Przepiśnik

Aplikacja webowa umożliwiająca wyszukiwanie, organizowanie oraz dzielenie się przepisami kulinarnymi.

Agnieszka Gutowska, Filip Misztal, Grzegorz Molak, Zuzanna Popławska, Anna Szymańska, Dorota Wlazło

Podział pracy

Rola	Technologie	Członkowie odpowiedzialni
Kierownik projektu, programista front-end	JavaScript	Anna Szymańska
Programista front-end	HTML, CSS	Agnieszka Gutowska
Programista front-end	JavaScript	Dorota Wlazło
Programista back-end	Java + Spring	Filip Misztal
Programista security	JWT token, cookies, Java	Grzegorz Molak
Programista back-end i bazy danych	MongoDB, Java	Zuzanna Popławska

Architektura systemu

Opis głównych wymagań funkcjonalnych i niefunkcjonalnych aplikacji.

Wymagania Funkcjonalne:

- Tworzenie konta użytkownika.
- Intuicyjne dodawanie własnych przepisów dzięki prostemu w obsłudze formularzowi.
- Każde konto ma przypisany prywatny przepiśnik z przepisami prywatnymi oraz przepisami innych użytkowników dodanymi do ulubionych.
- Możliwość dzielenia swojego przepiśnika na foldery, dodawnie przepisów do folderów.
- Wyszukiwanie przepisów w swoim przepiśniku z użyciem słów kluczowych oraz tagów (vege, ostre, itp.) .
- Udostępnianie przepisów innym użytkownikom.
- Wyszukiwanie przepisów wśród udostępnionych przez innych użytkowników z użyciem słów kluczowych oraz tagów (vege, ostre, śniadanie, itp.).
- Niezalogowani użytkownicy mogą przeglądać przepisy udostępnione przez użytkowników.
- Ciemny i jasny motyw.
- Możliwość dodawania zdjęć pod przepisami.

Wymagania niefunkcjonalne

- Bezpieczeństwo – Aplikacja jest zabezpieczona przed dostępem niezalogowanych użytkowników do poszczególnych funkcjonalności aplikacji. Hasła przechowywane w bazie danych są zaszyfrowane.
- Łatwość obsługi – aplikacja posiada prosty, nieskomplikowany i przyjemny interfejs. Użycie dowolnej funkcjonalności nie wymaga od użytkownika żadnej wiedzy o architekturze aplikacji. Dla komfortu użytkowania istnieje możliwość wyboru między jasnym a ciemnym motywem.

Architektura ogólna:

Użyta została trójwarstwowa architektura klient-serwer, w której klient przy użyciu przeglądarki wysyła dzięki JavaScript żądania do serwera pracującego w Javie, która z kolei komunikuje się z bazą danych MongoDB, po czym wysyła odpowiedzi.

Obsługi logiki biznesowej dokonujemy w języku Java z wykorzystaniem frameworka Spring. Do prezentacji danych oraz reagowania na zdarzenia użytkownika w przeglądarce używamy HTML, CSS oraz JavaScript.

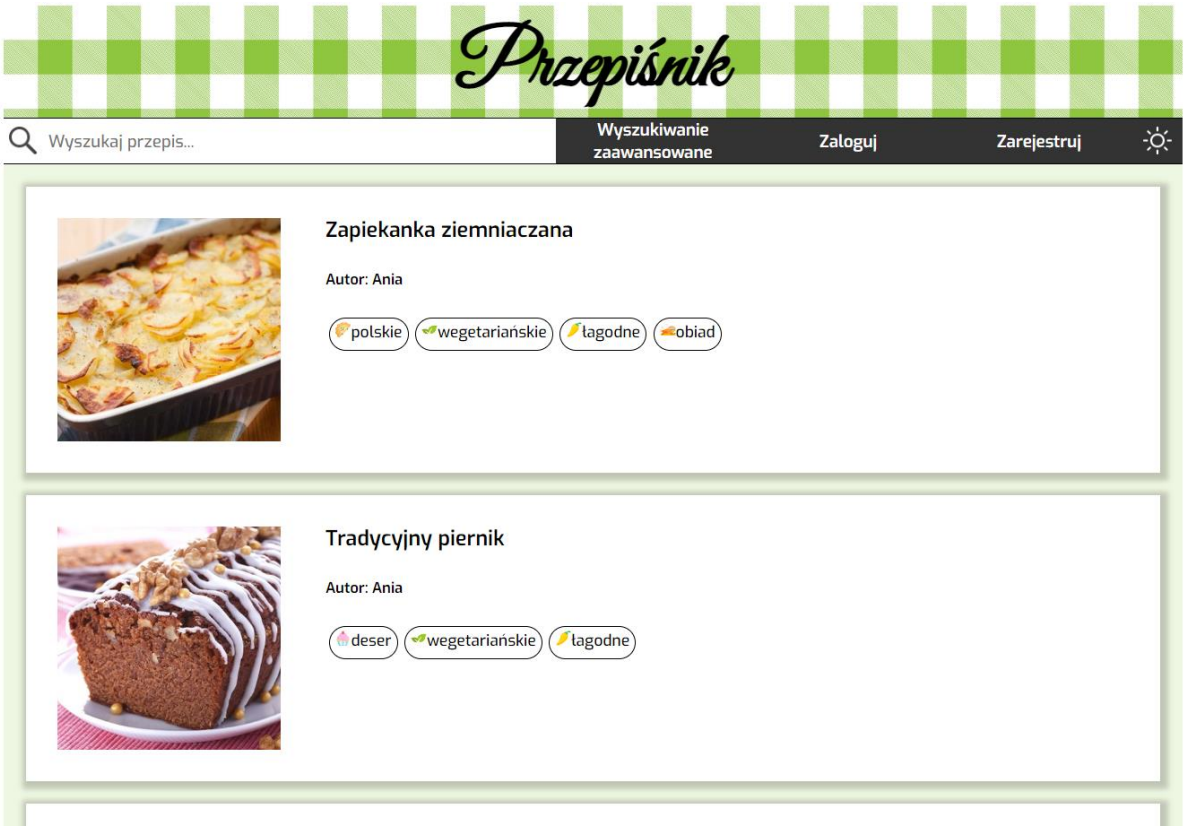
Warstwa prezentacji:

Warstwę prezentacji stanowi strona internetowa, łącząca się z uruchomioną aplikacją. Serwis dostarcza atrakcyjną oprawę graficzną umożliwiającą proste wykorzystanie wszystkich możliwości systemu. Warstwa kliencka została stworzona bez użycia żadnych bibliotek, jedynie z użyciem technologii HTML i CSS oraz JavaScript, który reaguje na zdarzenia użytkownika takie jak kliknięcie przycisku, wysyła żądania do serwera, odbiera je, oraz manipuluje dokumentem HTML, aby zaprezentować odebrane dane.

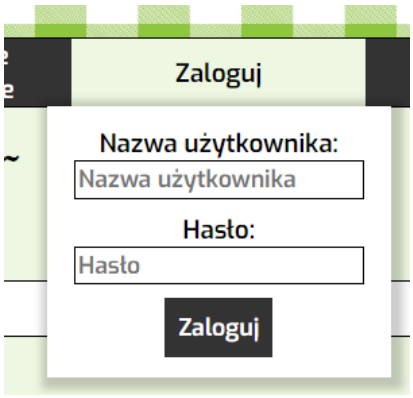
Serwer został zrealizowany jako system REST, więc możliwe jest uzyskanie dostępu do publicznych zasobów (a więc na przykład wyszukanie przepisów) przez bezpośrednie wysłanie zapytania HTTP do serwera (metoda GET). W odpowiedzi zwrócony zostanie dokument JSON.

Widoki w przeglądarce

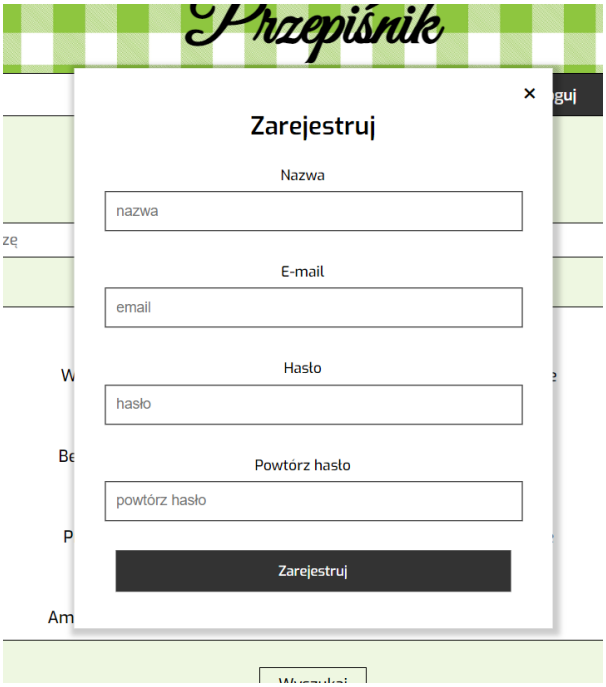
Po otworzeniu strony w przeglądarce użytkownikowi przedstawiony zostaje widok widoczny na rysunku 1. Umożliwia wejście w widoczny (publiczny przepis), wyszukanie przepisu, zalogowanie (rysunek 2) i rejestrację (rysunek 3).



Rysunek 1 Strona początkowa przepiśnika



Rysunek 2 Formularz logowania



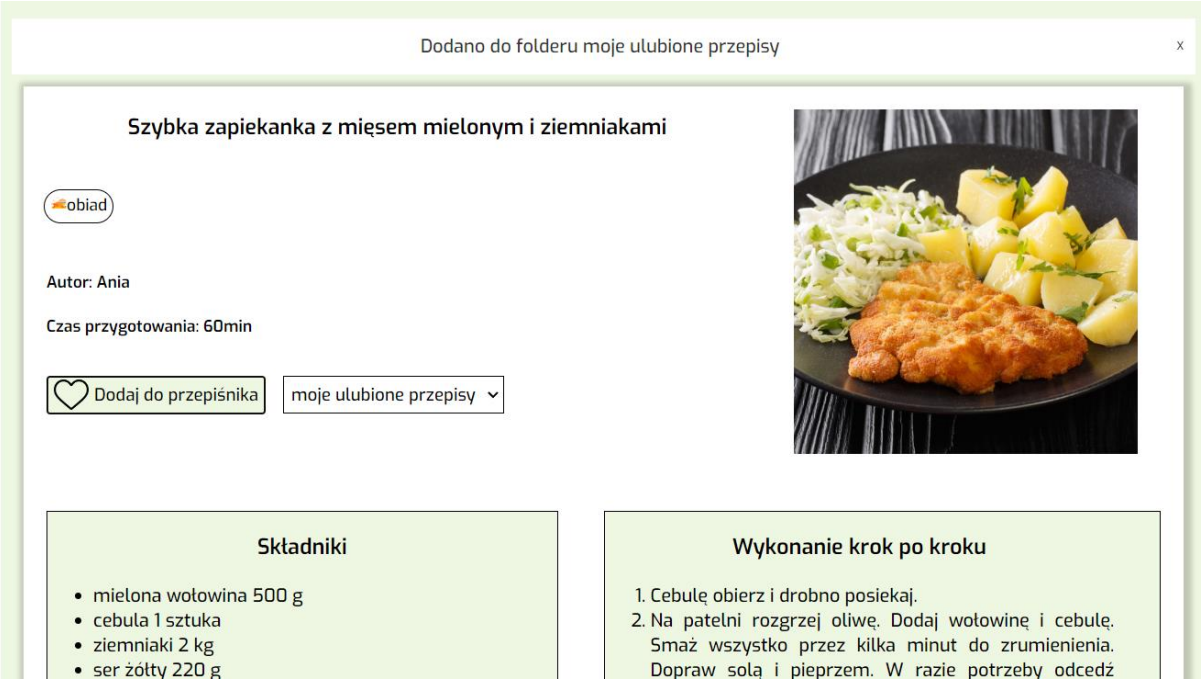
Rysunek 3 Formularz rejestracji

Po zalogowaniu się zostajemy przekierowani na stronę główną (rysunek 4), różniącą się od poprzedniej jedynie innym menu.

Rysunek 4 Strona główna, po zalogowaniu się widoczne jest inne menu

Widok przepisu jest różny dla osoby zalogowanej i niezalogowanej. Osoba zalogowana ma dodatkowo możliwość przeglądanego przepisu do któregoś ze swoich folderów jak widać na rysunku 6.

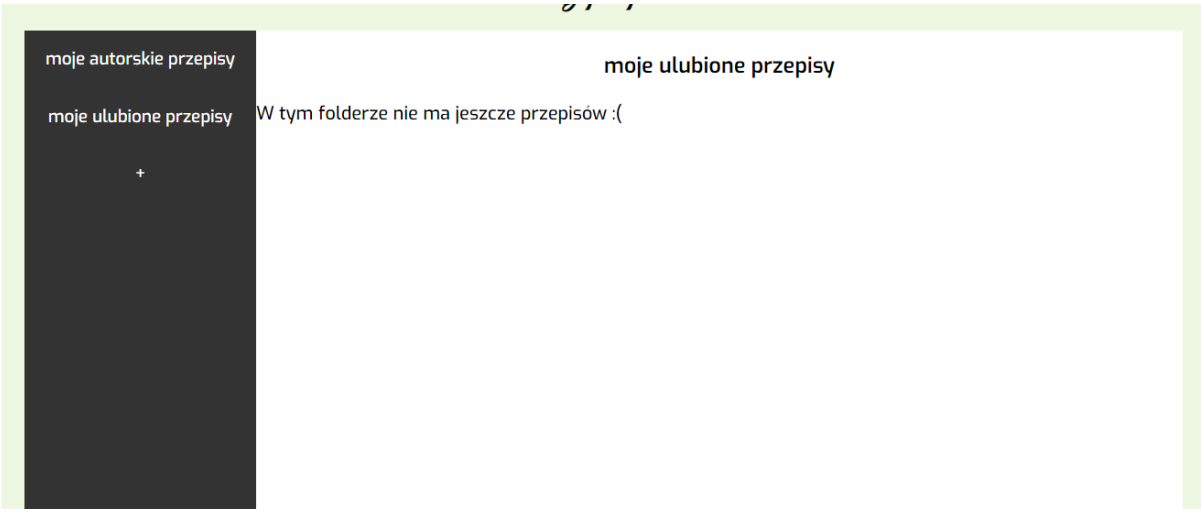
Rysunek 5 Widok przepisu dla osoby niezalogowanej



Rysunek 6 Widok przepisu dla osoby zalogowanej, po dodaniu przepisu do folderu

Widok „mojego przepiśnika” pokazuje zapisane przepisy, z możliwością wyboru folderu, jeżeli folder jest pusty – użytkownik zostaje o tym poinformowany

Rysunek 7 Folder z autorskimi przepisami

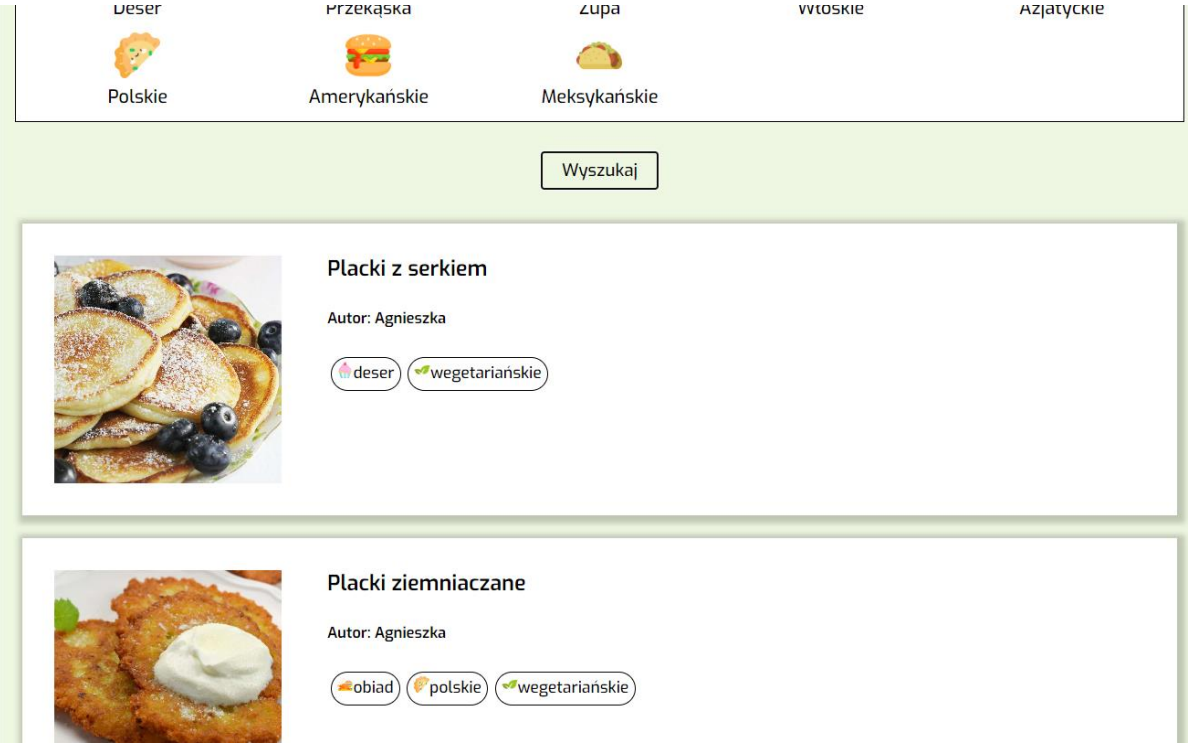


Rysunek 8 Pusty folder

Wyszukiwanie zaawansowane umożliwia wyszukiwanie przepisu przez jego nazwę oraz odpowiadające tagi(rysunek 9).

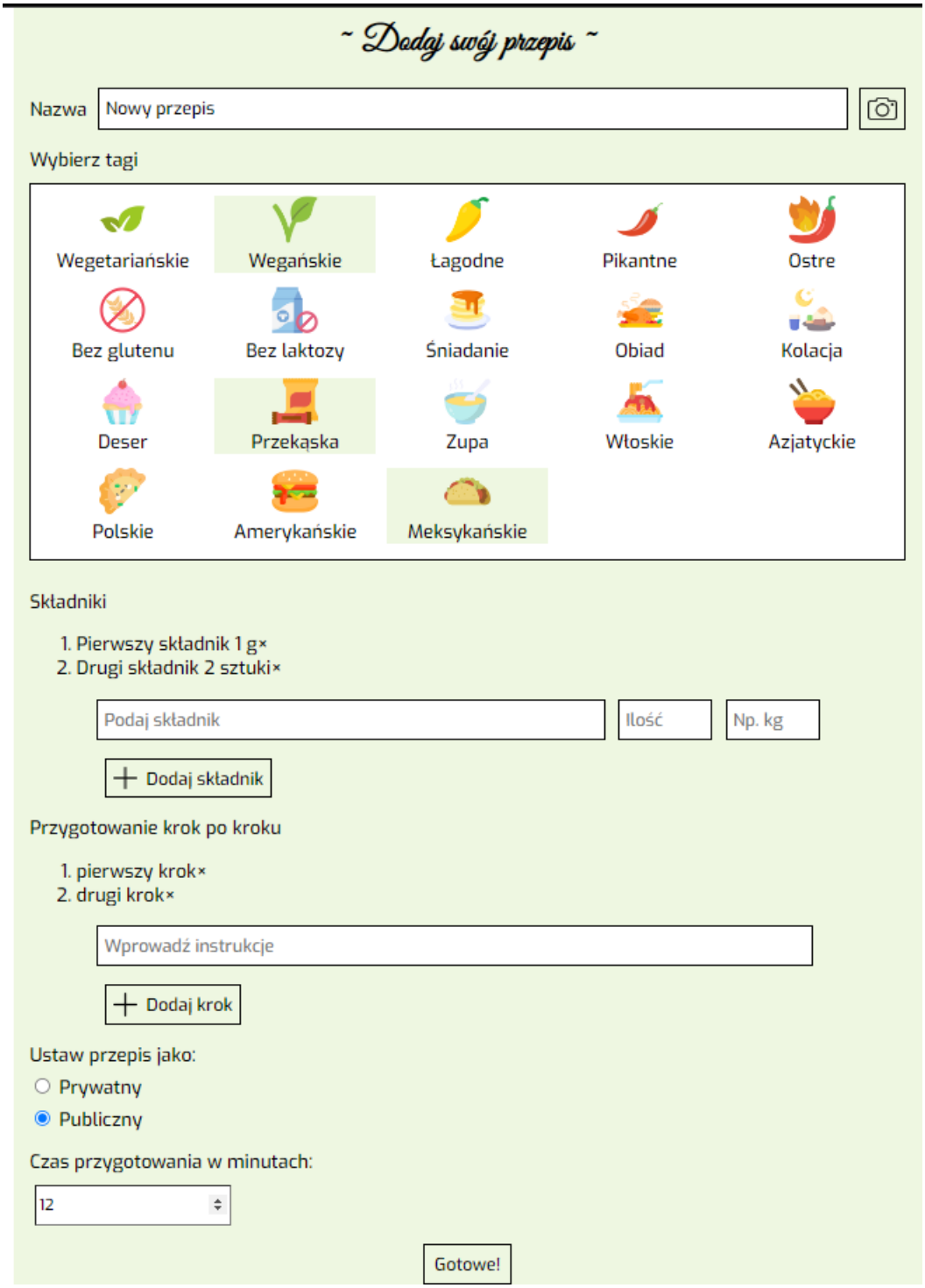
Rysunek 9 Wyszukiwanie zaawansowane

Odnalezione przepisy pojawiają się poniżej (Rysunek 10).



Rysunek 10 Znalezione przepisy

Dodawanie przepisu posiada prosty interfejs: wymaga podania nazwy, wyboru tagów, listy składników oraz kroków przygotowania. Opcjonalnie można również dodać zdjęcie. Pusty formularz widać na rysunku 11.



Rysunek 11 Dodawanie nowego przepisu

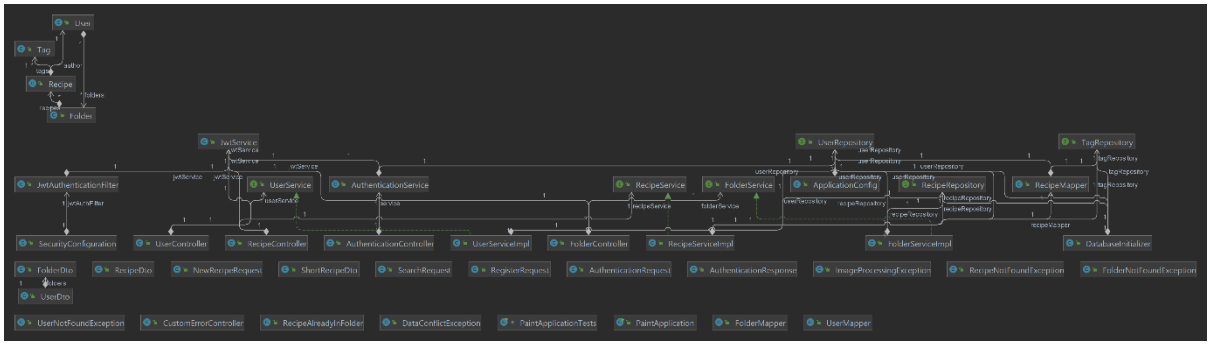
Warstwa logiki biznesowej:

Klasy języka Java zostały podzielone zgodnie ze swoimi funkcjami. W ten sposób powstały paczki z klasami(i czasem wewnętrznymi paczkami) odpowiedzialnymi za:

- *controller* - obsługę żądań od przeglądarki(endpointy). Każda z klas modelu posiada własny kontroler obsługujący zdarzenia z nią związane, dodatkowo utworzone zostały kontrolery błędów oraz autoryzacji.
- *DTOs* – zdefiniowanie(*model*) i mapowanie(*mappers*) obiektów do postaci odpowiedniej do transportu przez HTTP, a także definiującą typy żądań, które serwer ma przyjmować(*requests).*
- *exception* – definicje niestandardowych typów wyjątków aplikacji.
- *model* – definicje klas będących odwzorowaniem danych, wraz z deklaracją relacji między nimi. Klasami modelu są: *Folder*, *Recipe, User i Tag.*
- *repository* – deklaracje metod umożliwiających komunikację aplikacji z bazą danych. Korzystamy z możliwości automatycznej generacji zapytań dzięki dziedziczeniu po klasie *MongoRepository<klasa, Id>.* W przypadku implementacji zaawansowanego wyszukiwania niezbędna jednak okazała się własna implementacja fragmentów zapytań.
- *security* – bezpieczeństwo aplikacji
- *service* – szeroko pojętą logikę biznesową, zawarte klasy zazwyczaj zawierają obsługę zdarzeń, które odbierają klasy paczki *controller*

Poza tym zdecydowaliśmy się stworzyć dodatkową klasę, która jest odpowiedzialna za wstępne dodanie do bazy danych pewnej porcji danych.

Diagramy wszystkich klas użytych po stronie serwera przedstawione są na rysunku 12.



Rysunek 12 Klasy

Główne funkcje systemu

Zarządzanie sesją

Aby korzystać z niektórych funkcjonalności aplikacji, nie jest potrzebne zalogowanie się. Jednakże, w przypadku próby uzyskania dostępu do chronionych zasobów bądź funkcjonalności – takich jak usuwanie przepisów czy przegląd przepisów prywatnych, niezbędne jest uwierzytelnienie poprzedzone autoryzacją. Autoryzacja dokonywana jest poprzez zalogowanie się przy użyciu loginu i hasła. Otrzymane hasło jest następnie odpowiednio kodowane i porównywane z tym w bazie danych. W przypadku udanej autoryzacji serwer w odpowiedzi przesyła żądanie ustawienia ciasteczka zawierającego token służący do uwierzytelnienia.

Przetwarzanie danych

Klient w zależności od funkcjonalności może przesyłać parametry żądania poprzez użycie zmiennej w ścieżce URL, bądź w treści żądania(*request body*).

Otrzymane żądania są sprawdzane pod kątem poprawności – jeżeli jej struktura jest niezgodna z tą zdefiniowaną w aplikacji, następuje obsługa błędu, klient JavaScript zostaje poinformowany o rodzaju

błędu, aby odpowiednio zareagować. Jeżeli błąd nie wystąpił, następuje sprawdzenie, czy zasób jest w pewien sposób chroniony i jeżeli tak, następuje uwierzytelnienie żądania przez sprawdzenie ciasteczka. Po udanym uwierzytelnieniu można wykonać odpowiednie działanie związane z logiką biznesową.

Serwis obsługuje metody GET, POST, PUT oraz DELETE. W zależności od typu zapytania schematy działania mogą się różnić. Obsługa metody *GET* polega zazwyczaj na odnalezieniu i zwróceniu klientowi zasobu w bazie danych na podstawie informacji zawartych w żądaniu. Takimi informacjami mogą być na przykład: nazwa użytkownika, nazwa przepisu, tagi. W przypadku metody *POST* serwer tworzy nowy zasób na podstawie informacji zawartych w żądaniu, zapisuje go w bazie danych, a następnie zwraca klientowi informację o powodzeniu lub niepowodzeniu zadania. Za pomocą tej metody tworzony jest nowy przepis bądź folder, ale służy on również do autoryzacji. Metody PUT i DELETE niejako łączą ze sobą dwie poprzednie – wymagają odnalezienia zasobu w bazie, jego modyfikację lub delecję, a na końcu zapis i zwrócenie informacji użytkownikowi o powodzeniu lub niepowodzeniu.

Warstwa danych:

Ze względu na dokumentową strukturę przepisów oraz folderów zdecydowaliśmy się użyć MongoDB jako bazy danych. Ustawienia aplikacji poszukują lokalnie działającej bazy na standardowym porcie 27017. Nazwą bazy jest **przepisnik**, a wewnątrz niej zaimplementowane są kolekcje z dokumentami o poniższej strukturze([ ] – tablica).

```
Recipes {
        Id:  ObjectId
        Name: string
        Author: DBRef(users)
        Status: Boolean
        Tags:  [ DBRef(tags) ]
        Ingredients: [ string ]
        Steps: [ string ]
        timeMinutes: integer
        image: BinData
        }
Tags{
        Id: Objectd
        Name: string
}
Users{
        Id: ObjectId
        Username: string
```

Password: string

Email: string

Folders: [

{

Name: string

Recipes: [DBRef(recipes)]

}

]

Bezpieczeństwo:

Przy rejestracji użytkownik podaje swój login, email i hasło. Po sprawdzeniu czy login bądź hasło nie powtarzają się, podane przez użytkownika hasło zostaje odpowiednio zakodowane, aby mogło być przechowywane bezpiecznie w bazie. Podczas logowania przesłane przez użytkownika hasło również jest kodowane i wraz z loginem zostaje sprawdzone z przechowywanym w bazie danych.

Po pomyślnym zalogowaniu serwer wysyła w odpowiedzi polecenie ustawienia przez przeglądarkę ciasteczka z tokenem autoryzacyjnym. Token JWT zawiera między innymi: nazwę uwierzytelniającego się nim użytkownika, datę ważności oraz podpis. Podpis służy do weryfikacji autentyczności danych w tokenie, próba kradzieży tożsamości innego użytkownika lub przedłużenia czasu ważności tokenu powoduje, że podpis tokenu jest niezgodny z jego treścią. Niemożliwe jest również stworzenie przez atakującego własnego tokenu, ponieważ przy tworzeniu podpisu używany jest również pewien sekret, nigdy nie opuszczający serwera. Niemożliwe jest zatem prawidłowe podpisanie tokenu przez niezaufany podmiot.

Łańcuch filtrujący zapytania oraz sprawdzający je został wykonany przy pomocy Spring Security. Umożliwia ustawienie szczególnym typom żądań szczególnych reakcji – uwierzytelnienia użytkownika. Dzięki łańcuchowi ustawiono sprawdzanie posiadanie tokenu jedynie podczas dostępu do stron dostępnych tylko dla zalogowanych użytkowników – przegląd przepisów lub tworzenie nowego przepisu oraz uniemożliwiono wykonanie działań takich jak usuwanie przepisów bądź folderów, zmiana statusu przepisu z prywatnego na publiczny przez API. W tych ostatnich przypadkach sprawdzenie, czy użytkownik jest zalogowany może okazać się niewystarczające, dlatego porównywana jest wówczas tożsamość deklarowana wewnątrz tokenu z posiadaczem zmienianego zasobu. Dzięki temu użytkownik nie może na przykład usunąć czyjegoś przepisu.

Wykorzystywane zależności

Przez Mavena, wewnątrz pliku pom.xml

- **spring-boot-starter:** Jest to podstawowa zależność w projekcie Spring Boot, która dostarcza niezbędne funkcjonalności i konfiguracje dla uruchamiania aplikacji Spring.
- **spring-boot-devtools:** Zależność ta dostarcza narzędzia deweloperskie do automatycznego odświeżania aplikacji podczas rozwoju, np. automatyczne przeładowywanie zmian w kodzie, restart aplikacji itp.
- **spring-boot-starter-data-mongodb:** zapewnia integrację z bazą danych MongoDB.

- **spring-boot-starter-web: u**możliwia obsługę żądań HTTP, routowanie, obsługę widoków, serializację/deserializację danych itp.
- **springdoc-openapi-starter-webmvc-ui:** umożliwia wygenerowanie interaktywnego interfejsu użytkownika do przeglądania i testowania API.
- **spring-boot-starter-security:** umożliwiaja implementację mechanizmów zabezpieczeń w aplikacji Spring. Umożliwia uwierzytelnianie, autoryzację, zarządzanie sesjami, zabezpieczenia CSRF itp.
- **Lombok:** biblioteka Java, która automatycznie generuje kod dla rutynowych zadań, takich jak generowanie getterów, setterów, konstruktorów, metod toString() itp. Pozwala zmniejszyć objętośc kodu
- **jjwt-api, jjwt-impl, jjwt-jackson:** dotyczącą biblioteki JJWT, która umożliwia generowanie, weryfikację i obsługę JWT.
- **spring-boot-maven-plugin:** plugin Mavena, który ułatwia budowanie, kompilowanie i uruchamianie aplikacji Spring Boot bez użycia IDE.

Sposób ich konfiguracji – zawartość pliku application.properties

```
spring.data.mongodb.authentication-database=admin

spring.data.mongodb.username=admin

spring.data.mongodb.database=przepisnik

spring.data.mongodb.port=27017

spring.data.mongodb.host=localhost
```

Instalacja, uruchomienie

1. Uruchomienie MongoDB na porcie 27017 lub innym, określonym w application.properties
2. Uruchomienie serwisu
    a. Z poziomu IDE(w naszym przypadku Intellij)
    b. Za pomocą narzędzia Maven → mvn spring-boot:run
3. W przeglądarce: localhost:8080