ResponseEntity.ok("Cookie is valid");
        } else {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid or expired cookie");
        }
    }

    // Logging out user and deleting cookie from database
    /**
     * Logs out the user and deletes the session cookie.
     *
     * @param cookieValue the session cookie value
     * @param response the HTTP servlet response to remove the cookie from
     * @return a response entity with the logout result and HTTP status
     */
    @PostMapping("/logout")
    public ResponseEntity<String> logout(@CookieValue(value = "SESSION_ID", required = false) String cookieValue, HttpServletResponse response) {
        if (cookieValue != null) {
            authenticationService.logout(cookieValue);

            // Remove the cookie from the client
            Cookie cookie = new Cookie("SESSION_ID", null);
            cookie.setPath("/");
            cookie.setMaxAge(0);
            response.addCookie(cookie);

            return ResponseEntity.ok("Logged out successfully");
        } else {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("No active session");
        }
```

```
/**
 * Sends a password reset email.
 *
 * @param email the email request containing the user's email address
 * @return a response entity with the result and HTTP status
 */
@PostMapping("/resetPasswordEmail")
public ResponseEntity<String> resetPassword(@RequestBody ResetPasswordEmailRequest email) {
    String response = authenticationService.resetPasswordEmail(email.getEmail());
    if ("Success".equals(response)) {
        return ResponseEntity.ok("Password reset email sent");
    } else {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Email not found");
    }
}
```

```
/**
 * Resets the user's password.
 *
 * @param token the password reset token
 * @param passwordRequest the password reset request containing the new password
 * @return a response entity with the result and HTTP status
 */
@PostMapping("/reset")
public ResponseEntity<String> resetPassword(@RequestParam("token") String token, @RequestBody ResetPasswordRequest passwordRequest) {

    String response = authenticationService.resetPassword(token, passwordRequest.getPassword(), passwordRequest.getRepeatPassword());
    if("Passwords do not match".equals(response)){
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Passwords do not match");
    `
```

```
        }
        if ("Success".equals(response)) {
            return ResponseEntity.ok("Password reset successfully");
        } else {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Token not found");
        }
    }

    /**
     * Retrieves the authenticated user's details.
     *
     * @param cookieValue the session cookie value
     * @return a response entity with the user details and HTTP status
     */
    @GetMapping("/user")
    public ResponseEntity<User> getUsername(@CookieValue(value = "SESSION_ID", required = false) String cookieValue) {
        User user = authenticationService.validateCookieAndGetUser(cookieValue);
        if (cookieValue != null && user != null) {
            return ResponseEntity.ok(user);
        } else {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(null);
        }
    }

}
```

src/main/java/org/example/uzgotuje/controller/FileController.java

```
@@ -1,42 +1,51 @@
package org.example.uzgotuje.controller;

import org.springframework.core.io.InputStreamResource;
import org.springframework.core.io.Resource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

import jcifs.smb.SmbFile;
import jcifs.smb.SmbFileInputStream;

import java.io.IOException;
import java.io.InputStream;

+ /**
+  * REST controller for handling file retrieval requests.
+  */
@RestController
public class FileController {
```

```
+ /**
+  * REST controller for handling file retrieval requests.
+  */
  @RestController
  public class FileController {

+     /**
+      * Retrieves a file from the Samba server.
+      *
+      * @param filename the name of the file to retrieve
+      * @return a response entity containing the file resource or an error status
+      */
      @GetMapping("/files/{filename}")
      public ResponseEntity<Resource> getFile(@PathVariable String filename) {
          try {
              String sambaBaseUrl = "smb://192.168.0.188/shared/";
              String filePath = sambaBaseUrl + filename;
              SmbFile smbFile = new SmbFile(filePath);

              if (smbFile.exists() && smbFile.isFile()) {
                  InputStream inputStream = new SmbFileInputStream(smbFile);
                  Resource resource = new InputStreamResource(inputStream);

                  return ResponseEntity.ok()
                          .header(HttpHeaders.CONTENT_DISPOSITION, "inline; filename=\"" + filename + "\"")
                          .body(resource);
              } else {
                  return ResponseEntity.notFound().build();
              }
```

```
                        .body(resource);
                } else {
                    return ResponseEntity.notFound().build();
                }
            } catch (IOException e) {
                System.out.println("Error: " + e.getMessage());
                return ResponseEntity.internalServerError().build();
            }
        }
-   }
+   }
```

src/main/java/org/example/uzgotuje/controller/RecipeController.java

```
    //path to create recipe
    /**
     * Creates a new recipe.
     *
     * @param name the name of the recipe
     * @param description the description of the recipe
     * @param type the type of the recipe
     * @param jsonTags the JSON string of tags
     * @param jsonIngredients the JSON string of ingredients
     * @param images the images of the recipe
     * @param cookieValue the session cookie value
     * @return a response entity with the result and HTTP status
     */
    @PostMapping(path = "/create/recipe", consumes = {MediaType.MULTIPART_FORM_DATA_VALUE})
    public ResponseEntity<String> createRecipe(
            @RequestParam("name") String name,
            @RequestParam("description") String description,
            @RequestParam("type") String type,
            @RequestParam("tags") String jsonTags,
            @RequestParam("ingredients") String jsonIngredients,
            @RequestParam("images") MultipartFile[] images,
            @CookieValue(value = "SESSION_ID", required = false) String cookieValue) {
```

```
    //parse tags and ingredients
    List<Tag> tags = parseTags(jsonTags);
    List<RecipeIngredient> ingredients = parseIngredients(jsonIngredients);

    CreateRecipeRequest request = new CreateRecipeRequest(name, description, type, images, tags.toArray(new Tag[
    String response = recipeService.createRecipe(request, cookieValue);

    if ("Success".equals(response)) {
        return ResponseEntity.ok("Recipe created");
    } else if ("Unauthorized".equals(response)) {
        return ResponseEntity.status(401).body(response);
    }
    else {
        return ResponseEntity.badRequest().body(response);
    }
}

/**
 * Parses the JSON string of tags.
 *
 * @param json the JSON string of tags
 * @return a list of parsed tags
 */
private List<Tag> parseTags(String json) {
    try {
        ObjectMapper objectMapper = new ObjectMapper();
        return objectMapper.readValue(json, new TypeReference<>(){});
```

```
        //parse tags and ingredients
        List<Tag> tags = parseTags(jsonTags);
        List<RecipeIngredient> ingredients = parseIngredients(jsonIngredients);

        CreateRecipeRequest request = new CreateRecipeRequest(name, description, type, images, tags.toArray(new Tag[0]), ingredients.toArray(new RecipeIngredient[0]));
        String response = recipeService.createRecipe(request, cookieValue);

        if ("Success".equals(response)) {
            return ResponseEntity.ok("Recipe created");
        } else if ("Unauthorized".equals(response)) {
            return ResponseEntity.status(401).body(response);
        }
        else {
            return ResponseEntity.badRequest().body(response);
        }
    }

    /**
     * Parses the JSON string of tags.
     *
     * @param json the JSON string of tags
     * @return a list of parsed tags
     */
    private List<Tag> parseTags(String json) {
        try {
            ObjectMapper objectMapper = new ObjectMapper();
            return objectMapper.readValue(json, new TypeReference<>(){});
        } catch (Exception e) {
            throw new RuntimeException("Error parsing tags JSON", e);
            throw new RuntimeException("Error parsing tags JSON", e);
        }
    }

    /**
     * Parses the JSON string of ingredients.
     *
     * @param json the JSON string of ingredients
     * @return a list of parsed ingredients
     */
    private List<RecipeIngredient> parseIngredients(String json) {
        try {
            ObjectMapper objectMapper = new ObjectMapper();
            return objectMapper.readValue(json, new TypeReference<>(){});
        } catch (Exception e) {
            throw new RuntimeException("Error parsing ingredients JSON", e);
        }
    }

    //path to create / update rating
    /**
     * Creates or updates a rating for a recipe.
     *
     * @param request the request containing rating details
     * @param cookieValue the session cookie value
     * @return a response entity with the result and HTTP status
     */
    @PostMapping(path = "/create/rating")
    public ResponseEntity<String> createRating(@RequestBody CreateRatingRequest request , @CookieValue(value = "SESSION_ID", required = false) String cookieValue) {
```

```
    @PostMapping(path = "/create/rating")
    public ResponseEntity<String> createRating(@RequestBody CreateRatingRequest request , @CookieValue(value = "SESSION_ID", required = false) String cookieValue) {
        String response = recipeService.addRating(request.getRecipeId(),request.getScore(), cookieValue);

        if ("Success".equals(response)) {
            return ResponseEntity.ok("Rating created");
        } else if ("Unauthorized".equals(response)) {
            return ResponseEntity.status(401).body(response);
        }
        else {
            return ResponseEntity.badRequest().body(response);
        }
    }

    /**
     * Retrieves the user's rating for a recipe.
     *
     * @param recipeId the ID of the recipe
     * @param cookieValue the session cookie value
     * @return a response entity with the rating or an error status
     */
    @GetMapping(path = "/get/userRating")
    public ResponseEntity<Integer> getUserRating(@RequestParam("recipeId") Long recipeId, @CookieValue(value = "SESSION_ID", required = false) String cookieValue) {
        Integer response = recipeService.getUserRating(recipeId, cookieValue);
@ -127,53 +176,83 @@
        }
    }

    }

    /**
     * Creates a new favorite recipe.
     *
     * @param request the request containing favorite details
     * @param cookieValue the session cookie value
     * @return a response entity with the result and HTTP status
     */
    @PostMapping(path = "/create/favorite")
    public ResponseEntity<String> createFavorite(@RequestBody CreateFavoriteRequest request, @CookieValue(value = "SESSION_ID", required = false) String cookieValue) {
        String response = recipeService.updateFavorite(request.getRecipeId(), cookieValue);

        if ("Success".equals(response)) {
            return ResponseEntity.ok("Favorite created");
        } else if ("Unauthorized".equals(response)) {
            return ResponseEntity.status(401).body(response);
        }
        else {
            return ResponseEntity.badRequest().body(response);
        }
    }

    /**
     * Retrieves the user's favorite recipes.
     *
     * @param cookieValue the session cookie value
     * @return a response entity with the list of favorite recipes or an error status
     */
```

```
     */
    @GetMapping(path = "/get/favorites")
    public ResponseEntity<List<Recipe>> getFavoriteRecipes(@CookieValue(value = "SESSION_ID", required = false) String cookieValue) {
        List<Recipe> response = recipeService.getFavoriteRecipes(cookieValue);

        if (response != null) {
            return ResponseEntity.ok(response);
        } else {
            return ResponseEntity.badRequest().body(null);
        }
    }

    /**
     * Retrieves a recipe by its ID.
     *
     * @param id the ID of the recipe
     * @return a response entity with the recipe or an error status
     */
    @GetMapping(path = "/get/recipe")
    public ResponseEntity<Recipe> getRecipe(@RequestParam("id") Long id) {
        Recipe response = recipeService.getRecipe(id);

        if (response != null) {
            return ResponseEntity.ok(response);
        } else {
            return ResponseEntity.status(401).body(null);
        }
    }
```

```
    /**
     * Retrieves all recipes.
     *
     * @return a response entity with the list of recipes or an error status
     */
    @GetMapping(path = "/get/recipes")
    public ResponseEntity<List<Recipe>> getRecipes() {
        List<Recipe> response = recipeService.getRecipes();

        if (response != null) {
            return ResponseEntity.ok(response);
        } else {
            return ResponseEntity.status(401).body(null);
        }
    }

    /**
     * Searches for recipes based on the given criteria.
     *
     * @param request the search request containing search criteria
     * @return a response entity with the list of recipes or an error status
     */
    @PostMapping(path = "/get/recipeSearch")
    public ResponseEntity<List<Recipe>> getRecipesBySearch(@RequestBody RecipeSearchRequest request) {
        RecipeTypes type = null;
@@ -206,11 +285,21 @@
        }
    }
```

```
        }
    }

    /**
     * Retrieves all recipe types.
     *
     * @return a response entity with the list of recipe types
     */
    @GetMapping(path = "/get/recipeTypes")
    public ResponseEntity<RecipeTypes[]> getRecipeTypes() {
        return ResponseEntity.ok(RecipeTypes.values());
    }

    /**
     * Retrieves all tags.
     *
     * @return a response entity with the list of tags or an error status
     */
    @GetMapping(path = "/get/tags")
    public ResponseEntity<List<Tag>> getTags() {
        List<Tag> response = recipeService.getTags();
@@ -222,11 +311,21 @@
        }
    }

    /**
     * Retrieves all tag types.
     *
     * @return a response entity with the list of tag types
```

```
   */
  @GetMapping(path = "/get/tagTypes")
  public ResponseEntity<TagTypes[]> getTagTypes() {
      return ResponseEntity.ok(TagTypes.values());
  }

  /**
   * Retrieves all ingredients.
   *
   * @return a response entity with the list of ingredients or an error status
   */
  @GetMapping(path = "/get/ingredients")
  public ResponseEntity<List<Ingredient>> getIngredients() {
      List<Ingredient> response = recipeService.getIngredients();
-238,6 +337,12 @@
      }
  }

  /**
   * Retrieves all comments for a recipe.
   *
   * @param recipeId the ID of the recipe
   * @return a response entity with the list of comments or an error status
   */
  @GetMapping(path = "/get/commentsOfRecipe")
  public ResponseEntity<List<Comment>> getCommentsOfRecipe(@RequestParam("recipeId") Long recipeId) {
      List<Comment> response = recipeService.getCommentsOfRecipe(recipeId);
-249,6 +354,13 @@
      }
  }
```

```
    /**
     * Creates a new comment for a recipe.
     *
     * @param request the request containing comment details
     * @param cookieValue the session cookie value
     * @return a response entity with the result and HTTP status
     */
    @PostMapping(path = "/create/comment")
    public ResponseEntity<String> createComment(@RequestBody CreateCommentRequest request, @CookieValue(value = "SESSION_ID", required = false) String cookieValue) {
        String response = recipeService.createComment(request.getRecipeId(),request.getContent(), cookieValue);
@@ -263,6 +375,12 @@
        }
    }

    /**
     * Retrieves random recipes by type.
     *
     * @param type the type of the recipes
     * @return a response entity with the list of random recipes or an error status
     */
    @GetMapping(path = "/get/randomRecipes")
    public ResponseEntity<List<Recipe>> getRandomRecipes(@RequestParam("type") String type) {
        List<Recipe> response = recipeService.getRandomRecipesByType(type);
```

```
    @@ -263,14 +375,20 @@
            }
        }

+       /**
+        * Retrieves random recipes by type.
+        *
+        * @param type the type of the recipes
+        * @return a response entity with the list of random recipes or an error status
+        */
        @GetMapping(path = "/get/randomRecipes")
        public ResponseEntity<List<Recipe>> getRandomRecipes(@RequestParam("type") String type) {
            List<Recipe> response = recipeService.getRandomRecipesByType(type);

            if (response != null) {
                return ResponseEntity.ok(response);
            } else {
                return ResponseEntity.status(401).body(null);
            }
        }
    }
```

src/main/java/org/example/uzgotuje/database/entity/auth/ConfirmationToken.java

```
@@ -1,36 +1,52 @@
package org.example.uzgotuje.database.entity.auth;


import jakarta.persistence.*;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import java.time.LocalDateTime;

+ /**
+  * Entity representing a confirmation token used for user authentication.
+  */
@Getter
@Setter
@NoArgsConstructor
@Entity
public class ConfirmationToken {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
+
    @Column(nullable = false)
    private String token;
+
    @Column(nullable = false)
    private LocalDateTime createdAt;
```

```
    private String token;

    @Column(nullable = false)
    private LocalDateTime createdAt;

    @Column(nullable = false)
    private LocalDateTime expiresAt;

    private LocalDateTime confirmedAt = null;

    @ManyToOne(cascade = CascadeType.PERSIST)
    @JoinColumn(nullable = false, name = "user_id")
    private User user;

    public ConfirmationToken(String token, LocalDateTime createdAt, LocalDateTime expiresAt,User user) {
    /**
     * Constructs a new ConfirmationToken with the specified token, creation time, expiration time, and user.
     *
     * @param token the token string
     * @param createdAt the time the token was created
     * @param expiresAt the time the token expires
     * @param user the user associated with the token
     */
    public ConfirmationToken(String token, LocalDateTime createdAt, LocalDateTime expiresAt, User user) {
        this.token = token;
        this.createdAt = createdAt;
        this.expiresAt = expiresAt;
        this.user = user;
    }
`
```

```
  package org.example.uzgotuje.database.entity.auth;


  import jakarta.persistence.*;
  import lombok.Getter;
  import lombok.NoArgsConstructor;
  import lombok.Setter;

  import java.time.LocalDateTime;

+ /**
+  * Entity representing a confirmation token used for user authentication.
+  */
  @Getter
  @Setter
  @NoArgsConstructor
  @Entity
  public class ConfirmationToken {
      @Id
      @GeneratedValue(strategy = GenerationType.IDENTITY)
      private Long id;
+
      @Column(nullable = false)
      private String token;
+
      @Column(nullable = false)
      private LocalDateTime createdAt;
+
```

```
    private String token;

    @Column(nullable = false)
    private LocalDateTime createdAt;

    @Column(nullable = false)
    private LocalDateTime expiresAt;

    private LocalDateTime confirmedAt = null;

    @ManyToOne(cascade = CascadeType.PERSIST)
    @JoinColumn(nullable = false, name = "user_id")
    private User user;

    public ConfirmationToken(String token, LocalDateTime createdAt, LocalDateTime expiresAt,User user) {
    /**
     * Constructs a new ConfirmationToken with the specified token, creation time, expiration time, and user.
     *
     * @param token the token string
     * @param createdAt the time the token was created
     * @param expiresAt the time the token expires
     * @param user the user associated with the token
     */
    public ConfirmationToken(String token, LocalDateTime createdAt, LocalDateTime expiresAt, User user) {
        this.token = token;
        this.createdAt = createdAt;
        this.expiresAt = expiresAt;
        this.user = user;
    }
```

src/main/java/org/example/uzgotuje/database/entity/auth/SessionCookie.java

```
@@ -12,6 +12,9 @@

import java.util.*;

+ /**
+  * Entity representing a user in the authentication system.
+  */
@Entity
@Getter
@Setter
@@ -30,44 +33,87 @@ public class User implements UserDetails {
    private Boolean locked = false;
    private Boolean enabled = false;

+     /**
+      * Constructs a new User with the specified username, email, password, and role.
+      *
+      * @param username the username of the user
+      * @param email the email of the user
+      * @param password the password of the user
+      * @param appUserRole the role of the user
+      */
    public User(String username, String email, String password, UserRoles appUserRole) {
        this.username = username;
        this.email = email;
        this.password = password;
        this.appUserRole = appUserRole;
```

```
    /**
     * Returns the authorities granted to the user.
     *
     * @return a collection of granted authorities
     */
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        SimpleGrantedAuthority authority = new SimpleGrantedAuthority(appUserRole.name());
        return Collections.singletonList(authority);
    }

    /**
     * Returns the password used to authenticate the user.
     *
     * @return the password
     */
    @Override
    public String getPassword() {
        return password;
    }

    /**
     * Returns the username used to authenticate the user.
     *
     * @return the username
     */
    @Override
```

```
@Override
public String getUsername() {
    return username;
}

/**
 * Indicates whether the user's account has expired.
 *
 * @return true if the account is non-expired, false otherwise
 */
@Override
public boolean isAccountNonExpired() {
    return true;
}

/**
 * Indicates whether the user is locked or unlocked.
 *
 * @return true if the account is non-locked, false otherwise
 */
@Override
public boolean isAccountNonLocked() {
    return !locked;
}

/**
 * Indicates whether the user's credentials (password) has expired.
 *
 * @return true if the credentials are non-expired, false otherwise
```

```java
/**
 * Indicates whether the user's credentials (password) has expired.
 *
 * @return true if the credentials are non-expired, false otherwise
 */
@Override
public boolean isCredentialsNonExpired() {
    return true;
}

/**
 * Indicates whether the user is enabled or disabled.
 *
 * @return true if the user is enabled, false otherwise
 */
@Override
public boolean isEnabled() {
    return enabled;
```

```
@@ -1,22 +1,25 @@
package org.example.uzgotuje.database.entity.auth;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

import java.time.LocalDateTime;

+ /**
+  * Entity representing a session cookie used for user authentication.
+  */
@Entity
@Getter
@Setter
public class SessionCookie {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    private User user;

    private String cookieValue;
    private LocalDateTime expiryDate;
}
```

src/main/java/org/example/uzgotuje/database/entity/auth/UserRoles.java

```
@@ -1,6 +1,16 @@
package org.example.uzgotuje.database.entity.auth;

+ /**
+  * Enum representing the roles of a user in the authentication system.
+  */
public enum UserRoles {
+     /**
+      * Role for regular users.
+      */
        USER,
+
+     /**
+      * Role for administrators.
+      */
        ADMIN
}
```

src/main/java/org/example/uzgotuje/database/entity/recipe/Comment.java

```
@@ -7,6 +7,9 @@
import lombok.NoArgsConstructor;
import lombok.Setter;

+ /**
+  * Entity representing a comment on a recipe.
+  */
@Entity
@Getter
@Setter
@@ -25,6 +28,12 @@ public class Comment {
    @JoinColumn(name = "recipe_id", nullable = false)
    private Recipe recipe;

+    /**
+     * Constructs a new Comment with the specified content and username.
+     *
+     * @param content the content of the comment
+     * @param username the username of the commenter
+     */
    public Comment(String content, String username) {
        this.content = content;
        this.username = username;
```

src/main/java/org/example/uzgotuje/database/entity/recipe/Favorite.java

```
 import lombok.Setter;
 import org.example.uzgotuje.database.entity.auth.User;

+/**
+ * Entity representing a favorite recipe for a user.
+ */
 @Entity
 @Getter
 @Setter
@@ -26,8 +29,14 @@ public class Favorite {
     @JoinColumn(name = "recipe_id", nullable = false)
     private Recipe recipe; // Many-to-One relationship with Recipe

+    /**
+     * Constructs a new Favorite with the specified user and recipe.
+     *
+     * @param user the user who favorited the recipe
+     * @param recipe the recipe that is favorited
+     */
     public Favorite(User user, Recipe recipe) {
         this.user = user;
         this.recipe = recipe;
     }
-}
+}
```

src/main/java/org/example/uzgotuje/database/entity/recipe/Image.java

```
@@ -8,6 +8,9 @@
import lombok.NoArgsConstructor;
import lombok.Setter;

+/**
+ * Entity representing an image associated with a recipe.
+ */
+@Entity
+@Getter
+@Setter
@@ -25,6 +28,11 @@ public class Image {
    @JoinColumn(name = "recipe_id", nullable = false)
    private Recipe recipe;

+    /**
+     * Constructs a new Image with the specified image path.
+     *
+     * @param imagePath the path where the image is stored on the server
+     */
    public Image(String imagePath) {
        this.imagePath = imagePath;
    }
```

src/main/java/org/example/uzgotuje/database/entity/recipe/Ingredient.java

```
@@ -12,6 +12,9 @@

import java.util.Set;

+ /**
+  * Entity representing an ingredient in a recipe.
+  */
@Entity
@Getter
@Setter
@@ -28,6 +31,11 @@ public class Ingredient {
    @JsonIgnore
    private Set<RecipeIngredient> recipes;

+    /**
+     * Constructs a new Ingredient with the specified name.
+     *
+     * @param name the name of the ingredient
+     */
    public Ingredient(String name) {
        this.name = name;
    }
```

src/main/java/org/example/uzgotuje/database/entity/recipe/Rating.java

```
import lombok.Setter;
import org.example.uzgotuje.database.entity.auth.User;

+ /**
+  * Entity representing a rating given by a user to a recipe.
+  */
@Entity
@Getter
@Setter
@@ -29,6 +32,13 @@ public class Rating {
    @Column(nullable = false)
    private int score;

+     /**
+      * Constructs a new Rating with the specified user, recipe, and score.
+      *
+      * @param user the user who gave the rating
+      * @param recipe the recipe that is being rated
+      * @param score the score given to the recipe
+      */
    public Rating(User user, Recipe recipe, int score) {
        this.user = user;
        this.recipe = recipe;
```

src/main/java/org/example/uzgotuje/database/entity/recipe/Recipe.java

```
@@ -12,11 +12,14 @@
import java.util.HashSet;
import java.util.Set;

+ /**
+  * Entity representing a recipe.
+  */
@Entity
@Getter
@Setter
@NoArgsConstructor
- @EqualsAndHashCode(exclude = {"images", "ingredients", "tags","ratings"})
+ @EqualsAndHashCode(exclude = {"images", "ingredients", "tags", "ratings"})
public class Recipe {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
@@ -39,30 +42,40 @@ public class Recipe {

    @OneToMany(mappedBy = "recipe", cascade = CascadeType.ALL, orphanRemoval = true)
    @JsonManagedReference(value = "recipe-ingredient")
-     private Set<RecipeIngredient> ingredients = new HashSet<>();
+     private Set<RecipeIngredient> ingredients = new HashSet<>(); // One-to-Many relationship with RecipeIngredient

    @OneToMany(mappedBy = "recipe", cascade = CascadeType.ALL, orphanRemoval = true)
    @JsonIgnore
-     private Set<Rating> ratings = new HashSet<>();
+     private Set<Rating> ratings = new HashSet<>(); // One-to-Many relationship with Rating
```

```
@OneToMany(mappedBy = "recipe", cascade = CascadeType.ALL, orphanRemoval = true)
@JsonManagedReference(value = "recipe-comment")
private Set<Comment> comments = new HashSet<>();
private Set<Comment> comments = new HashSet<>(); // One-to-Many relationship with Comment

@ManyToMany
@JoinTable(
        name = "recipe_tag",
        joinColumns = @JoinColumn(name = "recipe_id"),
        inverseJoinColumns = @JoinColumn(name = "tag_id")
)
private Set<Tag> tags;

private Set<Tag> tags; // Many-to-Many relationship with Tag

/**
 * Constructs a new Recipe with the specified name, description, and type.
 *
 * @param name the name of the recipe
 * @param description the description of the recipe
 * @param type the type of the recipe
 */
public Recipe(String name, String description, String type) {
    this.name = name;
    this.description = description;
    this.type = RecipeTypes.valueOf(type);
}
```

```
    /**
     * Updates the average rating of the recipe based on its ratings.
     */
    public void updateAverageRating() {
        if (ratings.isEmpty()) {
            this.averageRating = 0.0;
@@ -73,5 +86,4 @@ public void updateAverageRating() {
                    .orElse(0.0);
        }
    }


}
}
```

src/main/java/org/example/uzgotuje/database/entity/recipe/RecipeIngredient.java

```
import jakarta.persistence.*;
import lombok.*;

+ /**
+  * Entity representing the relationship between a recipe and an ingr
+  * including the quantity and type of the ingredient used in the rec
+  */
@Entity
@Getter
@Setter
@@ -20,20 +24,27 @@ public class RecipeIngredient {
    @ManyToOne
    @JsonBackReference(value = "recipe-ingredient")
    @JoinColumn(name = "recipe_id")
-     private Recipe recipe;
+     private Recipe recipe; // Many-to-One relationship with Recipe

    @ManyToOne
    @JoinColumn(name = "ingredient_id")
-     private Ingredient ingredient;
-
-     private String quantity;     // Additional column
-     private String quantityType; // Additional column
```

```
    @ManyToOne
    @JoinColumn(name = "ingredient_id")
    private Ingredient ingredient;

    private String quantity;     // Additional column
    private String quantityType; // Additional column
```

```
    private Ingredient ingredient; // Many-to-One relationship with Ingredient

    private String quantity;     // Quantity of the ingredient used in the recipe
    private String quantityType; // Type of the quantity (e.g., grams, cups)

    /**
     * Constructs a new RecipeIngredient with the specified recipe, ingredient, quantity, and quantity type.
     *
     * @param recipe the recipe that uses the ingredient
     * @param ingredient the ingredient used in the recipe
     * @param quantity the quantity of the ingredient used
     * @param quantityType the type of the quantity (e.g., grams, cups)
     */
    public RecipeIngredient(Recipe recipe, Ingredient ingredient, String quantity, String quantityType) {
        this.recipe = recipe;
        this.ingredient = ingredient;
        this.quantity = quantity;
        this.quantityType = quantityType;
    }
```

src/main/java/org/example/uzgotuje/database/entity/recipe/RecipeTypes.java

```
@@ -1,7 +1,10 @@
package org.example.uzgotuje.database.entity.recipe;

+ /**
+  * Enum representing the different types of recipes.
+  */
public enum RecipeTypes {
-     BREAKFAST,
-     DINNER,
-     SUPPER
+     BREAKFAST, // Represents a breakfast recipe
+     DINNER,    // Represents a dinner recipe
+     SUPPER     // Represents a supper recipe
}
```

src/main/java/org/example/uzgotuje/database/entity/recipe/Tag.java

```
@@ -10,6 +10,9 @@

import java.util.Set;

+ /**
+  * Entity representing a tag that can be associated with recipes.
+  */
@Entity
@Getter
@Setter
@@ -21,14 +24,20 @@ public class Tag {
    private Long id;

    @Enumerated(EnumType.STRING)
-   private TagTypes tagType;
+   private TagTypes tagType; // Type of the tag

-   private String name;
+   private String name; // Name of the tag

    @ManyToMany(mappedBy = "tags")
    @JsonBackReference
-   private Set<Recipe> recipes;
-
+   private Set<Recipe> recipes; // Recipes associated with this tag
+
+   /**
+    * Constructs a new Tag with the specified tag type and name.
+    *
+    * @param tagType the type of the tag
+    * @param name the name of the tag
+    */
    public Tag(String tagType, String name) {
        this.tagType = TagTypes.valueOf(tagType);
        this.name = name;
```

src/main/java/org/example/uzgotuje/database/entity/recipe/TagTypes.java

```
@@ -1,8 +1,11 @@
 package org.example.uzgotuje.database.entity.recipe;

+/**
+ * Enum representing the different types of tags that can be associated with recipes.
+ */
 public enum TagTypes {
-    DIET,
-    CUISINE,
-    FLAVOR,
-    DIFFICULTY
+    DIET,        // Represents a dietary tag
+    CUISINE,     // Represents a cuisine tag
+    FLAVOR,      // Represents a flavor tag
+    DIFFICULTY   // Represents a difficulty level tag
 }
```

src/main/java/org/example/uzgotuje/database/repository/auth/ConfirmationTokenRepository.java

```
@@ -1,8 +1,11 @@
 package org.example.uzgotuje.database.entity.recipe;

+ /**
+  * Enum representing the different types of tags that can be associated with recipes.
+  */
 public enum TagTypes {
-    DIET,
-    CUISINE,
-    FLAVOR,
-    DIFFICULTY
+    DIET,        // Represents a dietary tag
+    CUISINE,     // Represents a cuisine tag
+    FLAVOR,      // Represents a flavor tag
+    DIFFICULTY   // Represents a difficulty level tag
 }
```

src/main/java/org/example/uzgotuje/database/repository/auth/SessionCookieRepository.java

```
@@ -5,6 +5,16 @@

 import java.util.Optional;

+/**
+ * Repository interface for managing SessionCookie entities.
+ */
 public interface SessionCookieRepository extends JpaRepository<SessionCookie, Long> {
+
+    /**
+     * Finds a SessionCookie by its cookie value.
+     *
+     * @param cookieValue the value of the cookie
+     * @return an Optional containing the found SessionCookie, or empty if not found
+     */
     Optional<SessionCookie> findByCookieValue(String cookieValue);
 }
```

src/main/java/org/example/uzgotuje/database/repository/auth/UserRepository.java

```
@@ -7,8 +7,18 @@

 import java.util.Optional;

+/**
+ * Repository interface for managing User entities.
+ */
 @Repository
 @Transactional(readOnly = true)
-public interface UserRepository extends JpaRepository<User,Long> {
+public interface UserRepository extends JpaRepository<User, Long> {
+
+    /**
+     * Finds a User by their email address.
+     *
+     * @param email the email address of the User
+     * @return an Optional containing the found User, or empty if not found
+     */
```

src/main/java/org/example/uzgotuje/database/repository/recipe/CommentRepository.java

```
@@ -7,6 +7,16 @@
import java.util.List;
import java.util.Optional;

+ /**
+  * Repository interface for managing Comment entities.
+  */
public interface CommentRepository extends JpaRepository<Comment, Long> {
+
+     /**
+      * Finds a list of Comments associated with a given Recipe.
+      *
+      * @param recipe the Recipe whose comments are to be found
+      * @return an Optional containing the list of found Comments, or empty if none found
+      */
        Optional<List<Comment>> findByRecipe(Recipe recipe);
}
```

src/main/java/org/example/uzgotuje/database/repository/recipe/FavoriteRepository.java

```
@@ -8,8 +8,33 @@
import java.util.List;
import java.util.Optional;

+ /**
+  * Repository interface for managing Favorite entities.
+  */
public interface FavoriteRepository extends JpaRepository<Favorite, Long> {
+
+     /**
+      * Finds a Favorite by the given User and Recipe.
+      *
+      * @param user the User associated with the Favorite
+      * @param recipe the Recipe associated with the Favorite
+      * @return an Optional containing the found Favorite, or empty if not found
+      */
        Optional<Favorite> findByUserAndRecipe(User user, Recipe recipe);
+
```

```
24  +
25  +     /**
26  +      * Finds all Favorites associated with a given User.
27  +      *
28  +      * @param user the User whose Favorites are to be found
29  +      * @return a list of Favorites associated with the given User
30  +      */
13  31        List<Favorite> findAllByUser(User user);
32  +
33  +     /**
34  +      * Deletes a Favorite by the given User and Recipe.
35  +      *
36  +      * @param user the User associated with the Favorite
37  +      * @param recipe the Recipe associated with the Favorite
38  +      */
14  39        void deleteByUserAndRecipe(User user, Recipe recipe);
15  40    }
```

src/main/java/org/example/uzgotuje/database/repository/recipe/ImageRepository.java

```
    @@ -5,6 +5,16 @@
5   5
6   6     import java.util.List;
7   7
8   + /**
9   +  * Repository interface for managing Image entities.
10  +  */
8   11    public interface ImageRepository extends JpaRepository<Image, Long> {
12  +
13  +     /**
14  +      * Finds a list of Images associated with a given Recipe ID.
15  +      *
16  +      * @param recipeId the ID of the Recipe whose images are to be found
17  +      * @return a list of Images associated with the given Recipe ID
18  +      */
9   19        List<Image> findByRecipeId(Long recipeId);
10  20    }
```

src/main/java/org/example/uzgotuje/database/repository/recipe/IngredientRepository.java

```
@@ -5,7 +5,16 @@

5  5
6  6   import java.util.Optional;
7  7
    8  + /**
    9  +  * Repository interface for managing Ingredient entities.
    10 +  */
8  11  public interface IngredientRepository extends JpaRepository<Ingredient, Long> {
9      -     //find ingredient by name
    12 +
    13 +     /**
    14 +      * Finds an Ingredient by its name.
    15 +      *
    16 +      * @param name the name of the Ingredient
    17 +      * @return an Optional containing the found Ingredient, or empty if not found
    18 +      */
10 19          Optional<Ingredient> findByName(String name);
11 20  }
```

src/main/java/org/example/uzgotuje/database/repository/recipe/RatingRepository.java

```
@@ -8,9 +8,41 @@

8  8   import java.util.List;
9  9   import java.util.Optional;
10 10
    11 + /**
    12 +  * Repository interface for managing Rating entities.
    13 +  */
11 14  public interface RatingRepository extends JpaRepository<Rating, Long> {
    15 +
    16 +     /**
    17 +      * Finds a Rating by the given User and Recipe.
    18 +      *
    19 +      * @param user the User associated with the Rating
    20 +      * @param recipe the Recipe associated with the Rating
    21 +      * @return an Optional containing the found Rating, or empty if not found
    22 +      */
```

```
15 +
16 +     /**
17 +      * Finds a Rating by the given User and Recipe.
18 +      *
19 +      * @param user the User associated with the Rating
20 +      * @param recipe the Recipe associated with the Rating
21 +      * @return an Optional containing the found Rating, or empty if not found
22 +      */
12 23       Optional<Rating> findByUserAndRecipe(User user, Recipe recipe);
24 +
25 +     /**
26 +      * Finds all Ratings associated with a given Recipe.
27 +      *
28 +      * @param recipe the Recipe whose Ratings are to be found
29 +      * @return a list of Ratings associated with the given Recipe
30 +      */
13 31       List<Rating> findAllByRecipe(Recipe recipe);
32 +
33 +     /**
34 +      * Finds all Ratings associated with a given User.
35 +      *
36 +      * @param user the User whose Ratings are to be found
37 +      * @return a list of Ratings associated with the given User
38 +      */
14 39       List<Rating> findAllByUser(User user);
40 +
41 +     /**
42 +      * Deletes a Rating by the given User and Recipe.
43 +      *
44 +      * @param user the User associated with the Rating
45 +      * @param recipe the Recipe associated with the Rating
46 +      */
15 47       void deleteByUserAndRecipe(User user, Recipe recipe);
16 48   }
```

src/main/java/org/example/uzgotuje/database/repository/recipe/RecipeIngredientRepository.java

```
@@ -5,9 +5,24 @@

 import java.util.List;

+/**
+ * Repository interface for managing RecipeIngredient entities.
+ */
 public interface RecipeIngredientRepository extends JpaRepository<RecipeIngredient, Long> {
-    //find all ingredients for recipe
+
+    /**
+     * Finds all ingredients for a given Recipe ID.
+     *
+     * @param recipeId the ID of the Recipe whose ingredients are to be found
+     * @return a list of RecipeIngredients associated with the given Recipe ID
+     */
     List<RecipeIngredient> findByRecipeId(Long recipeId);
-    //find all recipes with ingredient
+
+    /**
+     * Finds all recipes with a given Ingredient ID.
+     *
+     * @param ingredientId the ID of the Ingredient whose recipes are to be found
+     * @return a list of RecipeIngredients associated with the given Ingredient ID
+     */
     List<RecipeIngredient> findByIngredientId(Long ingredientId);
 }
```

src/main/java/org/example/uzgotuje/database/repository/recipe/RecipeRepository.java

```
@@ -9,11 +9,32 @@
import java.util.List;
import java.util.Optional;

+ /**
+  * Repository interface for managing Recipe entities.
+  */
public interface RecipeRepository extends JpaRepository<Recipe, Long>, JpaSpecificationExecutor<Recipe> {
+
+     /**
+      * Finds a Recipe by its name.
+      *
+      * @param name the name of the Recipe
+      * @return an Optional containing the found Recipe, or empty if not found
+      */
    Optional<Recipe> findByName(String name);

+     /**
+      * Deletes a Recipe by its ID.
+      *
+      * @param id the ID of the Recipe to be deleted
+      */
    void deleteById(Long id);

+     /**
+      * Finds a list of random Recipes by type.
+      *
+      * @param type the type of the Recipes to be found
+      * @return a list of random Recipes of the given type
+      */
    @Query(value = "SELECT * FROM recipe WHERE type = :type ORDER BY RAND() LIMIT 5", nativeQuery = true)
    List<Recipe> findRandomRecipesByType(@Param("type") String type);
}
...
```

src/main/java/org/example/uzgotuje/database/repository/recipe/RecipeSpecification.java

```
@@ -10,6 +10,12 @@

public class RecipeSpecification {

+    /**
+     * Creates a specification to filter recipes by a set of tags.
+     *
+     * @param tags the set of tags to filter recipes by
+     * @return a specification to filter recipes by the given tags
+     */
     public static Specification<Recipe> hasTags(Set<Tag> tags) {
         return (root, query, criteriaBuilder) -> {
             if (tags == null || tags.isEmpty()) {
@@ -26,6 +32,12 @@ public static Specification<Recipe> hasTags(Set<Tag> tags) {
         };
     }

+    /**
+     * Creates a specification to filter recipes by name or ingredient name.
+     *
+     * @param name the name to filter recipes by
+     * @return a specification to filter recipes by the given name or ingredient name
+     */
     public static Specification<Recipe> hasName(String name) {
         return (root, query, criteriaBuilder) -> {
             if (name == null || name.isEmpty()) {
@@ -47,6 +59,12 @@ public static Specification<Recipe> hasName(String name) {
         }

+    /**
+     * Creates a specification to filter recipes by type.
+     *
+     * @param type the type to filter recipes by
+     * @return a specification to filter recipes by the given type
+     */
     public static Specification<Recipe> hasType(RecipeTypes type) {
         return (root, query, criteriaBuilder) -> {
             if (type == null) {
```

src/main/java/org/example/uzgotuje/database/repository/recipe/TagRepository.java

```
@@ -5,8 +5,16 @@

 import java.util.Optional;

+/**
+ * Repository interface for managing Tag entities.
+ */
 public interface TagRepository extends JpaRepository<Tag, Long> {
-    //find tag by name
-    Optional<Tag> findByName(String name);

+    /**
+     * Finds a Tag by its name.
+     *
+     * @param name the name of the Tag
+     * @return an Optional containing the found Tag, or empty if not found
+     */
+    Optional<Tag> findByName(String name);
 }
```

src/main/java/org/example/uzgotuje/services/UserService.java

```
@@ -16,29 +16,61 @@
16 16  import java.util.Optional;
17 17  import java.util.UUID;
18 18
   19 + /**
   20 +  * Service class for managing users.
   21 +  */
19 22  @Service
20 23  @AllArgsConstructor
21 24  public class UserService implements UserDetailsService {
22 25
   26 +     /**
   27 +      * Repository for managing users.
   28 +      */
23 29      private final UserRepository userRepository;
   30 +
   31 +     /**
   32 +      * Configuration for password encoding.
   33 +      */
24 34      private final PasswordEncoderConfig passwordEncoderConfig;
   35 +
   36 +     /**
   37 +      * Service for managing confirmation tokens.
   38 +      */
25 39      private final ConfirmationTokenService confirmationTokenService;
26 40
   41 +     /**
   42 +      * Loads a user by their email.
   43 +      *
   44 +      * @param email the email of the user
   45 +      * @return the user details
   46 +      * @throws UsernameNotFoundException if the user is not found
   47 +      */
27 48      @Override
28 49      public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
29 50          return userRepository.findByEmail(email)
30 51                  .orElseThrow(() -> new UsernameNotFoundException("User with email " + email + " not found"));
31 52      }
```

```
+    /**
+     * Retrieves a user by their email.
+     *
+     * @param email the email of the user
+     * @return an optional containing the user if found, or empty if not found
+     */
     public Optional<User> getUserByEmail(String email){
         return userRepository.findByEmail(email);
     }

-    //register user
+    /**
+     * Registers a new user.
+     *
+     * @param user the user to register
+     * @return the registration response containing a message and token
+     */
     public RegistrationResponse signUpUser(User user){
         boolean userExists = userRepository.findByEmail(user.getEmail()).isPresent();

-        //if user exists, delete old token and send new one
+        // If user exists, delete old token and send new one
         if(userExists){
             confirmationTokenService.deleteConfirmationTokenByUser(user);
             String newToken = UUID.randomUUID().toString();
@@ -52,7 +84,7 @@ public RegistrationResponse signUpUser(User user){
             return new RegistrationResponse("Send new Token", newToken);
         }

-        //if user doesn't exist, encode password and save user
+        // If user doesn't exist, encode password and save user
         String encodedPassword = passwordEncoderConfig.passwordEncoder().encode(user.getPassword());

         user.setPassword(encodedPassword);
@@ -66,7 +98,11 @@ public RegistrationResponse signUpUser(User user){
         return new RegistrationResponse("Success",token);
     }

```

```
        87  +         // If user doesn't exist, encode password and save user
56      88            String encodedPassword = passwordEncoderConfig.passwordEncoder().encode(user.getPassword());
57      89
58      90            user.setPassword(encodedPassword);
            @@ -66,7 +98,11 @@ public RegistrationResponse signUpUser(User user){
66      98            return new RegistrationResponse("Success",token);
67      99        }
68      100
69          -
        101 +     /**
        102 +      * Enables a user by their email.
        103 +      *
        104 +      * @param email the email of the user
        105 +      */
70      106       public void enableUser(String email) {
71      107           User user = userRepository.findByEmail(email)
72      108                   .orElseThrow(() -> new UsernameNotFoundException("User with email " + email + " not found"));
            @@ -75,7 +111,12 @@ public void enableUser(String email) {
75      111           userRepository.save(user);
76      112       }
77      113
        114 +     /**
        115 +      * Updates a user.
        116 +      *
        117 +      * @param user the user to update
        118 +      */
78      119       public void updateUser(User user){
79      120           userRepository.save(user);
80      121       }
81          - }
        122 + }
```

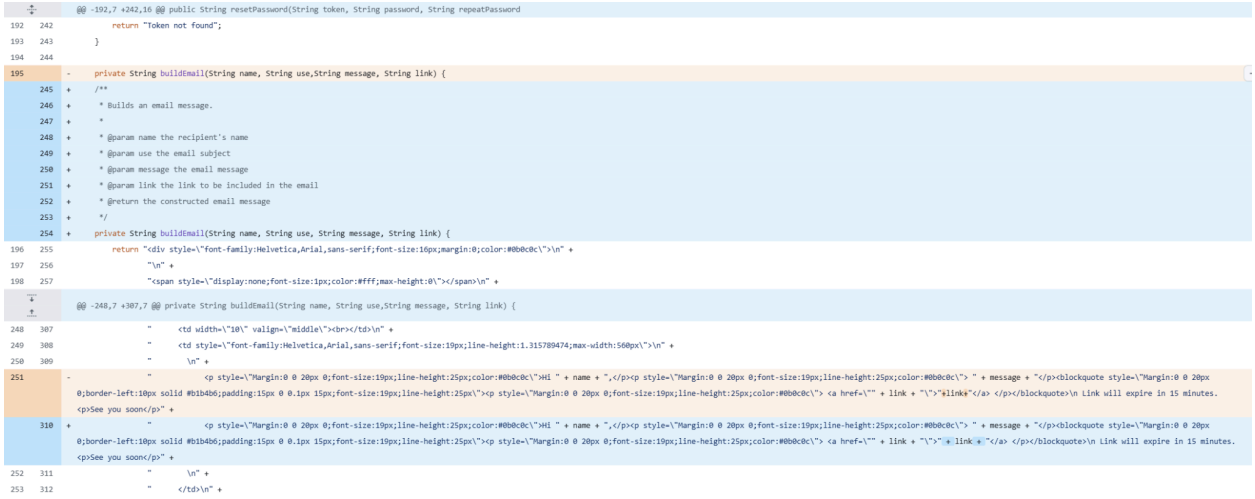
src/main/java/org/example/uzgotuje/services/authorization/AuthenticationService.java

```
@@ -32,15 +32,21 @@ public class AuthenticationService {
32 32       private final PasswordEncoderConfig passwordEncoderConfig;
33 33       private final EmailSender emailSender;
34 34
   35 +    /**
   36 +     * Registers a new user.
   37 +     *
   38 +     * @param request the registration request containing user details
   39 +     * @return a RegistrationResponse containing the result of the registration
   40 +     */
35 41       public RegistrationResponse register(RegistrationRequest request) {
36    -          boolean isEmailValid=emailValidator.test(request.getEmail());
37    -          if(!isEmailValid){
   42 +          boolean isEmailValid = emailValidator.test(request.getEmail());
   43 +          if (!isEmailValid) {
38 44               return new RegistrationResponse("Email is not valid", "");
39 45           }
40    -          if(!request.getPassword().equals(request.getRepeatPassword())){
   46 +          if (!request.getPassword().equals(request.getRepeatPassword())) {
41 47               return new RegistrationResponse("passwords do not match", "");
42 48           }
43    -          if(request.getUsername().isEmpty()){
   49 +          if (request.getUsername().isEmpty()) {
44 50               return new RegistrationResponse("username is empty", "");
45 51           }
46 52           RegistrationResponse response = userService.signUpUser(
@@ -55,27 +61,33 @@ public RegistrationResponse register(RegistrationRequest request) {
55 61           String link = "http://89.77.30.155/api/auth/confirm?token=" + response.getToken();
56 62           emailSender.send(
57 63                   request.getEmail(),
58    -                  buildEmail(request.getUsername()
59    -                  ,"Confirm your email",
60    -                  "Thank you for registering. Please click on the below link to activate your account:",
61    -                  link
   64 +                  buildEmail(request.getUsername(),
   65 +                          "Confirm your email",
   66 +                          "Thank you for registering. Please click on the below link to activate your account:"
```

```
65  +                   "Confirm your email",
66  +                   "Thank you for registering. Please click on the below link to activate your account:",
67  +                   link
62  68                  ));
63  69              return response;
64  70          }
65  71
72  +       /**
73  +        * Confirms a token and activates the associated user account.
74  +        *
75  +        * @param token the token to be confirmed
76  +        * @return a TokenResponse containing the result of the confirmation
77  +        */
66  78          @Transactional
67  79          public TokenResponse confirmToken(String token) {
68  80              Optional<ConfirmationToken> confirmationToken = confirmationTokenRepository.findByToken(token);
69  81              if (confirmationToken.isEmpty()) {
70  82                  return new TokenResponse("Token not found");
71  83              }
72      -           if(confirmationToken.get().getConfirmedAt() != null){
84  +           if (confirmationToken.get().getConfirmedAt() != null) {
73  85                  return new TokenResponse("Email already confirmed");
74  86              }
75  87
76  88              LocalDateTime expiredAt = confirmationToken.get().getExpiresAt();
77  89
78      -           if(expiredAt.isBefore(LocalDateTime.now())){
90  +           if (expiredAt.isBefore(LocalDateTime.now())) {
79  91                  return new TokenResponse("Token expired");
80  92              }
81  93
    @@ -86,6 +98,13 @@ public TokenResponse confirmToken(String token) {
86  98              return new TokenResponse("Email confirmed");
87  99          }
88  100
101 +       /**
102 +        * Logs in a user by validating their email and password.
103 +        *
104 +        * @param email the user's email
105 +        * @param password the user's password
106 +        * @return a cookie value if login is successful, otherwise "Invalid credentials"
```

```
     107 +      */
89   108        public String login(String email, String password) {
90   109            Optional<User> userOpt = userService.getUserByEmail(email);
91   110
     @@ -108,6 +127,12 @@ public String login(String email, String password) {
108  127            return "Invalid credentials";
109  128        }
110  129
     130 +      /**
     131 +       * Validates a session cookie.
     132 +       *
     133 +       * @param cookieValue the value of the cookie to be validated
     134 +       * @return true if the cookie is valid, otherwise false
     135 +       */
111  136        public boolean validateCookie(String cookieValue) {
112  137            Optional<SessionCookie> userCookieOpt = sessionCookieRepository.findByCookieValue(cookieValue);
113  138
     @@ -120,6 +145,12 @@ public boolean validateCookie(String cookieValue) {
120  145            return false;
121  146        }
122  147
     148 +      /**
     149 +       * Validates a session cookie and retrieves the associated user.
     150 +       *
     151 +       * @param cookieValue the value of the cookie to be validated
     152 +       * @return the associated User if the cookie is valid, otherwise null
     153 +       */
123  154        public User validateCookieAndGetUser(String cookieValue) {
124  155            Optional<SessionCookie> userCookieOpt = sessionCookieRepository.findByCookieValue(cookieValue);
125  156
     @@ -134,11 +165,22 @@ public User validateCookieAndGetUser(String cookieValue) {
134  165            return null;
135  166        }
136  167
     168 +      /**
     169 +       * Logs out a user by deleting their session cookie.
     170 +       *
     171 +       * @param cookieValue the value of the cookie to be deleted
     172 +       */
```

```
136  167
     168  +     /**
     169  +      * Logs out a user by deleting their session cookie.
     170  +      *
     171  +      * @param cookieValue the value of the cookie to be deleted
     172  +      */
137  173        public void logout(String cookieValue) {
138  174            sessionCookieRepository.findByCookieValue(cookieValue)
139  175                    .ifPresent(sessionCookieRepository::delete);
140  176        }
141  177
     178  +     /**
     179  +      * Sends a password reset email to the user.
     180  +      *
     181  +      * @param email the user's email
     182  +      * @return "Success" if the email was sent, otherwise "User not found"
     183  +      */
142  184        @Transactional
143  185        public String resetPasswordEmail(String email) {
144  186            Optional<User> userOpt = userService.getUserByEmail(email);
          @@ -166,6 +208,14 @@ public String resetPasswordEmail(String email) {
166  208            return "User not found";
167  209        }
168  210
     211  +     /**
     212  +      * Resets a user's password using a token.
     213  +      *
     214  +      * @param token the token to be used for password reset
     215  +      * @param password the new password
     216  +      * @param repeatPassword the repeated new password
     217  +      * @return "Success" if the password was reset, otherwise an error message
     218  +      */
169  219        public String resetPassword(String token, String password, String repeatPassword) {
170  220            if (!password.equals(repeatPassword)) {
171  221                return "Passwords do not match";
       @@ -192,7 +242,16 @@ public String resetPassword(String token, String password, String repeatPassword
192  242            return "Token not found";
193  243        }
```



```
@@ -192,7 +242,16 @@ public String resetPassword(String token, String password, String repeatPassword
        return "Token not found";
    }

-   private String buildEmail(String name, String use,String message, String link) {
+   /**
+    * Builds an email message.
+    *
+    * @param name the recipient's name
+    * @param use the email subject
+    * @param message the email message
+    * @param link the link to be included in the email
+    * @return the constructed email message
+    */
+   private String buildEmail(String name, String use, String message, String link) {
        return "<div style=\"font-family:Helvetica,Arial,sans-serif;font-size:16px;margin:0;color:#0b0c0c\">\n" +
                "\n" +
                "<span style=\"display:none;font-size:1px;color:#fff;max-height:0\"></span>\n" +
@@ -248,7 +307,7 @@ private String buildEmail(String name, String use,String message, String link) {
                "      <td width=\"10\" valign=\"middle\"><br></td>\n" +
                "      <td style=\"font-family:Helvetica,Arial,sans-serif;font-size:19px;line-height:1.315789474;max-width:560px\">\n" +
                "        \n" +
-               "          <p style=\"Margin:0 0 20px 0;font-size:19px;line-height:25px;color:#0b0c0c\">Hi " + name + ",</p><p style=\"Margin:0 0 20px 0;font-size:19px;line-height:25px;color:#0b0c0c\"> " + message + "</p><blockquote style=\"Margin:0 0 20px 0;border-left:10px solid #b1b4b6;padding:15px 0 0.1px 15px;font-size:19px;line-height:25px\"><p style=\"Margin:0 0 20px 0;font-size:19px;line-height:25px;color:#0b0c0c\"> <a href=\"" + link + "\">"+link+"</a> </p></blockquote>\n Link will expire in 15 minutes. <p>See you soon</p>" +
+               "          <p style=\"Margin:0 0 20px 0;font-size:19px;line-height:25px;color:#0b0c0c\">Hi " + name + ",</p><p style=\"Margin:0 0 20px 0;font-size:19px;line-height:25px;color:#0b0c0c\"> " + message + "</p><blockquote style=\"Margin:0 0 20px 0;border-left:10px solid #b1b4b6;padding:15px 0 0.1px 15px;font-size:19px;line-height:25px\"><p style=\"Margin:0 0 20px 0;font-size:19px;line-height:25px;color:#0b0c0c\"> <a href=\"" + link + "\">" + link + "</a> </p></blockquote>\n Link will expire in 15 minutes. <p>See you soon</p>" +
                "        \n" +
                "      </td>\n" +
```

src/main/java/org/example/uzgotuje/services/authorization/LoginRequest.java

```
@@ -10,6 +10,13 @@
@EqualsAndHashCode
@ToString
public class LoginRequest {
+    /**
+     * The email of the user attempting to log in.
+     */
    private final String email;
+
+    /**
+     * The password of the user attempting to log in.
+     */
    private final String password;
}
```

src/main/java/org/example/uzgotuje/services/authorization/RecaptchaService.java

```
@@ -15,10 +15,21 @@ public class RecaptchaService {

    private final RestTemplate restTemplate;

+    /**
+     * Constructs a new RecaptchaService with the given RestTemplate.
+     *
+     * @param restTemplate the RestTemplate to be used for making HTTP requests
+     */
    public RecaptchaService(RestTemplate restTemplate) {
        this.restTemplate = restTemplate;
    }

+    /**
+     * Verifies the reCAPTCHA response with Google's reCAPTCHA API.
+     *
+     * @param recaptchaResponse the reCAPTCHA response token provided by the client
+     * @return true if the reCAPTCHA response is valid, otherwise false
+     */
    public boolean verifyRecaptcha(String recaptchaResponse) {
        // Prepare request body
        String url = RECAPTCHA_VERIFY_URL + "?secret=" + recaptchaSecretKey + "&response=" + recaptchaResponse;
```

src/main/java/org/example/uzgotuje/services/authorization/RegistrationRequest.java

```
@@ -10,8 +10,23 @@
@EqualsAndHashCode
@ToString
public class RegistrationRequest {
+    /**
+     * The username of the user registering.
+     */
    private final String username;
+
+    /**
+     * The email of the user registering.
+     */
    private final String email;
+
+    /**
+     * The password of the user registering.
+     */
    private final String password;
+
+    /**
+     * The repeated password for confirmation.
+     */
    private final String repeatPassword;
}
```

src/main/java/org/example/uzgotuje/services/authorization/RegistrationResponse.java

```
@@ -6,6 +6,13 @@
@Getter
@AllArgsConstructor
public class RegistrationResponse {
+    /**
+     * The message indicating the result of the registration.
+     */
    private final String message;
+
+    /**
+     * The token generated upon successful registration.
+     */
    private final String token;
}
```

src/main/java/org/example/uzgotuje/services/authorization/ResetPasswordEmailRequest.java

```
@@ -11,10 +11,18 @@
@EqualsAndHashCode
@ToString
public class ResetPasswordEmailRequest {
+    /**
+     * The email of the user requesting a password reset.
+     */
    private final String email;
+
+    /**
+     * Constructs a new ResetPasswordEmailRequest with the given email.
+     *
+     * @param email the email of the user requesting a password reset
+     */
    @JsonCreator
    public ResetPasswordEmailRequest(@JsonProperty("email") String email) {
        this.email = email;
    }
-
}
```

src/main/java/org/example/uzgotuje/services/authorization/ResetPasswordRequest.java

```
@@ -10,6 +10,13 @@
@EqualsAndHashCode
@ToString
public class ResetPasswordRequest {
+    /**
+     * The new password for the user.
+     */
    private final String password;
+
+    /**
+     * The repeated password for confirmation.
+     */
    private final String repeatPassword;
- }
```

src/main/java/org/example/uzgotuje/services/email/EmailSender.java

```
@@ -1,5 +1,11 @@
package org.example.uzgotuje.services.email;

public interface EmailSender {
+    /**
+     * Sends an email to the specified recipient.
+     *
+     * @param to the recipient's email address
+     * @param email the content of the email to be sent
+     */
    void send(String to, String email);
}
```

src/main/java/org/example/uzgotuje/services/email/EmailService.java

```
@@ -10,23 +10,40 @@
import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Service;

+/**
+ * Service for sending emails.
+ */
@Service
@AllArgsConstructor
public class EmailService implements EmailSender {
+    /**
+     * The JavaMailSender used to send emails.
+     */
    private final JavaMailSender mailSender;
+
+    /**
+     * The logger for logging email sending errors.
+     */
    private final static Logger LOGGER = LoggerFactory.getLogger(EmailService.class);
+
+    /**
+     * Sends an email to the specified recipient asynchronously.
+     *
+     * @param to the recipient's email address
+     * @param email the content of the email to be sent
+     */
    @Override
    @Async
    public void send(String to, String email) {
```

src/main/java/org/example/uzgotuje/services/email/EmailValidator.java

```
@@ -5,11 +5,24 @@
import java.util.function.Predicate;
import java.util.regex.Pattern;

+ /**
+  * Service for validating email addresses.
+  */
@Service
public class EmailValidator implements Predicate<String> {
+     /**
+      * The pattern used to validate email addresses.
+      */
    private static final Pattern EMAIL_PATTERN = Pattern.compile(
            "^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+$"
    );
+
+     /**
+      * Tests whether the given email matches the email pattern.
+      *
+      * @param email the email address to be validated
+      * @return true if the email matches the pattern, false otherwise
+      */
    @Override
    public boolean test(String email) {
        return EMAIL_PATTERN.matcher(email).matches();
```

src/main/java/org/example/uzgotuje/services/fileStorage/FileStorageService.java

```
@@ -9,10 +9,22 @@
import java.nio.file.Paths;
import java.util.UUID;

+ /**
+  * Service for storing and managing files.
+  */
@Service
public class FileStorageService {
```

src/main/java/org/example/uzgotuje/services/recipe/CreateIngredientRequest.java

```
@@ -5,12 +5,23 @@
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.ToString;
-
+/**
+ * Request object for creating an ingredient.
+ */
@Getter
@EqualsAndHashCode
@ToString
public class CreateIngredientRequest {
+    /**
+     * The name of the ingredient.
+     */
    private final String name;
+
+    /**
+     * Constructs a new CreateIngredientRequest with the specified ingredient name.
+     *
+     * @param name the name of the ingredient
+     */
    @JsonCreator
    public CreateIngredientRequest(@JsonProperty("name") String name) {
        this.name = name;
```

src/main/java/org/example/uzgotuje/services/recipe/CreateRatingRequest.java

```
@@ -5,11 +5,21 @@
import lombok.Getter;
import lombok.ToString;

+ /**
+  * Request object for creating a rating for a recipe.
+  */
@Getter
@AllArgsConstructor
@EqualsAndHashCode
@ToString
public class CreateRatingRequest {
+     /**
+      * The ID of the recipe to be rated.
+      */
      private final Long recipeId;
+
+     /**
+      * The score given to the recipe.
+      */
      private final int score;
}
```

src/main/java/org/example/uzgotuje/services/recipe/CreateRecipeRequest.java

```
@@ -9,15 +9,41 @@
import org.example.uzgotuje.database.entity.recipe.Tag;
import org.springframework.web.multipart.MultipartFile;

+ /**
+  * Request object for creating a recipe.
+  */
@Getter
@AllArgsConstructor
@EqualsAndHashCode
@ToString
public class CreateRecipeRequest {
+     /**
+      * The name of the recipe.
+      */
        private final String name;
+
+     /**
+      * The description of the recipe.
+      */
        private final String description;
+
+     /**
+      * The type of the recipe (e.g., dessert, main course).
+      */
        private final String type;
+
+     /**
+      * The images associated with the recipe.
+      */
        private final MultipartFile[] images;
+
+     /**
+      * The tags associated with the recipe.
+      */
        private final Tag[] tags;
+
+     /**
+      * The ingredients used in the recipe.
+      */
        private final RecipeIngredient[] ingredients;
}
```

src/main/java/org/example/uzgotuje/services/recipe/CreateTagRequest.java

```
@@ -5,11 +5,21 @@
5  5   import lombok.Getter;
6  6   import lombok.ToString;
7  7
   8  + /**
   9  +  * Request object for creating a tag.
  10  +  */
8  11  @Getter
9  12  @EqualsAndHashCode
10 13  @AllArgsConstructor
11 14  @ToString
12 15  public class CreateTagRequest {
  16  +     /**
  17  +      * The name of the tag.
  18  +      */
13 19      private final String name;
  20  +
  21  +     /**
  22  +      * The type of the tag.
  23  +      */
14 24      private final String tagType;
15 25  }
```

src/main/java/org/example/uzgotuje/services/recipe/CreateIngredientRequest.java

```
@@ -1,18 +1,29 @@
package org.example.uzgotuje.services.recipe;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.ToString;
-
+ /**
+  * Request object for creating an ingredient.
+  */
@Getter
@EqualsAndHashCode
@ToString
public class CreateIngredientRequest {
+     /**
+      * The name of the ingredient.
+      */
        private final String name;
+
+     /**
+      * Constructs a new CreateIngredientRequest with the specified ingredient name.
+      *
+      * @param name the name of the ingredient
+      */
        @JsonCreator
        public CreateIngredientRequest(@JsonProperty("name") String name) {
            this.name = name;
        }
}
```

src/main/java/org/example/uzgotuje/services/recipe/CreateRatingRequest.java

```
@@ -1,15 +1,25 @@
package org.example.uzgotuje.services.recipe;

import lombok.AllArgsConstructor;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.ToString;

+ /**
+  * Request object for creating a rating for a recipe.
+  */
@Getter
@AllArgsConstructor
@EqualsAndHashCode
@ToString
public class CreateRatingRequest {
+     /**
+      * The ID of the recipe to be rated.
+      */
    private final Long recipeId;
+
+     /**
+      * The score given to the recipe.
+      */
    private final int score;
}
```

src/main/java/org/example/uzgotuje/services/recipe/CreateRecipeRequest.java

```
@@ -1,23 +1,49 @@
package org.example.uzgotuje.services.recipe;

import lombok.AllArgsConstructor;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.ToString;
import org.example.uzgotuje.database.entity.recipe.Ingredient;
import org.example.uzgotuje.database.entity.recipe.RecipeIngredient;
import org.example.uzgotuje.database.entity.recipe.Tag;
import org.springframework.web.multipart.MultipartFile;

+ /**
+  * Request object for creating a recipe.
+  */
@Getter
@AllArgsConstructor
@EqualsAndHashCode
@ToString
public class CreateRecipeRequest {
+     /**
+      * The name of the recipe.
+      */
        private final String name;
+
+     /**
+      * The description of the recipe.
+      */
        private final String description;
+
+     /**
+      * The type of the recipe (e.g., dessert, main course).
+      */
        private final String type;
+
+     /**
+      * The images associated with the recipe.
+      */
        private final MultipartFile[] images;
+
+     /**
```

```
19  33          private final String type;
    34  +
    35  +       /**
    36  +        * The images associated with the recipe.
    37  +        */
20  38          private final MultipartFile[] images;
    39  +
    40  +       /**
    41  +        * The tags associated with the recipe.
    42  +        */
21  43          private final Tag[] tags;
    44  +
    45  +       /**
    46  +        * The ingredients used in the recipe.
    47  +        */
22  48          private final RecipeIngredient[] ingredients;
23  49      }
```

src/main/java/org/example/uzgotuje/services/recipe/CreateTagRequest.java

```
@@ -1,15 +1,25 @@
package org.example.uzgotuje.services.recipe;

import lombok.AllArgsConstructor;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.ToString;

+ /**
+  * Request object for creating a tag.
+  */
@Getter
@EqualsAndHashCode
@AllArgsConstructor
@ToString
public class CreateTagRequest {
+     /**
+      * The name of the tag.
+      */
      private final String name;
+
+     /**
+      * The type of the tag.
+      */
      private final String tagType;
 }
```

src/main/java/org/example/uzgotuje/services/recipe/RecipeSearchRequest.java

```
@@ -1,21 +1,37 @@
 ~package org.example.uzgotuje.services.recipe;
 ~
 ~
 ~import lombok.*;
 ~import org.example.uzgotuje.database.entity.recipe.Tag;

 import java.util.List;

+ /**
+  * Request object for searching recipes.
+  */
 @Setter
 @Getter
 @AllArgsConstructor
 @EqualsAndHashCode
 @ToString
 public class RecipeSearchRequest {
+    /**
+     * The tags associated with the recipes to search for.
+     */
     private Tag[] tags;
+
+    /**
+     * The type of the recipes to search for (e.g., dessert, main course).
+     */
     private final String type;
-    private final String name;
-    private final boolean sortByRatingDesc;

+    /**
+     * The name of the recipes to search for.
+     */
+    private final String name;

+    /**
+     * Whether to sort the search results by rating in descending order.
+     */
+    private final boolean sortByRatingDesc;
 }
```

src/main/java/org/example/uzgotuje/services/recipe/RecipeService.java

```
@@ -1,31 +1,72 @@
package org.example.uzgotuje.services.recipe;

import lombok.AllArgsConstructor;
import org.example.uzgotuje.database.entity.auth.User;
import org.example.uzgotuje.database.entity.auth.UserRoles;
import org.example.uzgotuje.database.entity.recipe.*;
import org.example.uzgotuje.database.repository.recipe.*;
import org.example.uzgotuje.services.authorization.AuthenticationService;
import org.example.uzgotuje.services.fileStorage.FileStorageService;
import org.springframework.data.domain.Sort;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.util.*;

+ /**
+  * Service class for managing recipes.
+  */
@Service
@AllArgsConstructor
public class RecipeService {
+     /**
+      * Repository for managing recipes.
+      */
      private final RecipeRepository recipeRepository;
+
+     /**
+      * Repository for managing tags.
+      */
      private final TagRepository tagRepository;
+
+     /**
+      * Repository for managing ingredients.
+      */
      private final IngredientRepository ingredientRepository;
+
+     /**
```

```
        27  +
        28  +     /**
        29  +      * Repository for managing tags.
        30  +      */
21      31          private final TagRepository tagRepository;
        32  +
        33  +     /**
        34  +      * Repository for managing ingredients.
        35  +      */
22      36          private final IngredientRepository ingredientRepository;
        37  +
        38  +     /**
        39  +      * Repository for managing ratings.
        40  +      */
23      41          private final RatingRepository ratingRepository;
        42  +
        43  +     /**
        44  +      * Repository for managing favorites.
        45  +      */
24      46          private final FavoriteRepository favoriteRepository;
        47  +
        48  +     /**
        49  +      * Service for managing authentication.
        50  +      */
25      51          private final AuthenticationService authenticationService;
        52  +
        53  +     /**
        54  +      * Service for managing file storage.
        55  +      */
26      56          private final FileStorageService fileStorageService;
        57  +
        58  +     /**
        59  +      * Repository for managing comments.
        60  +      */
27      61          private final CommentRepository commentRepository;
28      62
        63  +     /**
        64  +      * Creates a new ingredient.
        65  +      *
        66  +      * @param request the request object containing ingredient details
        67  +      * @param cookieValue the cookie value for authentication
        68  +      * @return a string indicating the result of the operation
        69  +      */
29      70          public String createIngredient(CreateIngredientRequest request, String cookieValue) {
```

```
+    /**
+     * Creates a new ingredient.
+     *
+     * @param request the request object containing ingredient details
+     * @param cookieValue the cookie value for authentication
+     * @return a string indicating the result of the operation
+     */
    public String createIngredient(CreateIngredientRequest request, String cookieValue) {
        if(cookieValue == null) {
            return "Unauthorized";
        }
        //check if user is logged in
        if (authenticationService.validateCookieAndGetUser(cookieValue).getAppUserRole() != UserRoles.ADMIN) {
            return "Unauthorized";
        }
        //check if ingredient already exists
        if (ingredientRepository.findByName(request.getName()).isPresent()) {
            return "Bad request";
        }
        Ingredient ingredient = new Ingredient(request.getName());
        ingredientRepository.save(ingredient);
        return "Success";
    }

+    /**
+     * Creates a new tag.
+     *
+     * @param request the request object containing tag details
+     * @param cookieValue the cookie value for authentication
+     * @return a string indicating the result of the operation
+     */
    public String createTag(CreateTagRequest request, String cookieValue) {
        if(cookieValue == null) {
            return "Unauthorized";
        }
        //check if user is logged in
        if (authenticationService.validateCookieAndGetUser(cookieValue).getAppUserRole() != UserRoles.ADMIN) {
            return "Unauthorized";
        }
```

```
     112 +      * Checks the validity of a tag request.
     113 +      *
     114 +      * @param request the request object containing tag details
     115 +      * @return true if the request is valid, false otherwise
     116 +      */
  63 117        private boolean checkTagRequestValidity(CreateTagRequest request) {
  64 118            if(request.getName() == null || request.getName().isEmpty()) {
  65 119                return false;
          @@ -73,6 +127,13 @@
  73 127            return true;
  74 128        }
  75 129
     130 +      /**
     131 +      * Creates a new recipe.
     132 +      *
     133 +      * @param request the request object containing recipe details
     134 +      * @param cookieValue the cookie value for authentication
     135 +      * @return a string indicating the result of the operation
     136 +      */
  76 137        public String createRecipe(CreateRecipeRequest request, String cookieValue) {
  77 138            if(cookieValue == null) {
  78 139                return "Unauthorized";
          @@ -118,14 +179,34 @@
 118 179            return "Success";
 119 180        }
 120 181
     182 +      /**
     183 +      * Retrieves a recipe by its ID.
     184 +      *
     185 +      * @param recipeId the ID of the recipe
     186 +      * @return the recipe object, or null if not found
     187 +      */
 121 188        public Recipe getRecipe(Long recipeId) {
 122 189            return recipeRepository.findById(recipeId).orElse(null);
 123 190        }
 124 191
     192 +      /**
     193 +      * Retrieves all recipes.
     194 +      *
     195 +      * @return a list of all recipes
     196 +      */
 125 197        public List<Recipe> getRecipes() {
 126 198            return recipeRepository.findAll();
 127 199        }
```

```
125  197        public List<Recipe> getRecipes() {
126  198            return recipeRepository.findAll();
127  199        }
128  200
     201    +   /**
     202    +    * Searches for recipes based on tags, name, type, and sort order.
     203    +    *
     204    +    * @param tags the tags to search for
     205    +    * @param name the name to search for
     206    +    * @param type the type of recipes to search for
     207    +    * @param sortByRatingDesc whether to sort by rating in descending order
     208    +    * @return a list of recipes matching the search criteria
     209    +    */
129  210        public List<Recipe> searchRecipes(List<Tag> tags, String name, RecipeTypes type, boolean sortByRatingDesc) {
130  211            Specification<Recipe> spec = Specification.where(RecipeSpecification.hasTags(new HashSet<>(tags)))
131  212                    .and(RecipeSpecification.hasName(name))
          @@ -136,8 +217,12 @@
136  217            return recipeRepository.findAll(spec, sort);
137  218        }
138  219
139       -
140       -
     220    +   /**
     221    +    * Checks the validity of a create recipe request.
     222    +    *
     223    +    * @param request the request object containing recipe details
     224    +    * @return true if the request is valid, false otherwise
     225    +    */
141  226        private boolean checkCreateRecipeRequestValidity(CreateRecipeRequest request) {
142  227            if(request.getName() == null || request.getName().isEmpty()) {
143  228                return false;
          @@ -187,6 +272,14 @@
187  272            return true;
188  273        }
189  274
     275    +   /**
     276    +    * Adds a rating to a recipe.
     277    +    *
     278    +    * @param recipeId the ID of the recipe
     279    +    * @param score the score to be given
     280    +    * @param cookieValue the cookie value for authentication
```

```
189  274
     275  +      /**
     276  +       * Adds a rating to a recipe.
     277  +       *
     278  +       * @param recipeId the ID of the recipe
     279  +       * @param score the score to be given
     280  +       * @param cookieValue the cookie value for authentication
     281  +       * @return a string indicating the result of the operation
     282  +       */
190  283         public String addRating(Long recipeId, Integer score, String cookieValue) {
191  284             if(cookieValue == null) {
192  285                 return "Unauthorized";
          @@ -212,6 +305,13 @@
212  305             return "Success";
213  306         }
214  307
     308  +      /**
     309  +       * Retrieves the rating given by a user to a recipe.
     310  +       *
     311  +       * @param recipeId the ID of the recipe
     312  +       * @param cookieValue the cookie value for authentication
     313  +       * @return the rating score, or 0 if not found
     314  +       */
215  315         public Integer getUserRating(Long recipeId, String cookieValue) {
216  316             if(cookieValue == null) {
217  317                 return 0;
          @@ -228,7 +328,13 @@
228  328             return dbRating.getScore();
229  329         }
230  330
231       -      //update the value of favorite if it exists it deletes it, if it doesn't it creates it
     331  +      /**
     332  +       * Updates the favorite status of a recipe for a user.
     333  +       *
     334  +       * @param recipeId the ID of the recipe
     335  +       * @param cookieValue the cookie value for authentication
     336  +       * @return a string indicating the result of the operation
     337  +       */
232  338         public String updateFavorite(Long recipeId, String cookieValue) {
233  339             if(cookieValue == null) {
234  340                 return "Unauthorized";
          @@ -249,7 +355,12 @@
249  355             return "Success";
250  356         }
```

```
250  356         }
251  357
252       -      //get all favorite recipes of the user
          358 +  /**
          359 +   * Retrieves all favorite recipes of a user.
          360 +   *
          361 +   * @param cookieValue the cookie value for authentication
          362 +   * @return a list of favorite recipes
          363 +   */
253  364         public List<Recipe> getFavoriteRecipes(String cookieValue) {
254  365             if(cookieValue == null) {
255  366                 return new ArrayList<>();
     @@ -263,14 +374,32 @@
263  374             return recipes;
264  375         }
265  376
          377 +  /**
          378 +   * Retrieves all tags.
          379 +   *
          380 +   * @return a list of all tags
          381 +   */
266  382         public List<Tag> getTags() {
267  383             return tagRepository.findAll();
268  384         }
269  385
          386 +  /**
          387 +   * Retrieves all ingredients.
          388 +   *
          389 +   * @return a list of all ingredients
          390 +   */
270  391         public List<Ingredient> getIngredients() {
271  392             return ingredientRepository.findAll();
272  393         }
273  394
          395 +  /**
          396 +   * Creates a new comment for a recipe.
          397 +   *
          398 +   * @param recipeId the ID of the recipe
          399 +   * @param content the content of the comment
          400 +   * @param cookieValue the cookie value for authentication
          401 +   * @return a string indicating the result of the operation
          402 +   */
274  403         public String createComment(Long recipeId, String content, String cookieValue) {
275  404             if(cookieValue == null) {
276  405                 return "Unauthorized";
```

```
390 +     */
270 391       public List<Ingredient> getIngredients() {
271 392           return ingredientRepository.findAll();
272 393       }
273 394
    395 +     /**
    396 +      * Creates a new comment for a recipe.
    397 +      *
    398 +      * @param recipeId the ID of the recipe
    399 +      * @param content the content of the comment
    400 +      * @param cookieValue the cookie value for authentication
    401 +      * @return a string indicating the result of the operation
    402 +      */
274 403       public String createComment(Long recipeId, String content, String cookieValue) {
275 404           if(cookieValue == null) {
276 405               return "Unauthorized";
    @@ -289,6 +418,12 @@
289 418           return "Success";
290 419       }
291 420
    421 +     /**
    422 +      * Retrieves all comments of a recipe.
    423 +      *
    424 +      * @param recipeId the ID of the recipe
    425 +      * @return a list of comments for the recipe
    426 +      */
292 427       public List<Comment> getCommentsOfRecipe(Long recipeId) {
293 428           Optional<Recipe> recipe = recipeRepository.findById(recipeId);
294 429           if(recipe.isEmpty()) {
    @@ -297,6 +432,12 @@
297 432           return commentRepository.findByRecipe(recipe.get()).orElse(new ArrayList<>());
298 433       }
299 434
    435 +     /**
    436 +      * Retrieves random recipes by type.
    437 +      *
    438 +      * @param type the type of recipes to retrieve
    439 +      * @return a list of random recipes of the specified type
    440 +      */
300 441       public List<Recipe> getRandomRecipesByType(String type) {
301 442           if(type == null || type.isEmpty()) {
```

src/main/java/org/example/uzgotuje/services/token/ConfirmationTokenService.java

```
@@ -1,22 +1,37 @@
package org.example.uzgotuje.services.token;

import lombok.AllArgsConstructor;
import org.example.uzgotuje.database.entity.auth.ConfirmationToken;
import org.example.uzgotuje.database.entity.auth.User;
import org.example.uzgotuje.database.repository.auth.ConfirmationTokenRepository;
import org.springframework.stereotype.Service;

+ /**
+  * Service class for managing confirmation tokens.
+  */
@Service
@AllArgsConstructor
public class ConfirmationTokenService {
+     /**
+      * Repository for managing confirmation tokens.
+      */
    private final ConfirmationTokenRepository confirmationTokenRepository;

+     /**
+      * Saves a confirmation token.
+      *
+      * @param token the confirmation token to save
+      */
    public void saveConfirmationToken(ConfirmationToken token) {
        confirmationTokenRepository.save(token);
    }

+     /**
+      * Deletes a confirmation token by user.
+      *
+      * @param user the user whose confirmation token is to be deleted
+      */
    public void deleteConfirmationTokenByUser(User user) {
        confirmationTokenRepository.deleteByUser(user);
    }
-
}
```

src/main/java/org/example/uzgotuje/services/token/TokenResponse.java

```
@@ -1,10 +1,16 @@
package org.example.uzgotuje.services.token;

import lombok.AllArgsConstructor;
import lombok.Getter;

+ /**
+  * Response object for token-related operations.
+  */
@Getter
@AllArgsConstructor
public class TokenResponse {
+     /**
+      * The message associated with the token response.
+      */
    private final String message;
}
```