Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Dokumentacja końcowa

Bazy danych 2

Projekt

„Książka Kucharska Online”

Mateusz Kołacz
Mikołaj Bańkowski
Karol Bogumił
Jakub Poćwiardowski

prowadzący
Wojciech Sitek

Warszawa 2024

Spis treści

1. Skład zespołu

1. Jakub Poćwiardowski
2. Mikołaj Bańkowski
3. Karol Bogumił
4. Mateusz Kołacz

2. Opis projektu

Celem projektu było stworzenie bloga kulinarnego, zawierającego następujące funkcjonalności:

- Zakładanie konta,
- Zamieszczanie postów z przepisami kulinarnymi,
- Komentowanie postów innych użytkowników,
- Dodawanie postów do ulubionych
- Przeglądanie postów na swoim profilu.

3. Założenia realizacji projektu

Zakładana realizacja projektu przewidywała zastosowanie architektury trójwarstwowej. Założone zostało podzielenie aplikacji na następujące warstwy:

- Warstwa prezentacji danych (frontend),
- Warstwa biznesowa (backend),
- Warstwa bazodanowa.

Frontend powinien składać się z trzech widoków:

- Ekran logowania,
- Strona bloga,
- Profil użytkownika.

Każdy widok powinien znajdować się w osobnym module. Do tego niezbędne są serwisy:

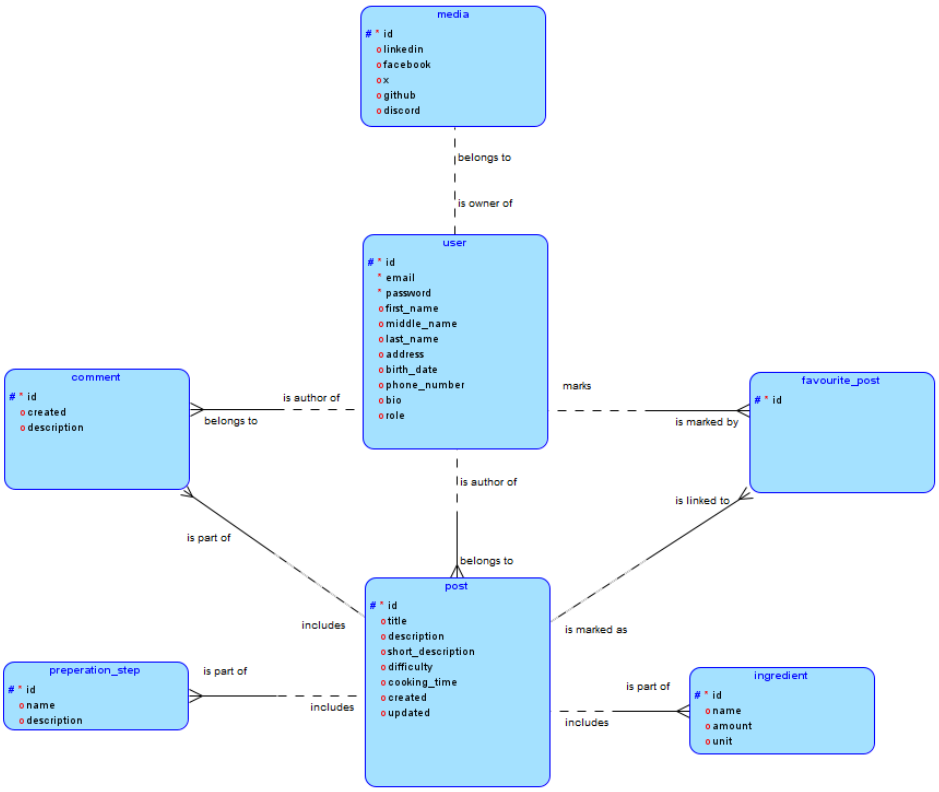
- Serwis wysyłający żądania GET do odpowiednich końcówek REST API,
- Serwis obsługujący autentykację i autoryzację (logowanie użytkownika).

Rys. 1: Struktura systemu

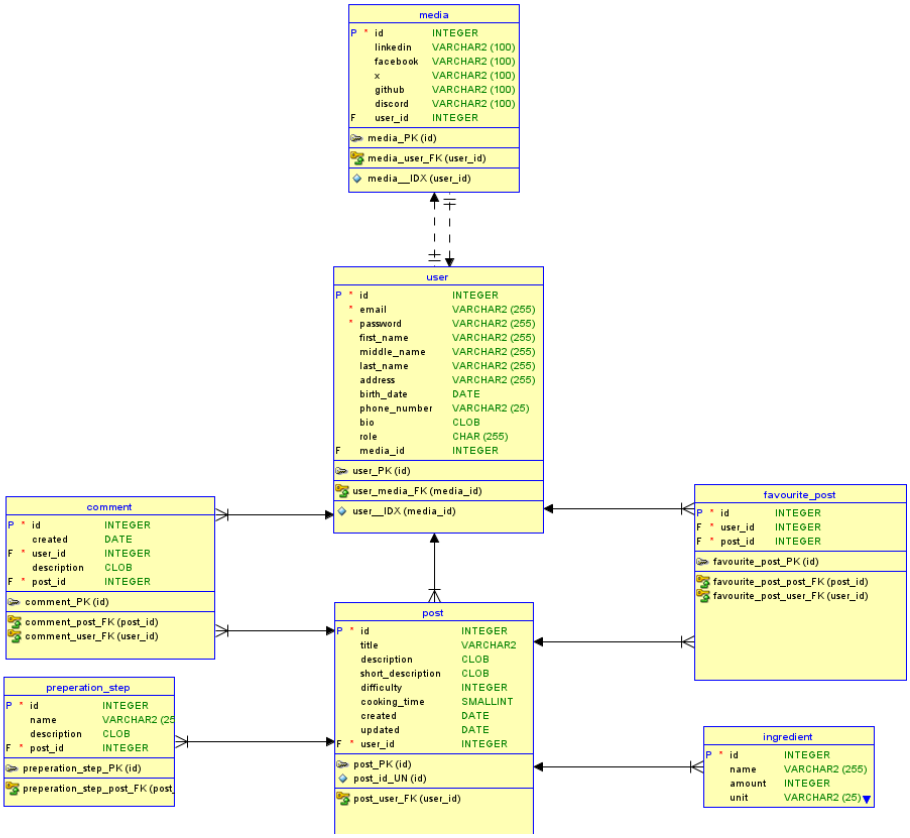
Podstawowymi modułami backendu powinny być encje, kontrolery oraz repozytoria dla każdej z tabel. Konieczne jest również stworzenie osobnego modułu bezpieczeństwa oraz logiki logowania się do aplikacji.

Baza danych powinna być reprezentowana przez następujące tabele:

- Media,
- User,
- Post,
- Favourite Post,
- Comment,
- Preparation Step,
- Ingredient.



Rys. 2: Diagram związków encji



Rys. 3: Model logiczny bazy danych

4. Technologie programistyczne

Baza danych: MySQL

Backend: Java 17, Spring Boot (Hibernate, JPA, REST Api), Security – JWT, Logging - SLF4J

Frontend: Angular (Typescript, Html, CSS & SASS)

5. Wymagania funkcjonalne

Zakładana wstępnie funkcjonalność przewidywała możliwość:

- Przeglądania wpisów na blogu,
- Udostępniania przepisów, postów,
- Dodawania postów do ulubionych,
- Przeglądania ulubionych postów,
- Zalogowania się jako czytelnik,
- Komentowania wpisów jako czytelnik,
- Przeglądania profilu użytkownika,
- Edycji profilu użytkownika.

6. Realizacja projektu

W niniejszym rozdziale podsumowane zostaną prace projektowe i zaprezentowane efekty działania zespołu w stosunku do przyjętych założeń.

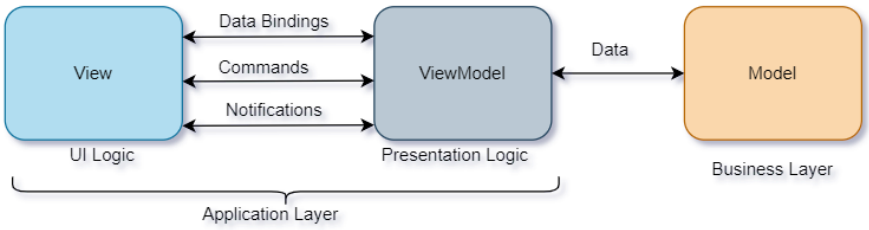
6.1. Zrealizowane funkcjonalności

Funkcjonalności które udało się zrealizować:

- **Przeglądanie wpisów na blogu,**
- **Udostępnianie przepisów, postów,**
- **Dodawanie postów do ulubionych,**
- **Dodawanie nowych postów,**
- **Przeglądanie ulubionych postów,**
- **Logowanie/rejestracja jako czytelnik,**
- **Komentowanie wpisów jako czytelnik,**
- **Przeglądanie profilu użytkownika,**
- **Edycja profilu użytkownika.**

6.2 Warstwa wizualna

Warstwa wizualna została stworzona za pomocą Angulara w oparciu o architekturę MVVM. Jest to aplikacja internetowa typu SPA (Single-Page-Aplication), składająca się z trzech widoków: ekran logowania/rejestracji, strona bloga, profil użytkownika.



Rys. 3: Architektura MVVM

Model reprezentuje dane i logikę biznesową aplikacji. W Angularze modele często są zwykłymi TypeScript-owymi klasami lub interfejsami.

Widok (*View*) odpowiada za prezentację danych i interakcję użytkownika. W Angularze widoki są reprezentowane przez komponenty.

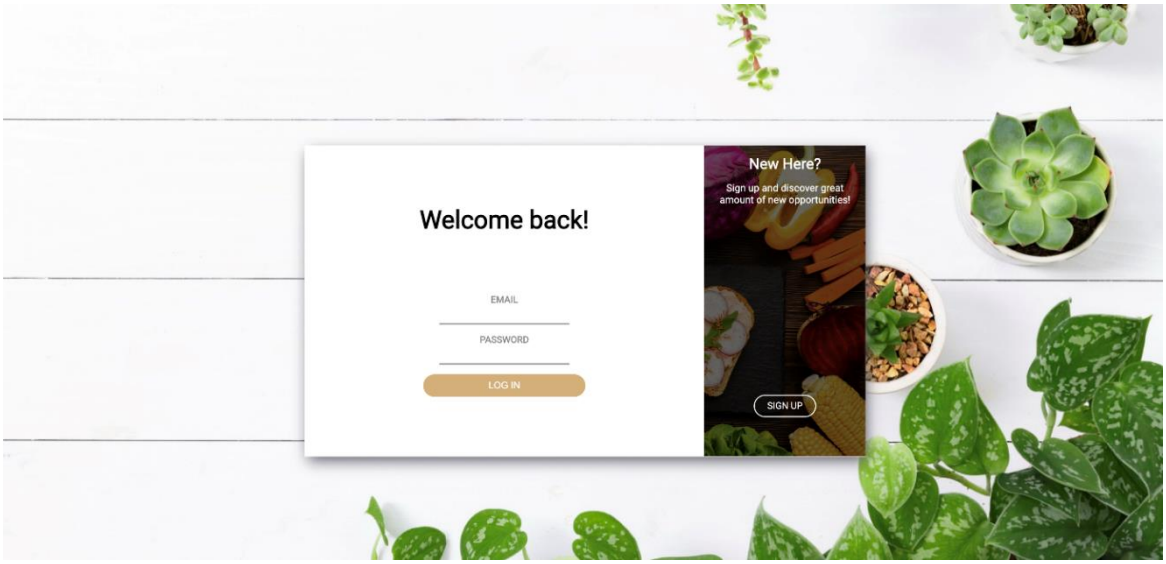
W Angularze obiekt **ViewModel** nie istnieje jako oddzielna instancja. Zamiast tego, logika i dane obiektu ViewModel są często zarządzane przez komponenty Angulara.

Dwukierunkowe wiązanie danych (*Two-Way Data Binding*) to jedna z kluczowych cech Angulara, która ułatwia synchronizację danych pomiędzy modelem a widokiem oraz między widokiem a modelem. Zmiany w modelu automatycznie odzwierciedlają się w widoku, a zmiany w widoku są odzwierciedlane w modelu.

Widok – *ekran logowania/rejestracji*

Funkcjonalności:

- Zalogowanie się/ zarejestrowanie się jako czytelnik

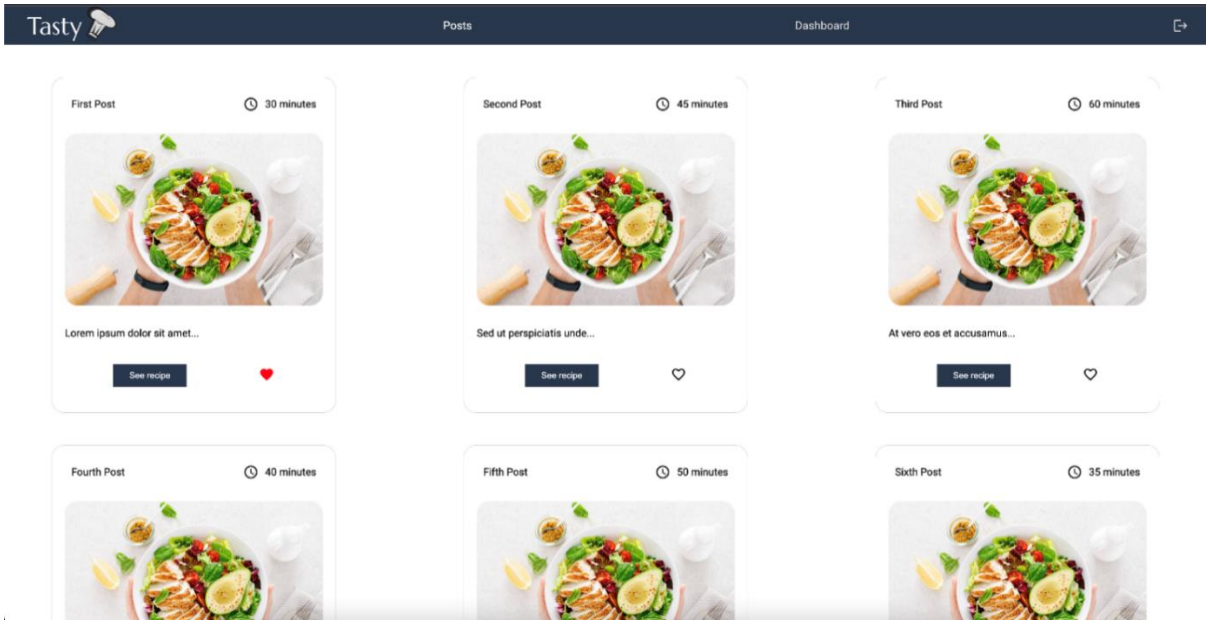


Rys. 4: Ekran logowania

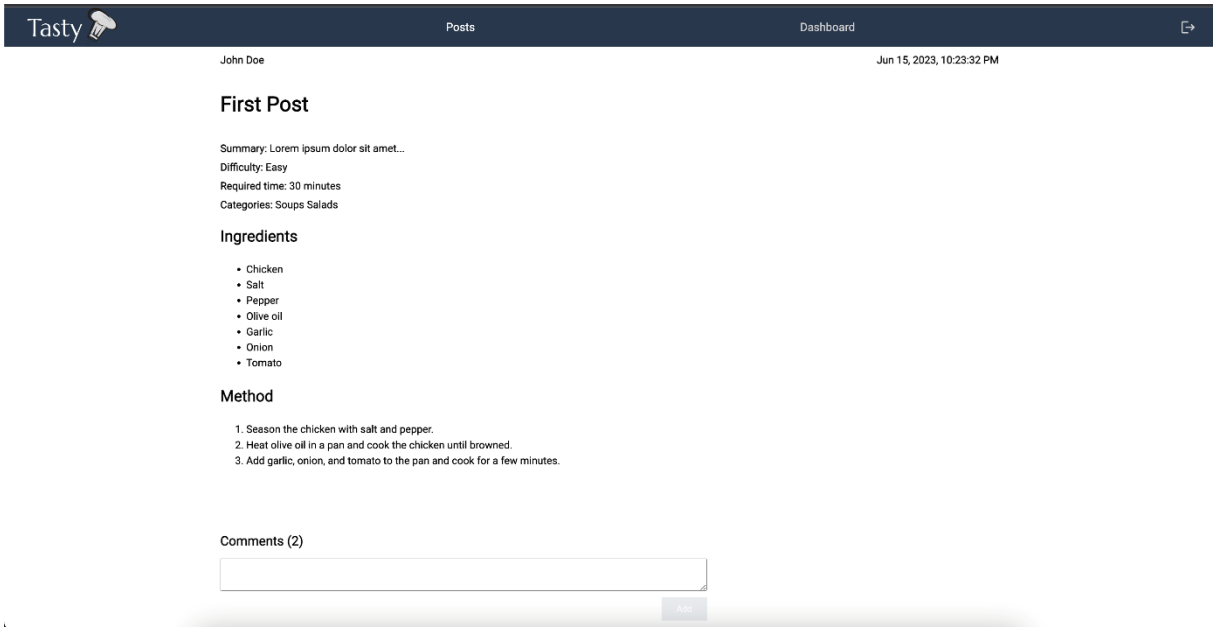
Widok – *strona bloga*

Funkcjonalności:

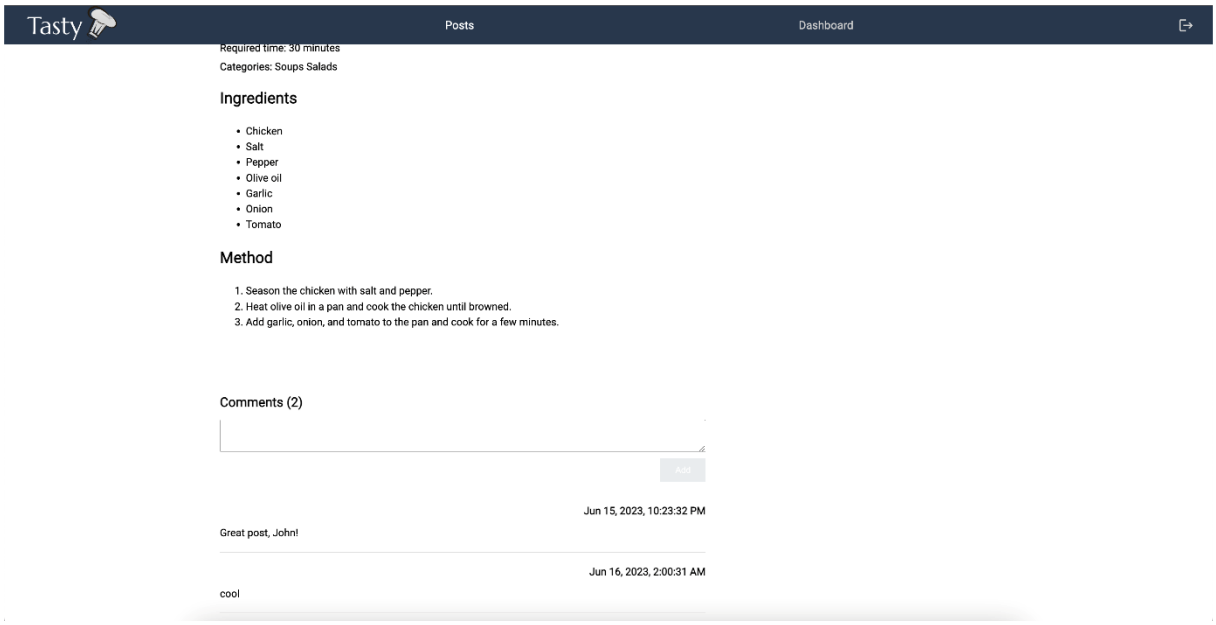
- Przeglądanie wpisów na blogu,
- Dodawanie postów do ulubionych,
- Komentowanie wpisów jako czytelnik.



Rys. 5: Strona główna bloga



Rys. 6: Widok po wejściu w przepis

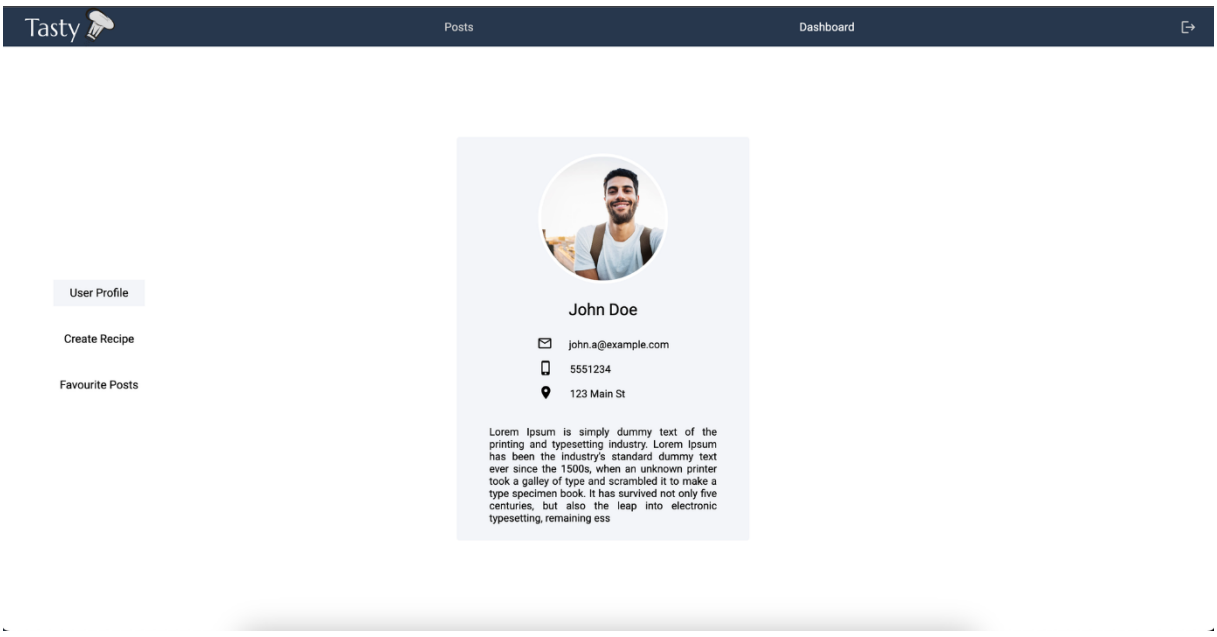


Rys. 7: Dalsza część widoku po wejściu w przepis

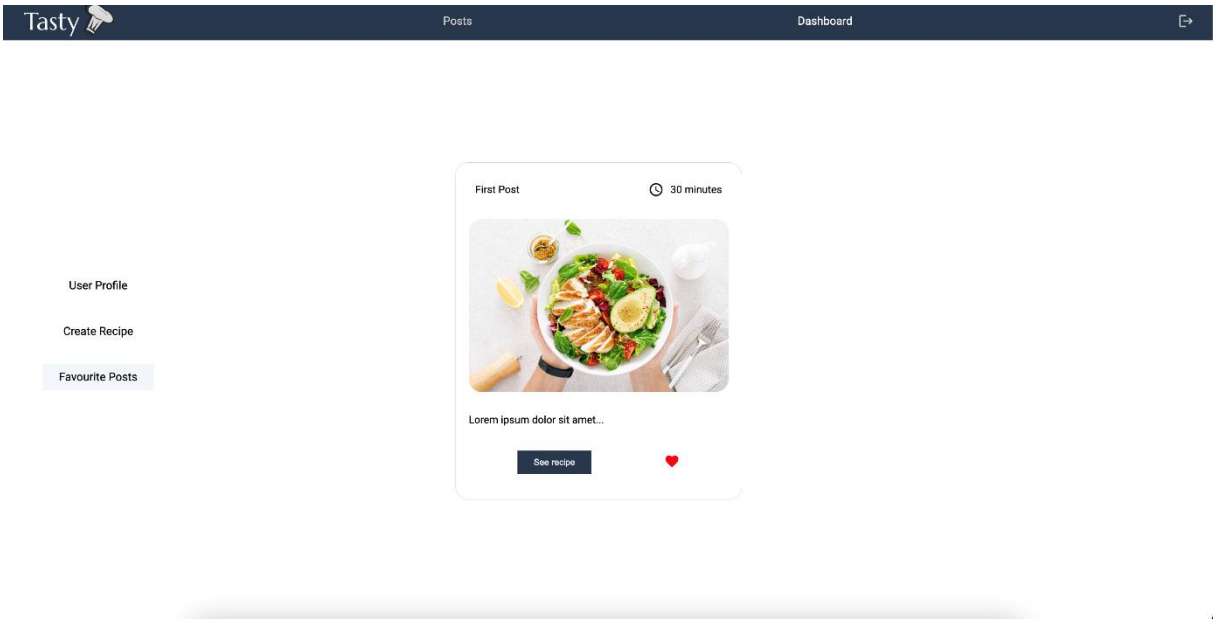
Widok – *profil użytkownika*

Funkcjonalności:

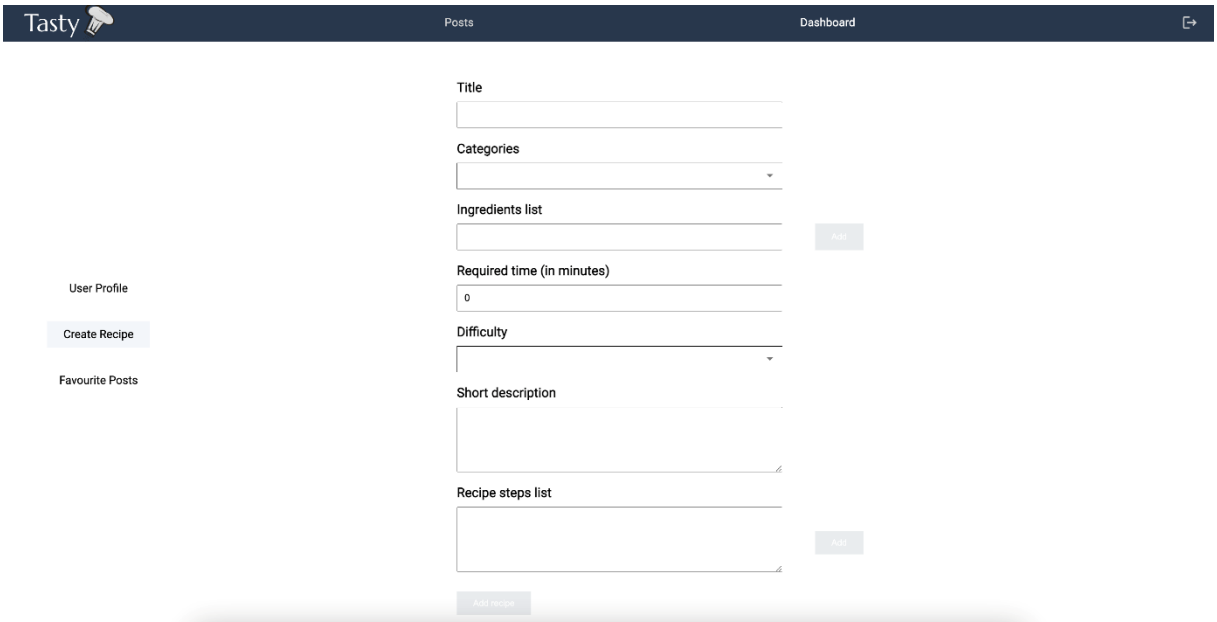
- Przeglądanie profilu użytkownika,
- Przeglądanie ulubionych postów,
- Dodawanie nowych postów.



Rys. 8: Profil użytkownika



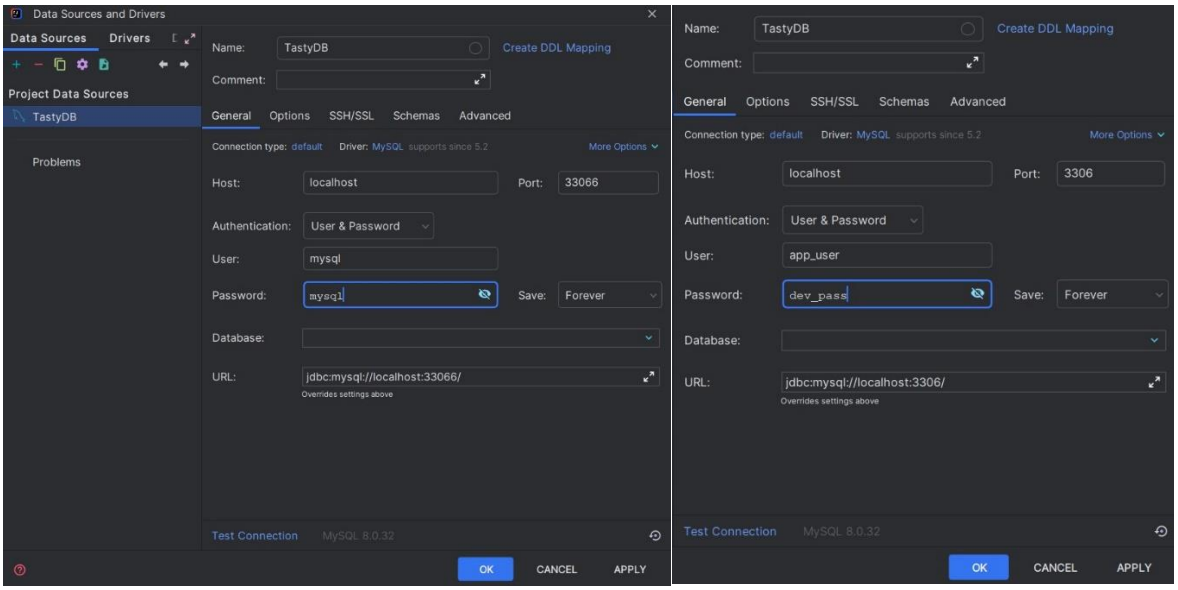
Rys. 9: Widok przeglądania ulubionych postów



Rys. 10: Interfejs tworzenia nowego postu

6.2 Baza danych

Rys. 11 przedstawia konfigurację bazy danych MySQL.



Rys. 11: Konfiguracja bazy danych MySQL

Zostały przygotowane dwa skrypty:

1. Skrypt tworzący bazę danych *schema.sql,*
2. Skrypt z danymi testowymi *data.sql.*

Na rys. 12, 13, 14, 15 przedstawione zostały zrzuty ekranu skryptu tworzącego bazę danych schema.sql.

```sql
CREATE TABLE IF NOT EXISTS `media`
(
    `id`        INT UNSIGNED NOT NULL AUTO_INCREMENT,
    `linked_in` VARCHAR(100),
    `facebook`  VARCHAR(100),
    `twitter`   VARCHAR(100),
    `github`    VARCHAR(100),
    `discord`   VARCHAR(100),
    PRIMARY KEY (`id`)
);


CREATE TABLE IF NOT EXISTS `users`
(
    `id`           INT UNSIGNED PRIMARY KEY NOT NULL AUTO_INCREMENT,
    `first_name`   VARCHAR(255)             NOT NULL,
    `middle_name`  VARCHAR(255),
    `last_name`    VARCHAR(255)             NOT NULL,
    `address`      VARCHAR(255)             NOT NULL,
    `birth_date`   DATE                     NOT NULL,
    `phone_number` VARCHAR(20)              NOT NULL,
    `email`        VARCHAR(255)             NOT NULL UNIQUE,
    `media_id`     INT UNSIGNED,
    `bio`          TEXT,
    `password`     VARCHAR(255)             NOT NULL,
    `role`         ENUM ('USER', 'ADMIN')   NOT NULL,
    FOREIGN KEY (`media_id`) REFERENCES `media` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Rys. 12: Skrypt tworzący bazę danych schema.sql

```sql
ALTER TABLE users
    MODIFY COLUMN password VARCHAR(255) NOT NULL;

CREATE TABLE IF NOT EXISTS `posts`
(
    `id`                INT UNSIGNED NOT NULL AUTO_INCREMENT,
    `title`             VARCHAR(255) NOT NULL,
    `description`       TEXT         NOT NULL,
    `short_description` TEXT         NOT NULL,
    `ingredients`       VARCHAR(255),
    `difficulty`        INT          NOT NULL,
    `cooking_time`      SMALLINT,
    `user_id`           INT UNSIGNED NOT NULL,
    `created`           DATETIME     NOT NULL DEFAULT CURRENT_TIMESTAMP,
    `updated`           DATETIME     NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    PRIMARY KEY (`id`),
    FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS `ingredients`
(
    `id`      INT PRIMARY KEY AUTO_INCREMENT,
    `name`    VARCHAR(255),
    `post_id` INT UNSIGNED NOT NULL,
    FOREIGN KEY (`post_id`) REFERENCES `posts` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Rys. 13: Skrypt tworzący bazę danych schema.sql

```sql
CREATE TABLE IF NOT EXISTS `preparation_steps`
(
    `id`      INT PRIMARY KEY AUTO_INCREMENT,
    `step`    VARCHAR(255),
    `post_id` INT UNSIGNED NOT NULL,
    FOREIGN KEY (`post_id`) REFERENCES `posts` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS `product_categories`
(
    `id`      INT PRIMARY KEY AUTO_INCREMENT,
    `step`    VARCHAR(255),
    `post_id` INT UNSIGNED NOT NULL,
    FOREIGN KEY (`post_id`) REFERENCES `posts` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE IF NOT EXISTS `favourite_posts`
(
    `id`      INT PRIMARY KEY AUTO_INCREMENT,
    `user_id` INT UNSIGNED NOT NULL,
    `post_id` INT UNSIGNED NOT NULL,
    FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (`post_id`) REFERENCES `posts` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Rys. 14: Skrypt tworzący bazę danych schema.sql

```sql
CREATE TABLE IF NOT EXISTS `comments`
(
    `id`      INT UNSIGNED NOT NULL AUTO_INCREMENT,
    `content` TEXT         NOT NULL,
    `user_id` INT UNSIGNED NOT NULL,
    `post_id` INT UNSIGNED NOT NULL,
    `created` DATETIME     NOT NULL DEFAULT CURRENT_TIMESTAMP,
    `updated` DATETIME     NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    PRIMARY KEY (`id`),
    FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (`post_id`) REFERENCES `posts` (`id`) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Rys. 15: Skrypt tworzący bazę danych schema.sql

Rys. 16 – 21 prezentują zrzuty ekranu ze skryptu zapewniającego dane testowe data.sql.

```sql
INSERT INTO media (linked_in, facebook, twitter, github, discord)
VALUES ('https://www.linkedin.com/in/user1', 'https://www.facebook.com/user1', 'https://twitter.com/user1',
        'https://github.com/user1', 'https://discord.com/user1'),
       ('https://www.linkedin.com/in/user2', 'https://www.facebook.com/user2', 'https://twitter.com/user2',
        'https://github.com/user2', 'https://discord.com/user2'),
       ('https://www.linkedin.com/in/user3', 'https://www.facebook.com/user3', 'https://twitter.com/user3',
        'https://github.com/user3', 'https://discord.com/user3'),
       ('https://www.linkedin.com/in/user4', 'https://www.facebook.com/user4', 'https://twitter.com/user4',
        'https://github.com/user4', 'https://discord.com/user4'),
       ('https://www.linkedin.com/in/user5', 'https://www.facebook.com/user5', 'https://twitter.com/user5',
        'https://github.com/user5', 'https://discord.com/user5');
```

Rys. 16: Skrypt z danymi testowymi data.sql

```sql
INSERT INTO users (first_name, middle_name, last_name, address, birth_date, phone_number, email, media_id, bio,
                   password, role)
VALUES ('John', 'A', 'Doe', '123 Main St', '1990-01-01', '5551234', 'john.a@example.com', 1,
        'Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry''s standard dummy text ever
        '$2a$10$t.t.oER7nqv1ANC5ZovqkeIgJ7tRdJhOm/aYPJdHNo.9qSAuulPXe', 'USER'),
       ('Jane', NULL, 'Austin', '456 Maple Ave', '1995-02-15', '5555678', 'jane@example.com', 2,
        'Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry''s standard dummy text ever
        '$2a$10$9/qNOo1zreqXX60MIX9jLeEzarQiQKR.SngFO1Nj50apFmCuSjc36', 'USER'),
       ('Bob', 'B', 'Marley', '789 Oak St', '1985-05-10', '5559101', 'bob.b@example.com', 3,
        'Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry''s standard dummy text ever
        '$2a$10$0wzzCw4dxCWaxbzQ5Kl1u.XKOBIXImBQiFvDEEXrSzjNwniFReYfe', 'USER'),
       ('Sarah', 'L', 'Maas', '321 Elm St', '1992-04-18', '5555555', 'sarah.l@example.com', 4,
        'Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry''s standard dummy text ever
        '$2a$10$y4WElh4eKhaWk8lN7P6qku1LbVCiiqTKyeUh0LXMRfRSrkg11778u', 'USER'),
       ('Tom', 'J', 'Kruger', '567 Birch Ave', '1988-11-30', '5551234', 'tom.j@example.com', null,
        'Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry''s standard dummy text ever
        '$2a$10$y4WElh4eKhaWk8lN7P6qku1LbVCiiqTKyeUh0LXMRfRSrkg11778u', 'USER'),
       ('Emily', NULL, 'Jackman', '789 Oak St', '1995-07-12', '5555678', 'emily@example.com', 5,
        'Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry''s standard dummy text ever
        '$2a$10$1BAudE5dhFT3D5lXH28AW.JNpNS5DkeqoSUUAmb8ieByMaX8Qz.ay', 'USER');
```

Rys. 17: Skrypt z danymi testowymi data.sql

```sql
INSERT INTO posts (title, short_description, difficulty, cooking_time, user_id, created, updated)
VALUES ('First Post', 'Lorem ipsum dolor sit amet...', 1, 30, 1, NOW(), NOW()),
       ('Second Post', 'Sed ut perspiciatis unde...', 2, 45, 1, NOW(), NOW()),
       ('Third Post', 'At vero eos et accusamus...', 3, 60, 2, NOW(), NOW()),
       ('Fourth Post', 'Duis aute irure dolor in repreh...', 2, 40, 2, NOW(), NOW()),
       ('Fifth Post', 'Excepteur sint occaecat cupidatat...', 3, 50, 3, NOW(), NOW()),
       ('Sixth Post', 'Ut enim ad minim veniam...', 1, 35, 1, NOW(), NOW());


INSERT INTO comments (content, user_id, post_id, created, updated)
VALUES ('Great post, John!', 2, 1, NOW(), NOW()),
       ('I really enjoyed this article, thanks for sharing!', 3, 2, NOW(), NOW()),
       ('I totally agree, Bob!', 1, 3, NOW(), NOW()),
       ('Great post, Tom!', 1, 4, NOW(), NOW()),
       ('I really enjoyed this article, thanks for sharing!', 2, 5, NOW(), NOW()),
       ('I totally agree, Emily!', 3, 6, NOW(), NOW());
```

Rys. 18: Skrypt z danymi testowymi data.sql

```sql
INSERT INTO `ingredients` (`name`, `post_id`)
VALUES ('Chicken', 1),
       ('Salt', 1),
       ('Pepper', 1),
       ('Olive oil', 1),
       ('Garlic', 1),
       ('Onion', 1),
       ('Tomato', 1),
       ('Pasta', 2),
       ('Parmesan cheese', 2),
       ('Butter', 2),
       ('Basil', 2),
       ('Garlic powder', 2),
       ('Ground beef', 3),
       ('Bell pepper', 3),
       ('Onion', 3),
       ('Chili powder', 3),
       ('Cumin', 3);

INSERT INTO `preparation_steps` (`step`, `post_id`)
VALUES ('Season the chicken with salt and pepper.', 1),
       ('Heat olive oil in a pan and cook the chicken until browned.', 1),
       ('Add garlic, onion, and tomato to the pan and cook for a few minutes.', 1),
       ('Boil water in a pot and cook the pasta until al dente.', 2),
       ('Drain the pasta and mix in butter, Parmesan cheese, basil, and garlic powder.', 2),
       ('Heat a skillet and cook ground beef, bell pepper, and onion until browned.', 3),
       ('Add chili powder and cumin to the skillet and cook for a few more minutes.', 3);
```

Rys. 19: Skrypt z danymi testowymi data.sql

```sql
INSERT INTO `preparation_steps` (`step`, `post_id`)
VALUES ('Season the chicken with salt and pepper.', 1),
       ('Heat olive oil in a pan and cook the chicken until browned.', 1),
       ('Add garlic, onion, and tomato to the pan and cook for a few minutes.', 1),
       ('Boil water in a pot and cook the pasta until al dente.', 2),
       ('Drain the pasta and mix in butter, Parmesan cheese, basil, and garlic powder.', 2),
       ('Heat a skillet and cook ground beef, bell pepper, and onion until browned.', 3),
       ('Add chili powder and cumin to the skillet and cook for a few more minutes.', 3);


INSERT INTO product_categories (name, post_id)
VALUES ('Soups', 1),
       ('Salads', 1),
       ('Sandwiches/Wraps', 2),
       ('By Meat', 2),
       ('Pasta', 3),
       ('Soups', 4),
       ('Salads', 5),
       ('Sandwiches/Wraps', 4),
       ('By Meat', 6),
       ('Pasta', 6);
```

Rys. 20: Skrypt z danymi testowymi data.sql

```
95
96      INSERT INTO favourite_posts (user_id, post_id)
97      VALUES (1, 1),
98             (1, 2),
99             (2, 3),
100            (3, 1),
101            (4, 2),
102            (4, 3),
103            (4, 4);
104
```

Rys. 21: Skrypt z danymi testowymi data.sql

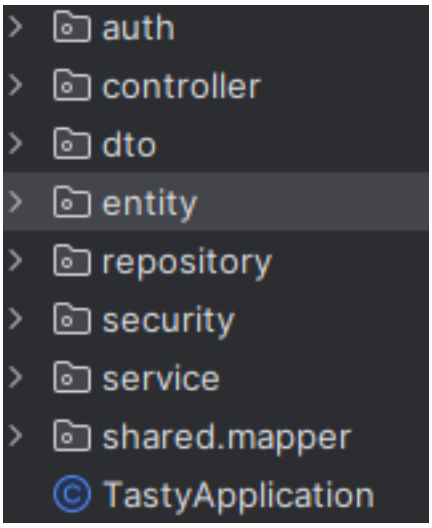
6.3 Backend – część serwerowa

Backend aplikacji stanowi projekt stworzony w języku Java w popularnym frameworku Spring Boot. Jego głównym zadaniem jest obsługa zapytań przesyłanych przez użytkownika korzystającego z warstwy wizualnej w przeglądarce i komunikacja z bazą danych.

Kluczową odpowiedzialnością tego serwisu jest uwierzytelnianie oraz autoryzacja użytkowników. Wykorzystano tutaj zestaw narzędzi Spring Security i zaimplementowano tokeny JSON Web Token do uwierzytelniania zapytań. Dzięki przenośności oraz standardowości tej procedury, jest to stosunkowo proste do wykonania i ogranicza się do poprawnej konfiguracji kilku klas.

Komunikacja z bazą wykorzystuje Java DataBase Connectivity (JDBC) do połączenia oraz Java Persistence API (JPA) do mapowania obiektowo-relacyjnego. Dzięki temu, w kodzie można stosować czytelny i wygodny model obiektowy, który będzie skutecznie wprowadzany do relacyjnej bazy danych.

Struktura projektu serwera zorganizowana jest w klasyczny sposób, często spotykany w projektach w oparciu o Spring Boot. Wykorzystuje komunikację pomiędzy klasami Controller, DTO (data transfer object) i Entity. Dzięki temu zrozumiały jest przepływ informacji i odpowiedzialność za każdą ingerencję i manipulację danymi.



Rys. 22: Struktura projektu

```java
@Data
@Entity
@Table(name = "posts")
@NoArgsConstructor
public class Post {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NonNull
    @Column(name = "title", nullable = false)
    private String title;

    @OneToMany(mappedBy = "post", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<PreparationStep> preparationSteps = new ArrayList<>();

    @OneToMany(mappedBy = "post", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Ingredient> ingredients = new ArrayList<>();

    @OneToMany(mappedBy = "post", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Comment> comments = new ArrayList<>();

    @OneToMany(mappedBy = "post", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<ProductCategory> categories = new ArrayList<>();
```

Rys. 23: Fragment definicji klasy Post odpowiadającej encji Post w bazie danych