Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное

образовательное учреждение высшего образования

«Самарский национальный исследовательский университет

имени академика С.П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

Финальный отчёт

Дисциплина: «Технологии сетевого программирования»

Финализация приложения «Книга рецептов»

Выполнил: Диш Софья, Шкуркина Мария

Группа: 6303-010302D

Самара, 2025

АРХИТЕКТУРА ПРИЛОЖЕНИЯ

Проект представляет собой веб-приложение, предназначенное для работы с рецептами. Архитектура построена по модели "клиент-сервер" и включает следующие ключевые компоненты:

1. Клиентская часть (HTML + JS):
- Отвечает за отображение форм регистрации, входа, профиля и различных страниц с рецептами.
- Реализует взаимодействие с сервером через fetch-запросы с JWT-аутентификацией.

2. Серверная часть (Java + Spring):
- Обрабатывает все HTTP-запросы.
- Реализует REST API.
- Обеспечивает JWT-аутентификацию и управление пользователями, событиями и реакциями.

3. База данных (PostgreSQL):
- Хранит информацию о пользователях и рецептах.

4. Docker-контейнеризация:
- db (PostgreSQL):
Используется стандартный образ PostgreSQL с добавлением необходимых настроек для подключения API-контейнера.

- app (Spring Boot, HTML+JS)
Использует стандартизированный Dockerfile для сборки JAR-файлов с помощью Maven. Доступен по адресу: localhost:8080. Сервер ожидает соединение с базой данных.

ВЗАИМОДЕЙСТВИЕ КОМПОНЕНТОВ

- Запрос от клиента: Клиент отправляет запрос на сервер (например, для получения рецепта).
- Обработка запроса сервером: Сервер принимает запрос, обрабатывает его и обращается к базе данных для извлечения необходимой информации.
- Ответ базы данных: База данных возвращает запрашиваемую информацию серверу.
- Ответ сервера клиенту: Сервер формирует ответ и отправляет его обратно клиенту (например, возвращает рецепты определенной категории).

СТРУКТУРА БАЗЫ ДАННЫХ

- User(id_user,name ,password, bday, phone, mail, user_role): Хранит информацию о пользователях.

- Recipe(id_recipe, id_user, title, id_category, description, manual, time, picture): Хранит информацию о рецептах.

- Ingredient(id_ingridient, title, unit_measure): Хранит информацию об ингредиентах.

- Composition_recept(id_recipes, id_ingridient, quantity): Хранит информацию о том, какие ингредиенты входят в состав рецепта.

- Categoriy(id_category, title): Хранит информацию о категориях.

- Favorite_recipe(id_user, id_recipe): Хранит информацию о том, какие рецепты добавлены в избранное.

- Token(id, access_token, refresh_token, is_logged_out, user_id): Хранит информацию о токенах.

СТРУКТУРА API

Метод	**Название метода**	**URL**
POST	Регистрация пользователя	/a/registration
POST	Вход пользователя	/a/login
POST	Смена пароля	/a/password_change
POST	Добавление рецепта	/api/recipes/create
GET	Получение списка рецептов	/api/recipes/search
GET	Получение выбранного рецепта	/api/recipes/getRecipe/{id}
GET	Получить все ингредиенты	/api/ingredients
POST	Создание ингредиента	/api/ingredients/create
GET	Вывод всех категорий	/api/categories
POST	Создание категории	/api/categories/create
GET	Получение одной категории	/api/categories/getCategory/{id}

Метод	Название метода	URL
GET	Получение профиля пользователя	/api/users/profile
GET	Получение всех рецептов пользователя	/api/users/myRecipe
GET	Получение любимых рецептов	/api/users/myFavouriteRecipe
POST	Добавление рецепта в избранное	/api/users/favourite-recipes/{recipeId}
DELETE	Удаление рецепта из избранного	/api/users/removeFavourite/{recipeId}

СТЕК ТЕХНОЛОГИЙ

- PostgreSQL
- Java + Spring
- HTML + JavaScript
- DevOps: Docker, Docker Compose

ДЕМОНСТРАЦИЯ (USE CASES)

1. Регистрация и вход:
- Пользователь переходит на /registration/ и создаёт учётную запись.
- Получает access_token и redirect'ится на /login/.

2. Отображение рецептов:
- Загружается соответствующая страница рецептов.
- Рецепты представляют собой карточки с названием и временем.

3. Отображение рецепта:
- Показ всей информации рецепта (название, категория, инструкция, описание, ингредиенты, время приготовления).
- Поиск по фильтру (ингредиенту, названию, категории).

4. Создание рецепта:
- Клик по кнопке создания — открывается модальное окно.
- Пользователь вводит данные, подтверждает — новый рецепт появляется в списке.

5. Добавление рецепта в избранное:
- На карточке можно поставить лайк на рецепт, и он появится на отдельной странице.

6. Создание категории и ингредиента:
- Кнопки "Создать категорию" и "Создать ингредиент" доступны только админу.

7. Выход из системы:
- Пользователь может выйти из системы, тогда ему будет доступен только просмотр существующих рецептов.

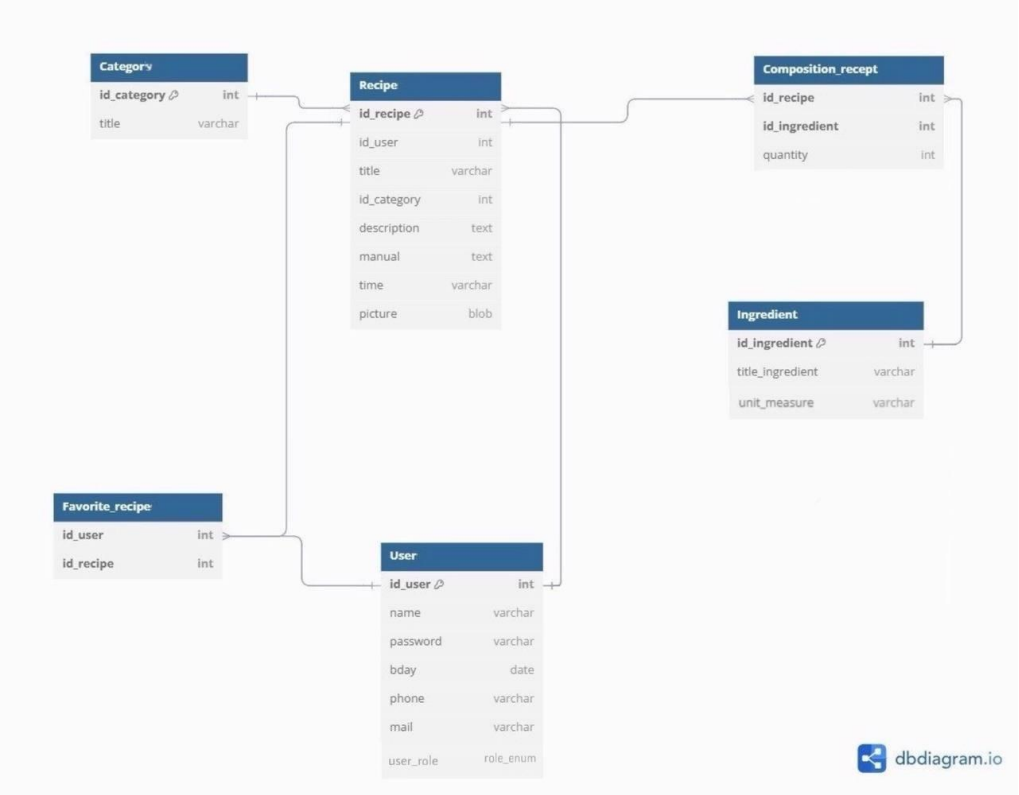

Рисунок 1– Диаграмма БД

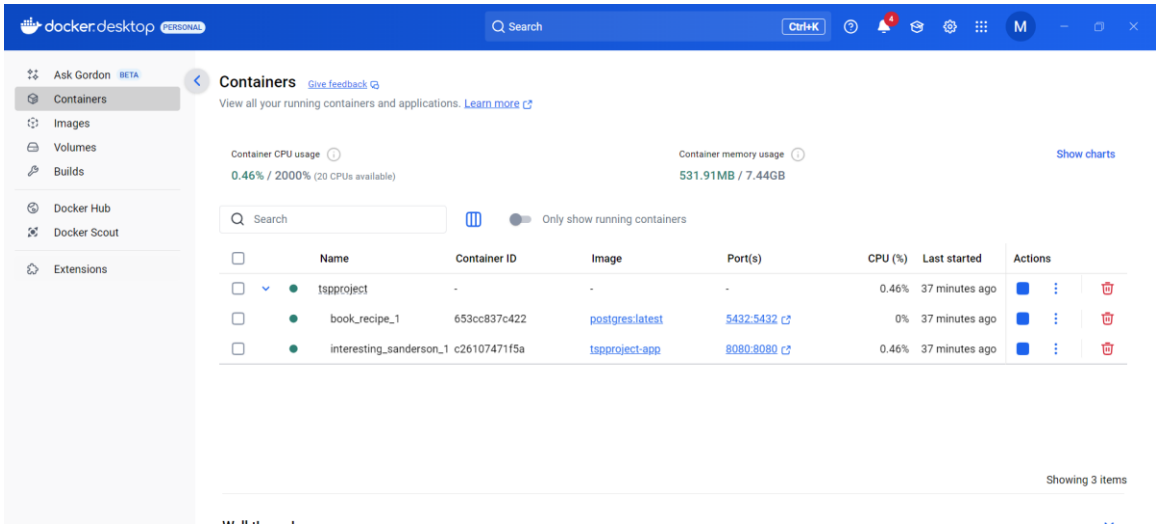

Рисунок 2 – Работающие контейнеры

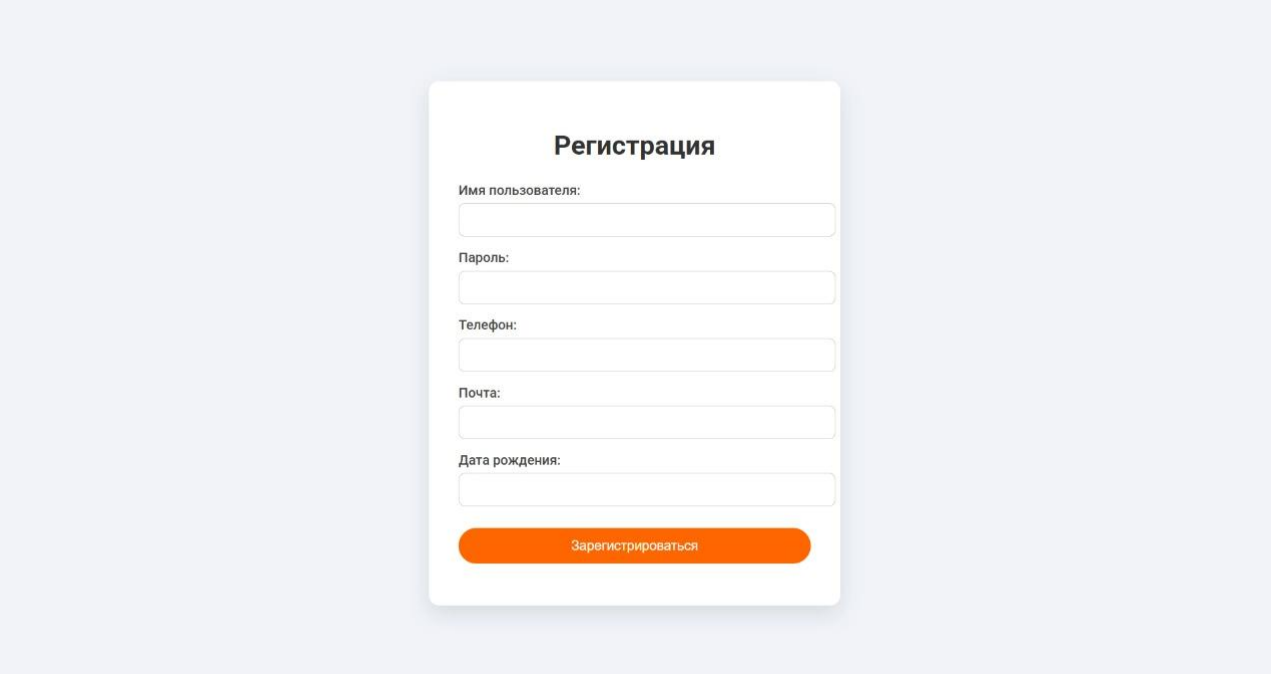

Рисунок 3 – Страница регистрации

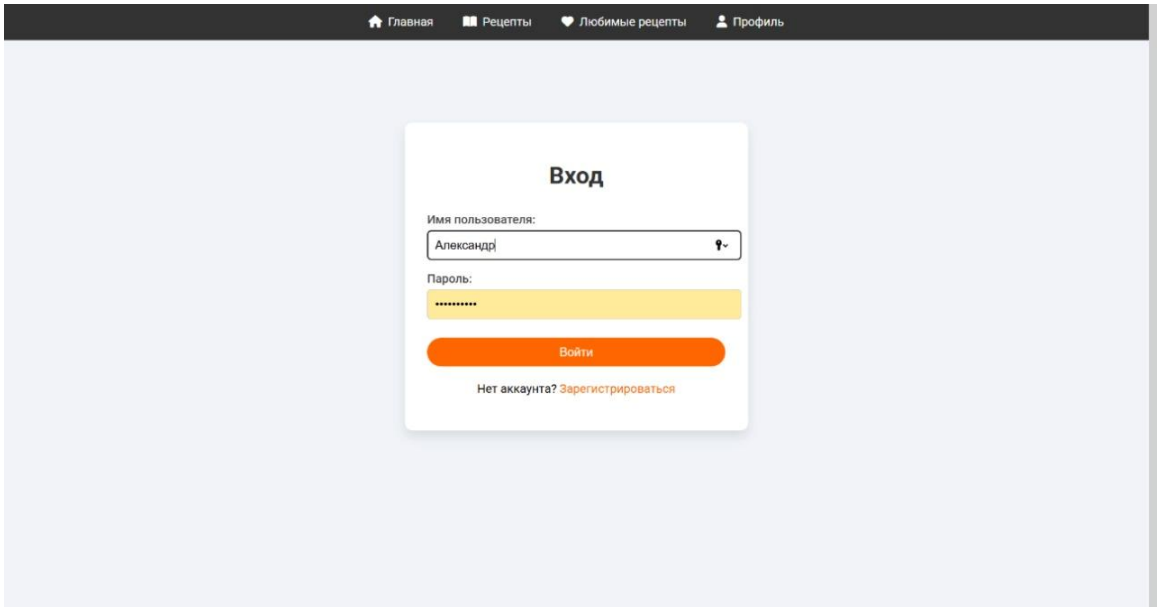

Рисунок 4 – Страница входа

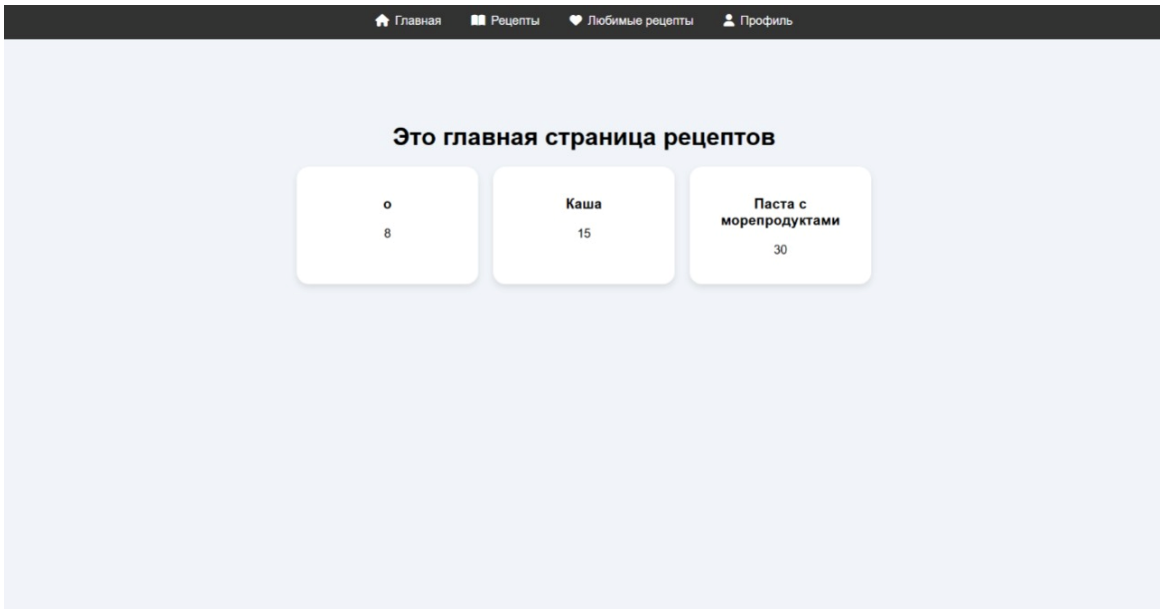

Рисунок 5 – Главная страница

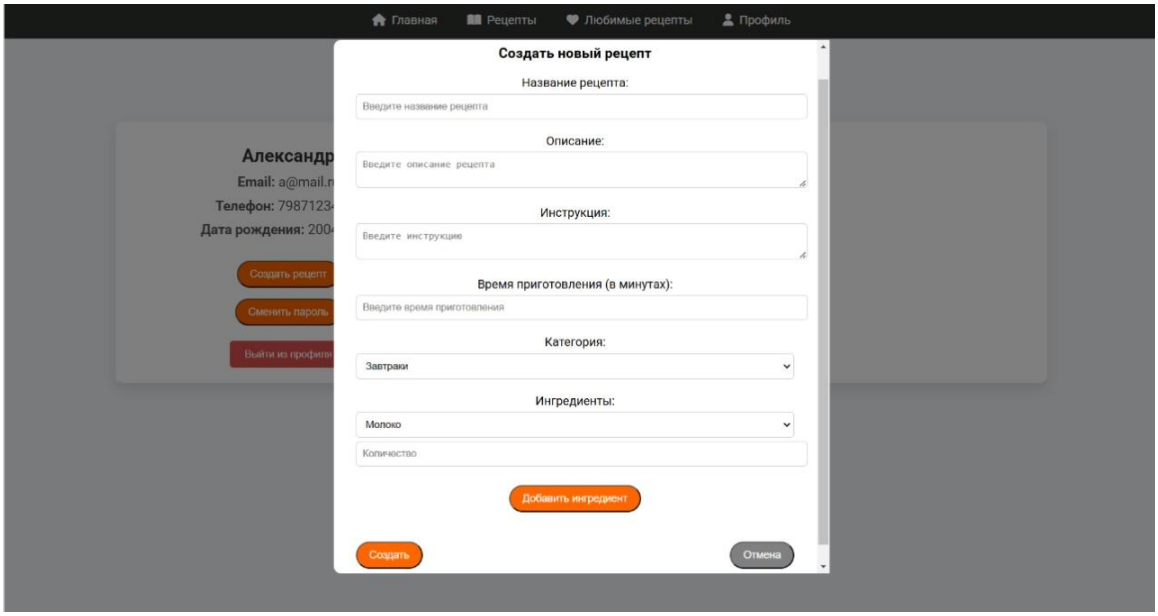

Рисунок 6 – Окно создания рецепта

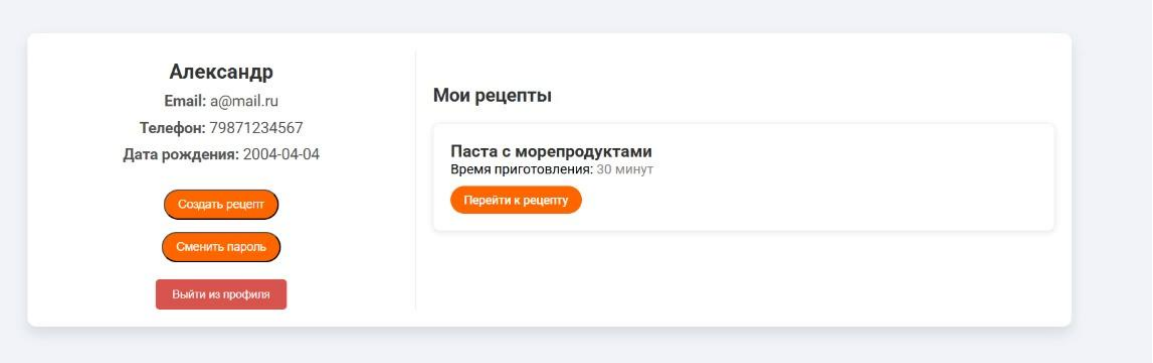

Рисунок 7 – Профиль обычного пользователя

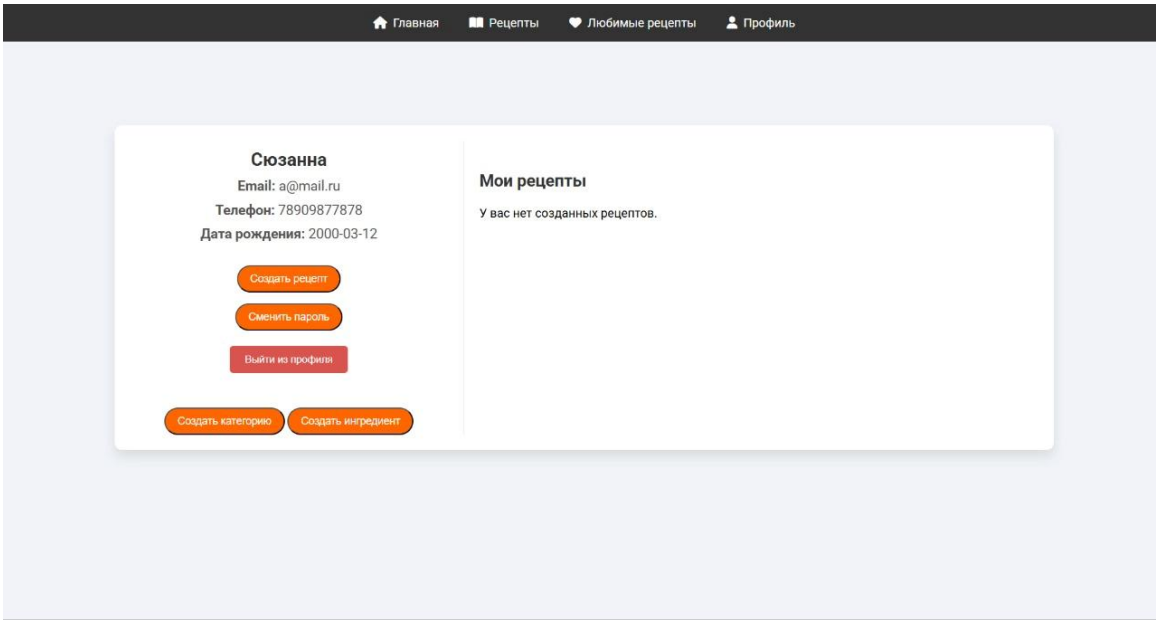

Рисунок 8 – Профиль администратора

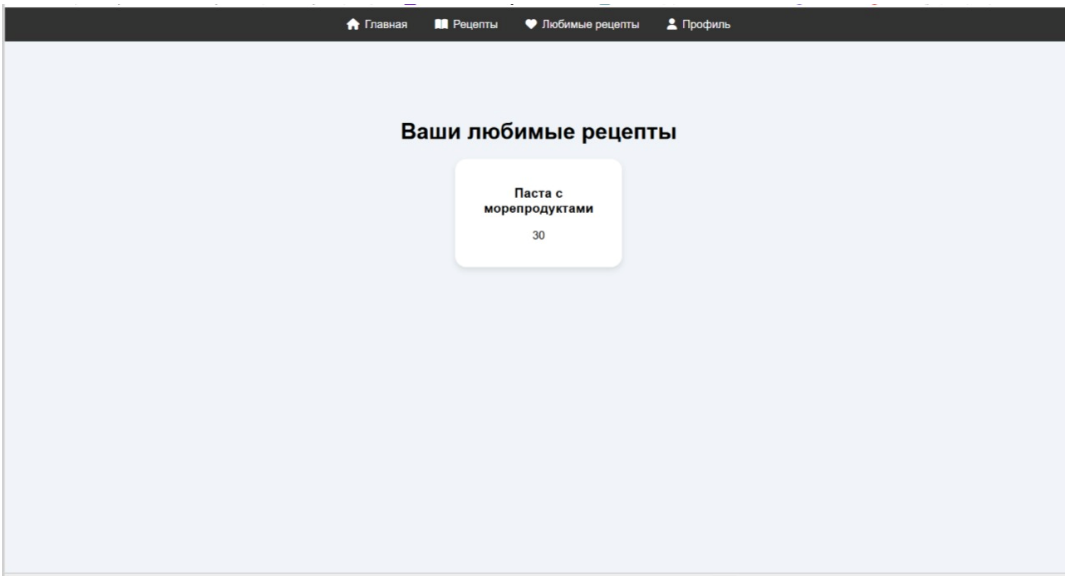

Рисунок 9 – Любимые рецепты

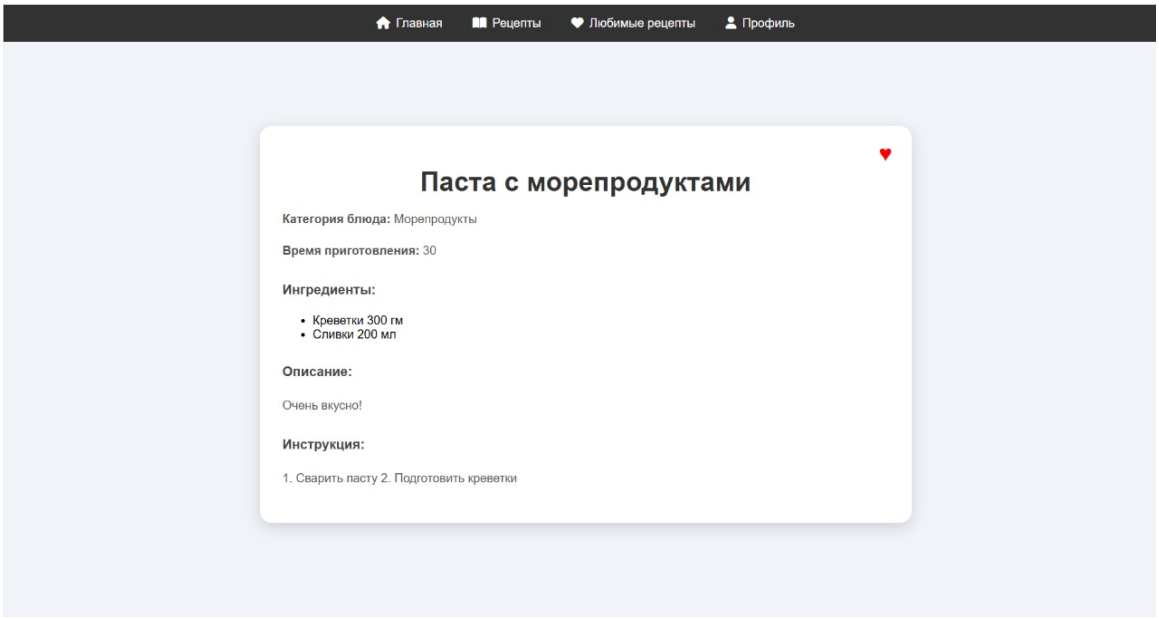

Рисунок 10 – Страница рецепта

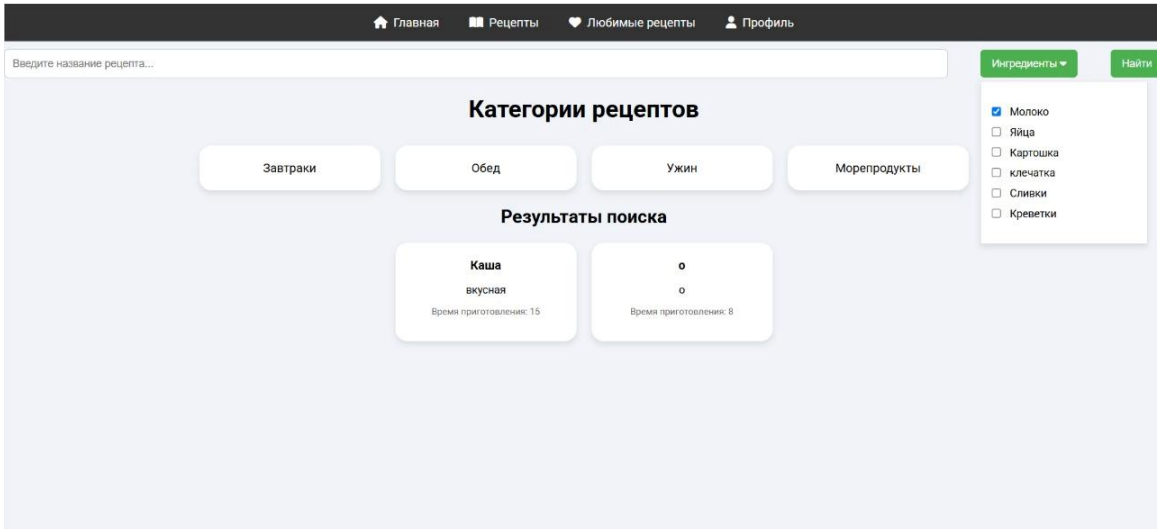

Рисунок 11 – Страница поиска рецептов по фильтрам, названию и категории

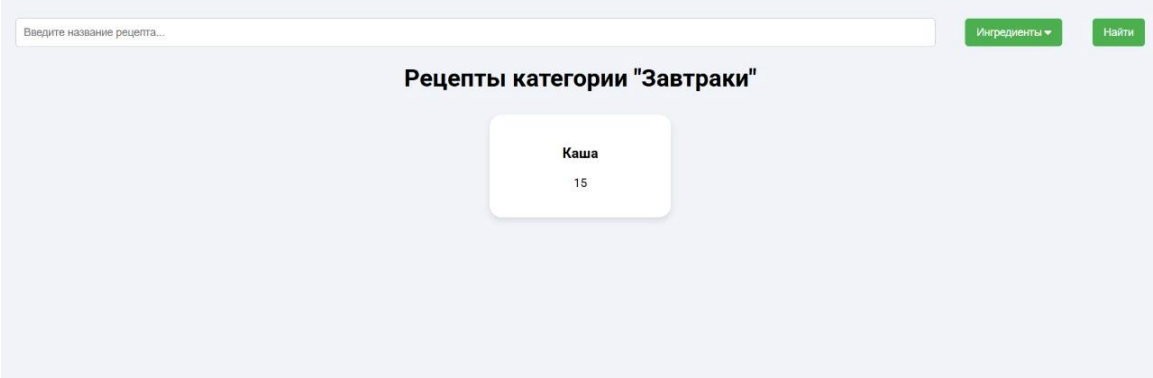

Рисунок 12 – Рецепты выбранной категории

КОД ПРОГРАММЫ

UserService

```
@Service
public class UserService implements UserDetailsService {

    private final UserRepository userRepository;
    private final PasswordService passwordUtil;

    @Autowired
    public UserService(UserRepository userRepository, PasswordService passwordUtil) {
        this.userRepository = userRepository;
        this.passwordUtil = passwordUtil;
    }

    @Autowired
    private EntityManager entityManager;
    // Проверка пароля пользователя, то есть вход
    public boolean checkPassword(String rawPassword, String email) {
        User user = userRepository.findByMail(email);
        return user != null && passwordUtil.matches(rawPassword, user.getPassword());
    }

    // Метод для поиска пользователя по email
    public User getUserByEmail(String email) {
        return userRepository.findByMail(email);
    }

    // Метод для поиска пользователя по имени
    public User getUserByName(String name) {
        return userRepository.findByName(name)
                .orElseThrow(() -> new RuntimeException("User not found"));
    }

    // Метод для поиска пользователя по ID
    public User getUserById(int id) {
        return userRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("User not found"));
    }

    // Метод для удаления пользователя по его ID
    public void deleteUser(int id) {

        userRepository.deleteById(id);
    }
    public User updateUser(Integer id,User updatedUser) {

        User existingUser = getUserById(id);
        // При изменении пароля – хэшируем заново
        if (!existingUser.getPassword().equals(updatedUser.getPassword())) {
            existingUser.setPassword(passwordUtil.encodePassword(updatedUser.getPassword()));
        }
```

```
        existingUser.setBday(updatedUser.getBday());
        existingUser.setMail(updatedUser.getMail());
        existingUser.setPhone(updatedUser.getPhone());
        existingUser.setName(updatedUser.getUsername());

        return userRepository.save(existingUser);
    }


    public boolean existsByUsername(String username) {
        User user = userRepository.findByName(username).orElse(null);
        if (user != null) {
            return true;
        }
        return false;
    }

    public boolean existsByEmail(String email) {
        User user = userRepository.findByMail(email);
        if (user != null) {
            return true;
        }
        return false;
    }

    @Override
    public UserDetails loadUserByUsername(String name) throws UsernameNotFoundException {
        return userRepository.findByName(name)
                .orElseThrow(() -> new UsernameNotFoundException("Пользователь не найден"));
    }
}
```

RecipeService

```
@Service
@Transactional //в одной транзакции
public class RecipeService {
    private final RecipeRepository recipeRepository;
    @Autowired
    private CompositionRecipeRepository compositionRecipeRepository;

    @Autowired
    public RecipeService(RecipeRepository recipeRepository) {
        this.recipeRepository = recipeRepository;
    }

    public Recipe createRecipe(Recipe recipe) {
        return recipeRepository.save(recipe);
    }
    // Метод для получения рецепта по его ID
    @Transactional
    public Recipe getRecipeById(int id) {
        return recipeRepository.findById(id)
```

```
            .orElseThrow(() -> new RuntimeException("Recipe not found"));
    }

    // Метод для получения всех рецептов пользователя по его ID
    @Transactional
    public List<Recipe> getRecipesByUserId(User userId) {

        return recipeRepository.findByUserId(userId)
                .orElseThrow(() -> new RuntimeException("Recipes not found"));
    }

    // Метод для получения всех рецептов по категории
    @Transactional
    public Optional<List<Recipe>> getRecipesByCategoryId(Category categoryId) {
        return recipeRepository.findByCategory(categoryId);
    }

    // Метод для поиска рецептов по части названия (с использованием LIKE)
    public Optional<List<Recipe>>searchRecipesByTitle(String title) {
        return recipeRepository.findByTitleContainingIgnoreCase(title);
    }

    // Метод для удаления рецепта по его ID
    public void deleteRecipe(int id) {
        recipeRepository.deleteById(id);
    }

    //Апдейт рецепта
    public Recipe updateRecipe(Recipe updatedRecipe) {

        Recipe existingRecipe = getRecipeById(updatedRecipe.getId());
        existingRecipe.setTitle(updatedRecipe.getTitle());
        existingRecipe.setDescription(updatedRecipe.getDescription());
        existingRecipe.setManual(updatedRecipe.getManual());
        existingRecipe.setTime(updatedRecipe.getTime());
        return recipeRepository.save(existingRecipe);
    }

    public List<RecipeAnswerDTO> searchRecipes(String title, Long categoryId, List<String>
ingredients) {
        List<Recipe> recipes = recipeRepository.findByFilters(title, categoryId, ingredients);

        return recipes.stream()
                .map(r -> {
                    // Преобразуем CompositionRecipe в IngredientDTO
                    List<IngredientDTO> ingredientDTOs = r.getIngredients().stream()
                            .map(comp -> {
                                IngredientDTO dto = new IngredientDTO();
                                dto.setIngredientId(comp.getIngredient().getId_ingredient());
                                dto.setIngredientTitle(comp.getIngredient().getTitle());
                                dto.setIngredientUnit(comp.getIngredient().getUnitMeasure());
                                dto.setQuantity(comp.getQuantity());
                                return dto;
                            })
                            .collect(Collectors.toList());
```

```
                // Возвращаем рецепт с ингредиентами
                return new RecipeAnswerDTO(
                        r.getId(),
                        r.getCategory().getId_category(),
                        r.getTitle(),
                        r.getDescription(),
                        r.getManual(),
                        r.getTime(),
                        ingredientDTOs
                );
            })
            .collect(Collectors.toList());
    }


}
```

JWTService

```
@Service
public class JwtService {

    @Value("${jwt.secret}")
    private String secretKey;

    @Value("${jwt.access-token-expiration}")
    private long accessTokenExpiration;

    @Value("${jwt.refresh-token-expiration}")
    private long refreshTokenExpiration;

    private final TokenRepository tokenRepository;

    public JwtService(TokenRepository tokenRepository) {
        this.tokenRepository = tokenRepository;
    }
    private SecretKey getSigningKey() {

        byte[] keyBytes = Decoders.BASE64URL.decode(secretKey);

        return Keys.hmacShaKeyFor(keyBytes);
    }
    private String generateToken(User user, long expiryTime) {
        JwtBuilder builder = Jwts.builder()
                .subject(user.getUsername())
                .issuedAt(new Date(System.currentTimeMillis()))
                .expiration(new Date(System.currentTimeMillis() + expiryTime))
                .signWith(getSigningKey());

        return builder.compact();
    }
    public String generateAccessToken(User user) {

        return generateToken(user, accessTokenExpiration);
```

```
    }
    public String generateRefreshToken(User user) {

        return generateToken(user, refreshTokenExpiration);
    }
    private Claims extractAllClaims(String token) {

        JwtParserBuilder parser = Jwts.parser();
        parser.verifyWith(getSigningKey());

        return parser.build()
                .parseSignedClaims(token)
                .getPayload();
    }
    public <T> T extractClaim(String token, Function<Claims, T> resolver) {

        Claims claims = extractAllClaims(token);

        return resolver.apply(claims);
    }
    public String extractUsername(String token) {
        return extractClaim(token, Claims::getSubject);
    }
    private Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration);
    }
    private boolean isAccessTokenExpired(String token) {
        return !extractExpiration(token).before(new Date());
    }
    public boolean isValid(String token, UserDetails user) {

        String username = extractUsername(token);

        boolean isValidToken = tokenRepository.findByAccessToken(token)
                .map(t -> !t.isLoggedOut()).orElse(false);

        return username.equals(user.getUsername())
                && isAccessTokenExpired(token)
                && isValidToken;
    }
    public boolean isValidRefresh(String token, User user) {

        String username = extractUsername(token);

        boolean isValidRefreshToken = tokenRepository.findByRefreshToken(token)
                .map(t -> !t.isLoggedOut()).orElse(false);

        return username.equals(user.getUsername())
                && isAccessTokenExpired(token)
                && isValidRefreshToken;
    }
}
```

API-контроллеры

AuthenticationController

```
@RestController
@RequestMapping("/a")
public class AuthenticationController {
   @Autowired
   private final AuthenticationService authenticationService;

   private final UserService userService;

   public AuthenticationController(AuthenticationService authenticationService, UserService
userService) {
      this.authenticationService = authenticationService;
      this.userService = userService;
   }
   @PostMapping("/registration")
   public ResponseEntity<List<String>> register(
         @Valid
         @RequestBody RegistrationRequestDto registrationDto, BindingResult bindingResult) {
      if (bindingResult.hasErrors()) {
         List<String> errors = bindingResult.getFieldErrors()
               .stream()
               .map(DefaultMessageSourceResolvable::getDefaultMessage)
               .collect(Collectors.toList());
         return ResponseEntity.badRequest().body(errors);  // Возвращаем ошибки
      }

      try {
         authenticationService.register(registrationDto);
         return ResponseEntity.ok(Collections.singletonList("Регистрация прошла успешно"));
      } catch (Exception e) {
         return ResponseEntity
               .status(HttpStatus.BAD_REQUEST)
               .body(Collections.singletonList("Ошибка сервера при регистрации"));
      }
   }

   @PostMapping("/login")
   public ResponseEntity<?> authenticate(@RequestBody LoginRequestDto request) {
      try {
         AuthenticationResponseDto response = authenticationService.authenticate(request);
         return ResponseEntity.ok(response);
      } catch (BadCredentialsException e) {
         return ResponseEntity
               .status(HttpStatus.BAD_REQUEST)
               .body(Collections.singletonList("Неверный логин или пароль"));
      } catch (Exception e) {
         return ResponseEntity
               .status(HttpStatus.INTERNAL_SERVER_ERROR)
               .body(Collections.singletonList("Ошибка сервера при попытке входа"));
      }
   }
```

```
    @PostMapping("/refresh_token")
    public ResponseEntity<AuthenticationResponseDto> refreshToken(
            HttpServletRequest request,
            HttpServletResponse response) {

        return authenticationService.refreshToken(request, response);
    }

    @PostMapping("/password_change")
    public ResponseEntity<String> changePassword(@RequestBody ChangePasswordDto
passwordDto) {
        try {
            authenticationService.changePassword(passwordDto);
            return ResponseEntity.ok("Пароль успешно изменен!");
        } catch (SecurityException e) {
            return ResponseEntity
                    .status(HttpStatus.BAD_REQUEST)
                    .body(e.getMessage()); // покажет "Старый пароль неверный"
        } catch (Exception e) {
            return ResponseEntity
                    .status(HttpStatus.INTERNAL_SERVER_ERROR)
                    .body("Произошла ошибка при изменении пароля");
        }
    }


}
```

UserController

```
@RestController
@RequestMapping("/api/users")
public class UserController {

    private final UserService userService;
    @Autowired
    private RecipeService recipeService;
    @Autowired
    private RecipeRepository recipeRepository;
    @Autowired
    private UserRepository userRepository;

    @Autowired
    public UserController(UserService userService) {
        this.userService = userService;
    }
```

```
    @GetMapping("/profile")//ппросмотр своего профиля
    public ResponseEntity<UserProfileDTO> getUserById() {
            Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
            UserDetails userDetails = (UserDetails) authentication.getPrincipal();
            User currentUser = (User) userService.loadUserByUsername(userDetails.getUsername());
            UserProfileDTO profileDTO=new UserProfileDTO();
            profileDTO.setUsername(currentUser.getUsername());
            profileDTO.setEmail(currentUser.getMail());
            profileDTO.setPhone(currentUser.getPhone());
            profileDTO.setBday(currentUser.getBday());
            profileDTO.setRole(currentUser.getRole());

            return ResponseEntity.ok(profileDTO);  // Отправляем 200 OK с объектом user

    }
    //Удаление пользователя
    @DeleteMapping("/id/{id}")
    public ResponseEntity<String> deleteUserById(@PathVariable int id) {
        Optional<User> userOpt = userRepository.findById(id);
        if (userOpt.isEmpty()) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND)
                    .body("Пользователь с id " + id + " не найден");
        }

        userRepository.deleteById(id);
        return ResponseEntity.noContent().build(); // 204, если успешно удалён
    }

    @PutMapping("/{id}") // Добавляем путь для идентификатора пользователя
    @ResponseStatus(HttpStatus.OK)
    public ResponseEntity<User> update(@PathVariable Integer id, @RequestBody User user) {
        userService.updateUser(id, user); // Передаем id для обновления конкретного
пользователя
//        user.setId_user(id);
        return new ResponseEntity<>(user, HttpStatus.OK); // Возвращаем статус 200 OK
    }

    // Метод для добавления рецепта в избранное
    @PostMapping("/favourite-recipes/{recipeId}")
    public ResponseEntity<Void> addFavouriteRecipe(@PathVariable Integer recipeId) {
        // Получаем пользователя и рецепт по ID
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        UserDetails userDetails = (UserDetails) authentication.getPrincipal();
        User currentUser = (User) userService.loadUserByUsername(userDetails.getUsername());
        Recipe recipe = recipeService.getRecipeById(recipeId);

        // Добавляем рецепт в избранное пользователя
        currentUser.getFavouriteRecipes().add(recipe);

        // Сохраняем изменения в базе данных
        userRepository.save(currentUser);

        return ResponseEntity.ok().build();
    }
```

```
@GetMapping("/myFavouriteRecipe") // мои любимые рецепты
public ResponseEntity<List<RecipeAnswerDTO>> getMyFavRecipe() {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    String username = authentication.getName();

    User currentUser = (User) userService.loadUserByUsername(username);
    Set<Recipe> recipes = currentUser.getFavouriteRecipes();

    List<RecipeAnswerDTO> recipeDTOs = recipes.stream()
            .map(r -> {
                List<IngredientDTO> ingredientDTOs = r.getIngredients().stream()
                        .map(comp -> {
                            IngredientDTO dto = new IngredientDTO();
                            dto.setIngredientId(comp.getIngredient().getId_ingredient());
                            dto.setIngredientTitle(comp.getIngredient().getTitle());
                            dto.setIngredientUnit(comp.getIngredient().getUnitMeasure());
                            dto.setQuantity(comp.getQuantity());
                            return dto;
                        })
                        .collect(Collectors.toList());

                return new RecipeAnswerDTO(
                        r.getId(),
                        r.getCategory().getId_category(),
                        r.getTitle(),
                        r.getDescription(),
                        r.getManual(),
                        r.getTime(),
                        ingredientDTOs
                );
            })
            .collect(Collectors.toList());

    return ResponseEntity.ok(recipeDTOs);
}

@GetMapping("/myRecipe")
public ResponseEntity<List<RecipeAnswerDTO>> getMyRecipe() {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    String username = authentication.getName();

    User currentUser = (User) userService.loadUserByUsername(username);
    Set<Recipe> recipes = currentUser.getRecipes();

    List<RecipeAnswerDTO> recipeDTOs = recipes.stream()
            .map(r -> {
                List<IngredientDTO> ingredientDTOs = r.getIngredients().stream()
                        .map(comp -> {
                            IngredientDTO dto = new IngredientDTO();
                            dto.setIngredientId(comp.getIngredient().getId_ingredient());
                            dto.setIngredientTitle(comp.getIngredient().getTitle());
                            dto.setIngredientUnit(comp.getIngredient().getUnitMeasure());
                            dto.setQuantity(comp.getQuantity());
                            return dto;
                        })
```

```
                    .collect(Collectors.toList());

            return new RecipeAnswerDTO(
                    r.getId(),
                    r.getCategory().getId_category(),
                    r.getTitle(),
                    r.getDescription(),
                    r.getManual(),
                    r.getTime(),
                    ingredientDTOs
            );
        })
        .collect(Collectors.toList());

    return ResponseEntity.ok(recipeDTOs);
}



@DeleteMapping("/removeFavourite/{recipeId}")//удаление рцеепта из избранного
public ResponseEntity<String> removeFavouriteRecipe( @PathVariable Integer recipeId) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    UserDetails userDetails = (UserDetails) authentication.getPrincipal();
    User currentUser = (User) userService.loadUserByUsername(userDetails.getUsername());

    Recipe recipe = recipeService.getRecipeById(recipeId);

    // Проверяем, есть ли рецепт в избранных
    if (currentUser.getFavouriteRecipes().contains(recipe)) {
        // Удаляем рецепт из коллекции избранных рецептов
        currentUser.getFavouriteRecipes().remove(recipe);

        // Сохраняем пользователя (Hibernate автоматически обновит таблицу связи)
        userService.registerUser(currentUser);

        return ResponseEntity.ok("Рецепт удален из избранного");
    } else {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Рецепт не удалось
удалить из зибранного");
    }
}

@GetMapping("/isFavourite/{recipeId}")
public ResponseEntity<Boolean> isFavourite(@PathVariable int recipeId) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    UserDetails userDetails = (UserDetails) authentication.getPrincipal();
    User currentUser = (User) userService.loadUserByUsername(userDetails.getUsername());
    Recipe recipe = recipeService.getRecipeById(recipeId);

    boolean isFavourite = currentUser.getFavouriteRecipes().contains(recipe);
    return ResponseEntity.ok(isFavourite);
}

}
```

RecipeController

```
@RestController
@RequestMapping("/api/recipes")
public class RecipeController {

    @Autowired
    private RecipeRepository recipeRepository;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private CategoryRepository categoryRepository;

    @Autowired
    private IngredientRepository ingredientRepository;

    @Autowired
    private CompositionRecipeRepository compositionRecipeRepository;

    @Autowired
    private RecipeService recipeService;

    @Autowired
    private UserService userService;

    @Autowired
    private CategoryService categoryService;

    @Autowired
    private IngredientService ingredientService;

    @PostMapping("/create")
    public ResponseEntity<Recipe> createRecipe(@RequestBody RecipeDTO request) {
        // Создаем основной объект рецепта
        Recipe recipe = new Recipe();
        recipe.setTitle(request.getTitle());
        recipe.setDescription(request.getDescription());
        recipe.setManual(request.getManual());
        recipe.setTime(request.getTime());
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        UserDetails userDetails = (UserDetails) authentication.getPrincipal();

        User currentUser = (User) userService.loadUserByUsername(userDetails.getUsername());
        Category category = categoryService.getCategoryById(request.getCategoryId());

        recipe.setUser(currentUser);
        recipe.setCategory(category);

        // Создаем список компонентов для рецепта
        Set<CompositionRecipe> compositionRecipes = new HashSet<>();

        // Для каждого ингредиента создаем CompositionRecipe
```

```
        for (IngredientDTO ingredient : request.getIngredients()) {
            CompositionRecipe composition = new CompositionRecipe();

            // Получаем ингредиент из базы данных
            Ingredient ingr = ingredientService.getIngredientById(ingredient.getIngredientId());

            composition.setIngredient(ingr);
            composition.setQuantity(ingredient.getQuantity());
            composition.setRecipe(recipe);

            compositionRecipes.add(composition);
        }

        // Устанавливаем связь с компонентами
        recipe.setIngredients(compositionRecipes);

        // Сохраняем рецепт
        Recipe savedRecipe = recipeService.createRecipe(recipe);

        return ResponseEntity.created(URI.create("/recipes/" + savedRecipe.getId()))
                .body(savedRecipe);
    }


    @GetMapping("/search")
    public ResponseEntity<List<RecipeAnswerDTO>> searchRecipes(
            @RequestParam(required = false) String title,
            @RequestParam(required = false) Long categoryId,
            @RequestParam(required = false) List<String> ingredients
            )
    {

        List<RecipeAnswerDTO> recipes = recipeService.searchRecipes(title,categoryId,
ingredients);
        return ResponseEntity.ok(recipes);
    }

    @GetMapping("/getRecipe/{id}")
    public ResponseEntity<RecipeAnswerDTO> getRecipe(@PathVariable int id)
    {
       Recipe r = recipeService.getRecipeById(id);
        // Преобразуем CompositionRecipe в IngredientDTO
        List<IngredientDTO> ingredientDTOs = r.getIngredients().stream()
                .map(comp -> {
                    IngredientDTO dto = new IngredientDTO();
                    dto.setIngredientId(comp.getIngredient().getId_ingredient());
                    dto.setIngredientTitle(comp.getIngredient().getTitle());
                    dto.setIngredientUnit(comp.getIngredient().getUnitMeasure());
                    dto.setQuantity(comp.getQuantity());
                    return dto;
                })
                .collect(Collectors.toList());

        // Возвращаем DTO с ингредиентами
        RecipeAnswerDTO recipeAnswerDTO = new RecipeAnswerDTO(
```

```
                r.getId(),
                r.getCategory().getId_category(),
                r.getTitle(),
                r.getDescription(),
                r.getManual(),
                r.getTime(),
                ingredientDTOs
        );

        return ResponseEntity.ok(recipeAnswerDTO);
    }

}
```

HTML-страницы

Главная страница

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Главная</title>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.5.0/css/all.min.css">
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding-top: 60px;
            background-color: #f0f4f8;
        }
        .navbar {
            position: fixed;
            top: 0;
            width: 100%;
            background-color: #333;
            display: flex;
            justify-content: center;
            gap: 40px;
            padding: 15px 0;
            z-index: 1000;
        }
        .nav-button {
            color: white;
            text-decoration: none;
            font-size: 16px;
            display: flex;
            align-items: center;
            gap: 8px;
            transition: color 0.3s;
        }
        .nav-button:hover {
            color: #ffa500;
```

```
        }
        h1 {
            text-align: center;
            margin-top: 100px;
        }
        .recipe-card {
            background-color: #fff;
            border-radius: 16px;
            box-shadow: 0 4px 8px rgba(0,0,0,0.1);
            padding: 20px;
            width: 200px;
            text-align: center;
            transition: transform 0.2s, box-shadow 0.2s;
            cursor: pointer;
        }
        .recipe-card:hover {
            transform: translateY(-5px);
            box-shadow: 0 8px 16px rgba(0,0,0,0.15);
        }
        .recipes-container {
            display: flex;
            flex-wrap: wrap;
            gap: 20px;
            justify-content: center;
        }
    </style>
</head>
<body>


<div class="navbar">
    <a class="nav-button" href="/main"><i class="fas fa-home"></i> Главная</a>
    <a class="nav-button" href="/recipes"><i class="fas fa-book-open"></i> Рецепты</a>
    <a class="nav-button" href="/favourite_recipes"><i class="fas fa-heart"></i> Любимые
рецепты</a>
    <a class="nav-button" href="/profile"><i class="fas fa-user"></i> Профиль</a>
</div>


<h1>Это главная страница рецептов</h1>




<div class="recipes-container" id="recipesContainer">
    <!-- Здесь будут отображаться рецепты -->
</div>


<script>


    // Функция для загрузки всех рецептов
```

```
    async function fetchRecipesAll() {
        try {
            const response = await fetch(`/api/recipes/search`);
            const recipes = await response.json();
            const container = document.getElementById('recipesContainer');
            container.innerHTML = '';


            recipes.forEach(recipe => {
                const card = document.createElement('div');
                card.className = 'recipe-card';
                card.innerHTML = `
                    <h3>${recipe.title}</h3>
                    <p>${recipe.time}</p>
                `;


                // Добавляем обработчик клика для перехода на страницу рецепта
                card.addEventListener('click', () => {
                    window.location.href = `/getRecipe/${recipe.id}`;
                });


                container.appendChild(card);
            });


        } catch (error) {
            console.error("Ошибка загрузки рецептов:", error);
            document.getElementById('recipesContainer').innerHTML = '<p style="color:
red;">Ошибка загрузки данных</p>';
        }
    }


    // Загружаем рецепты при загрузке страницы
    fetchRecipesAll();
</script>
</body>
</html>
```

Страница рецептов по категории

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Рецепты по категории</title>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.5.0/css/all.min.css">
    <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;500;700&display=swap"
rel="stylesheet">
    <style>
        body {
```

```
    font-family: 'Roboto', sans-serif;
    margin: 0;
    padding-top: 60px;
    background-color: #f0f4f8;
}


.navbar {
    position: fixed;
    top: 0;
    width: 100%;
    background-color: #333;
    display: flex;
    justify-content: center;
    gap: 40px;
    padding: 15px 0;
    z-index: 1000;
}


.nav-button {
    color: white;
    text-decoration: none;
    font-size: 16px;
    display: flex;
    align-items: center;
    gap: 8px;
    transition: color 0.3s;
}


.nav-button:hover {
    color: #ffa500;
}


h1 {
    text-align: center;
    margin-bottom: 30px;
}


.filters {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin: 20px;
}


.filters input[type="text"] {
    padding: 10px;
    font-size: 14px;
    width: 80%;
    border-radius: 5px;
```

```
    border: 1px solid #ccc;
}


.filters button {
    padding: 10px 15px;
    font-size: 14px;
    border-radius: 5px;
    background-color: #4CAF50;
    color: white;
    border: none;
    cursor: pointer;
}


.filters button:hover {
    background-color: #45a049;
}


.ingredients-dropdown {
    display: none;
    position: absolute;
    background-color: white;
    border: 1px solid #ddd;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    padding: 10px;
    margin-top: 5px;
    width: 200px;
}


.ingredients-dropdown ul {
    list-style-type: none;
    padding: 0;
}


.ingredients-dropdown li {
    display: flex;
    align-items: center;
    gap: 8px;
    font-size: 14px;
    padding: 4px 0;
}


.recipe-card {
    background-color: #fff;
    border-radius: 16px;
    box-shadow: 0 4px 8px rgba(0,0,0,0.1);
    padding: 20px;
    width: 200px;
    text-align: center;
    transition: transform 0.2s, box-shadow 0.2s;
```

```
            cursor: pointer;
        }


        .recipe-card:hover {
            transform: translateY(-5px);
            box-shadow: 0 8px 16px rgba(0,0,0,0.15);
        }


        .recipes-container {
            display: flex;
            flex-wrap: wrap;
            gap: 20px;
            justify-content: center;
        }
    </style>
</head>
<body>


<!-- Навигационная панель -->
<div class="navbar">
    <a class="nav-button" href="/main"><i class="fas fa-home"></i> Главная</a>
    <a class="nav-button" href="/recipes"><i class="fas fa-book-open"></i> Рецепты</a>
    <a class="nav-button" href="/favourite_recipes"><i class="fas fa-heart"></i> Любимые
рецепты</a>
    <a class="nav-button" href="/profile"><i class="fas fa-user"></i> Профиль</a>
</div>


<!-- Фильтры -->
<div class="filters">
    <input type="text" id="titleInput" placeholder="Введите название рецепта..." />
    <div style="position: relative;">
        <button type="button" onclick="toggleIngredients()">Ингредиенты <i class="fas fa-caret-
down"></i></button>
        <div class="ingredients-dropdown" id="ingredientsDropdown">
            <ul id="ingredientList">
                <!-- Ингредиенты появятся тут -->
            </ul>
        </div>
    </div>
    <button onclick="searchByTitle()">Найти</button>
</div>


<h1 id="categoryTitle">Рецепты категории</h1>


<div class="recipes-container" id="recipesContainer">
    <!-- Рецепты отобразятся здесь -->
</div>
```

```
<script>
   const categoryId = window.location.pathname.split('/')[3];
 const urlParams = new URLSearchParams(window.location.search);
 const categoryName = urlParams.get('name');
 if (categoryName) {
   document.getElementById('categoryTitle').textContent = `Рецепты категории
"${categoryName}"`;
 }


   async function fetchRecipesByCategory() {
      try {
         const response = await fetch(`/api/recipes/search?categoryId=${categoryId}`);
         const recipes = await response.json();
         const container = document.getElementById('recipesContainer');
         container.innerHTML = '';


         recipes.forEach(recipe => {
            const card = document.createElement('div');
            card.className = 'recipe-card';
            card.innerHTML = `
               <h3>${recipe.title}</h3>
               <p>${recipe.time}</p>
            `;
            card.addEventListener('click', () => {
               window.location.href = `/getRecipe/${recipe.id}`;
            });
            container.appendChild(card);
         });
      } catch (error) {
         console.error("Ошибка загрузки рецептов:", error);
      }
   }


   function toggleIngredients() {
      const dropdown = document.getElementById('ingredientsDropdown');
      dropdown.style.display = dropdown.style.display === 'block' ? 'none' : 'block';
   }


   async function fetchIngredients() {
      try {
         const response = await fetch('/api/ingredients');
         const ingredients = await response.json();
         const list = document.getElementById('ingredientList');


         ingredients.forEach(ing => {
            const li = document.createElement('li');
            li.innerHTML = `
               <input type="checkbox" value="${ing.title}" id="ingredient-${ing.id_ingredient}">
               <label for="ingredient-${ing.id_ingredient}">${ing.title}</label>
            `;
```

```
            list.appendChild(li);
        });
    } catch (e) {
        console.error('Ошибка при загрузке ингредиентов:', e);
    }
}


async function searchByTitle() {
    const title = document.getElementById('titleInput').value;
    const selectedIngredients = Array.from(document.querySelectorAll('#ingredientList
input:checked'))
        .map(cb => cb.value);


    const params = new URLSearchParams();
    if (title) params.append('title', title);
    if (categoryId) params.append('categoryId', categoryId);
    selectedIngredients.forEach(ing => params.append('ingredients', ing));


    try {
        const response = await fetch(`/api/recipes/search?${params.toString()}`);
        const recipes = await response.json();
        const container = document.getElementById('recipesContainer');
        container.innerHTML = '';


        if (recipes.length === 0) {
            container.innerHTML = '<p style="text-align:center; color: gray;">Рецепты не
найдены</p>';
            return;
        }


        recipes.forEach(recipe => {
            const card = document.createElement('div');
            card.className = 'recipe-card';
            card.innerHTML = `
                <h3>${recipe.title}</h3>
                <p>${recipe.time}</p>
            `;
            card.addEventListener('click', () => {
                window.location.href = `/getRecipe/${recipe.id}`;
            });
            container.appendChild(card);
        });
    } catch (error) {
        console.error("Ошибка поиска рецептов:", error);
    }
}


// Инициализация
window.onload = () => {
```

```
        fetchIngredients();
        fetchRecipesByCategory();
    };
</script>
</body>
</html>
```