

POLITECHNIKA WROCŁAWSKA  
WYDZIAŁ ELEKTRONIKI

---

KIERUNEK: INFORMATYKA (INF)

SPECJALNOŚĆ: INŻYNIERIA SYSTEMÓW INFORMATYCZNYCH (INS)

PRACA DYPLOMOWA  
INŻYNIERSKA

Aplikacja webowa do zarządzania domowym  
ogrodem

A web application for managing a home garden

AUTOR:

Łukasz Broll

PROWADZĄCY PRACĘ:

Dr inż. Marek Woda, W4/K9

OCENA PRACY:

---

WROCŁAW, 2018

# Spis treści

1.	Wstęp.....	4
1.1.	Cel pracy.....	4
1.2.	Zakres pracy .....	5
1.3.	Układ pracy.....	5
2.	Specyfikacja wymagań.....	6
2.1.	Wymagania funkcjonalne .....	6
2.2.	Wymagania niefunkcjonalne .....	7
3.	Projekt .....	8
3.1.	Architektura .....	8
3.1.1.	Wzorzec MVC.....	9
3.1.2.	Baza danych MySQL .....	9
3.1.3.	Aplikacja serwerowa Spring .....	9
3.1.4.	Aplikacja kliencka Angular.....	12
3.2.	Diagram przypadków użycia .....	13
3.3.	Diagram encji .....	14
3.3.1.	Encja User .....	14
3.3.2.	Encja Garden .....	14
3.3.3.	Encja Plant.....	15
3.4.	Diagram klas modelu.....	15
3.5.	Narzędzia .....	16
4.	Implementacja .....	17
4.1.	Pakiety .....	17
4.2.	Komponenty .....	18
4.3.	Adresowanie .....	19
4.3.1.	Adresowanie w aplikacji serwerowej .....	20

4.3.2.	Adresowanie w aplikacji klienckiej .....	21
4.4.	Zabezpieczenia .....	21
4.4.1.	Autoryzacja konta przez e-mail.....	22
4.5.	Automatyczne sprawdzanie statusu pracy .....	25
4.6.	Cykliczny e-mail z listą ogrodów wymagających uwagi .....	25
4.7.	Czat ogrodników.....	27
5.	Testy .....	29
5.1.	Testy jednostkowe modeli .....	29
5.2.	Testy funkcjonalne .....	30
5.3.	Testy kontrolerów .....	31
6.	Aplikacja .....	32
7.	Podsumowanie .....	40
	Spis rysunków .....	41
	Spis listingów .....	42
	Literatura .....	43

# 1. Wstęp

Roślinność to nieodzowny element życia ludzi. Bez tlenu, który wytwarzają rośliny, życie na Ziemi by nie istniało. W dobie bardzo dużego zużycia zasobów leśnych, m.in. wycinki lasów tropikalnych, ważne jest dbanie o możliwie jak najmniejszą degradację środowiska naturalnego. Każda, nawet najmniejsza roślina ma wpływ na natlenienie powietrza i redukcję dwutlenku węgla, dzięki czemu niweluje szkodliwy wpływ przemysłu na naturę. Czy to duży domowy ogród, czy kwiat na oknie, warto dbać o obecność roślin. Niestety każda roślina wymaga mniejszej lub większej uwagi do życia. Należy dbać o nawodnienie, nasłonecznienie i ochronę przed bakteriami i wirusami. Tylko regularna opieka gwarantuje profity z posiadania domowego ogrodu – wizualne oraz zdrowotne.

## 1.1. Cel pracy

Celem pracy jest utworzenie narzędzia wspomagającego zarządzanie domowym ogrodem, które wspierać będzie regularne prace i obowiązki, konieczne do utrzymania zdrowia roślin. Dzięki przechowywaniu w jednym miejscu informacji o wszystkich swoich roślinach i ogrodach, łatwiej będzie kontrolować bieżące obowiązki, a listy roślin wymagających uwagi, ułatwią i przyspieszą obowiązki. Narzędzie zostanie zrealizowane w postaci aplikacji webowej, tj. programu działającego na serwerze, dostępnego tylko za pośrednictwem urządzenia z dostępem do sieci Internet, bez względu na używany system operacyjny, gdzie klientem aplikacji jest dostępna przeglądarka internetowa. Wszystkie zasoby i dane zapisywane przez użytkownika w tego typu aplikacji, przechowywane są w zewnętrznej bazie danych. Zastosowane technologie mają gwarantować, że obsługa aplikacji będzie intuicyjna, a interfejs nowoczesny.

## **1.2. Zakres pracy**

Początkowym etapem pracy będzie specyfikacja wymagań funkcjonalnych oraz nie-funkcjonalnych aplikacji. Na tym etapie, konieczne jest dokładne zdefiniowanie co ma oferować aplikacja oraz czego nie będzie oferować. W kolejnym etapie zostaną wybrane technologie, w których projekt zostanie zrealizowany, a następnie zostanie utworzony projekt aplikacji serwerowej i klienckiej oraz model bazy danych. Następnym krokiem będzie implementacja rozwiązania oraz jego testy. Na końcu wykonana zostanie dokumentacja aplikacji.

## **1.3. Układ pracy**

Kolejne rozdziały pracy odnoszą się do etapów tworzenia aplikacji. W 2. rozdziale opisane są wymagania, jakie musi spełniać aplikacja. W 3. rozdziale zawarte są informacje o architekturze aplikacji. Przedstawiony jest projekt systemu, struktura bazy danych i klas oraz wykorzystane narzędzia. W 4. rozdziale opisana jest implementacja aplikacji, czyli m.in. połączenia między pakietami, adresowania kontrolerów, zastosowane zabezpieczenia oraz niektóre z implementacji funkcjonalności. W rozdziale 5. opisany jest proces testowania, a w rozdziale 6. zaprezentowany jest wygląd i funkcjonalność gotowej aplikacji. W ostatnim rozdziale znajduje się podsumowanie pracy.

## 2. Specyfikacja wymagań

W rozdziale opisane są wymagania, jakie musi spełniać aplikacja do zarządzani domowym ogrodem. Podstawowym wymaganiem jest udostępnienie użytkownikami narzędzia do zarządzania ogrodem i roślinami, które będzie dostępne w języku polskim. Poniżej wymienione zostały wymagania funkcjonalne oraz нефункционалне.

### 2.1. Wymagania funkcjonalne

1. Logowanie do systemu opiera się o nazwę użytkownika oraz hasło. W przypadku rejestracji należy podać również imię, nazwisko oraz adres e-mail.
2. Konto do czasu aktywacji przez e-mail na podstawie kodu weryfikującego jest nieaktywne – można się zalogować, ale funkcjonalności są wyłączone.
3. Hasło jest przechowywane w bazie danych w postaci zaszyfrowanej za pomocą funkcji bcrypt[10].
4. Użytkownik może tworzyć ogrody, które będą przypisane tylko do jego konta. Każdy ogród identyfikowany jest poprzez nazwę oraz można go edytować lub usunąć. Przechowywane są również czasy utworzenia i ostatniej edycji ogrodu oraz jego opis.
5. W ogrodzie przechowywana jest lista roślin, a każda roślina składa się na:
  - a. nazwę,
  - b. opis,
  - c. notatki,
  - d. prace związane z rośliną, tj. podlewanie, nawożenie, przesadzanie, dawkowanie środków ochrony roślin – w skład każdej pracy wchodzi:
    - i. cykl wykonywania czynności,
    - ii. ostatni termin jej wykonania,
    - iii. aktualny status czynności.
  - e. czas utworzenia oraz ostatniej modyfikacji rośliny,
6. Wszystkie dane rośliny, oprócz ostatniego wykonywania każdej z czynności, można edytować, a samą roślinę usuwać.
7. Cykl wykonywania każdej pracy związanej z rośliną, jest wybierany z zdefiniowanej listy terminów.

8. Aktualny status pracy związanej z rośliną jest automatycznie aktualizowany, jeśli minął termin jej ostatniego wykonania.
9. Informacje o roślinie, na podstawie jej nazwy, można wyszukać bezpośrednio z poziomu aplikacji w zewnętrznych serwisach.
10. Do użytkownika cyklicznie wysyłany jest e-mail z listą ogrodów, w których są oczekujące prace.

## 2.2. Wymagania niefunkcjonalne

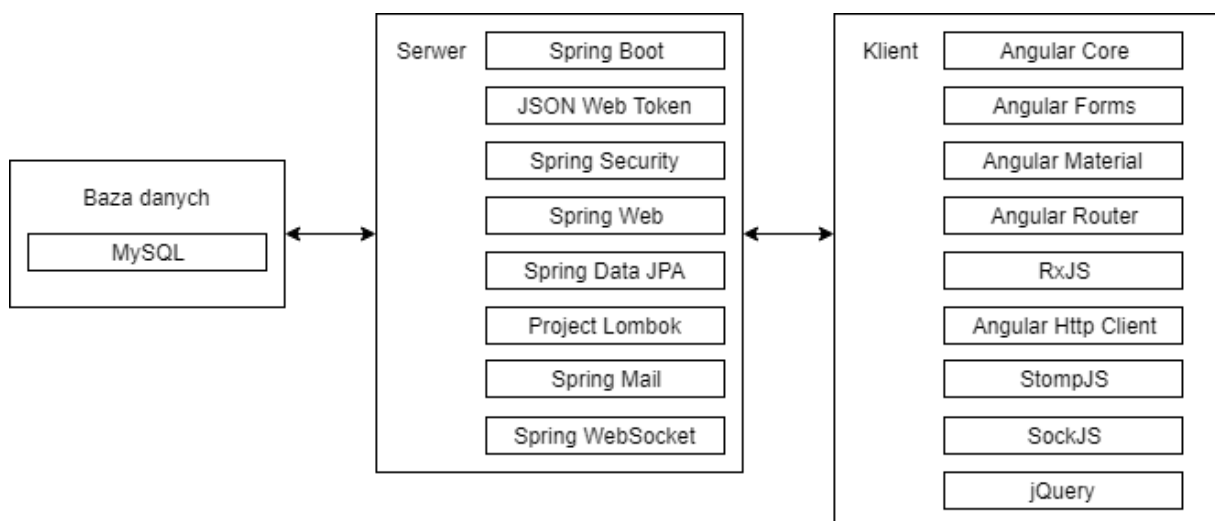
1. Aplikacja powinna działać poprawnie na poniższych przeglądarkach:
  - **Google Chrome** w wersji 60 lub wyższej,
  - **Mozilla Firefox** w wersji 54 lub wyższej,
2. Technologie oraz rozwiązania systemów informatycznych, wykorzystane do utworzenia aplikacji, powinny być darmowe.
3. Aplikacja powinna działać, niezależnie od systemu operacyjnego użytkownika.
4. Obsługa aplikacji powinna być intuicyjna i wygodna dla każdego użytkownika.

## 3. Projekt

Projekt został oparty o technologie związane z językiem Java. Taka decyzja została podjęta, ponieważ tok studiów jest ściśle związany z tym językiem oraz autor ma w nim największe doświadczenie. Czynnikiem decydującym o skorzystaniu z wybranych bibliotek oraz rozwiązań, były ich popularność oraz nowoczesność, co wiąże się z dużą społecznością wsparcia i dokumentacji.

### 3.1. Architektura

Aplikacja została podzielona na trzy moduły: bazę danych, aplikację serwerową oraz kliencką. Dzięki takiemu rozwiązaniu łatwiej jest kontrolować działanie całego systemu oraz modyfikować poszczególne komponenty. W związku z tym, schemat systemu razem z wykorzystanymi technologiami, wygląda następująco:



Rysunek 1 Architektura systemu



### 3.1.1. Wzorzec MVC

Rozwinięciem skrótu MVC jest Model-View-Controller, czyli Model-Widok-Kontroler[1]. Jest to wzorzec projektowy pozwalający na podział struktury aplikacji na moduły.

W przypadku aplikacji webowej, wzorzec MVC wykorzystywany jest w następujący sposób:

1. Kontroler odbiera żądanie http, na którego podstawie odpowiednio przetwarza i interpretuje dane wejściowe.
2. Model odbiera przetworzone dane wejściowe z kontrolera i przetwarza je, np. aktualizując stan obiektu.
3. Widok odbiera z powrotem dane z kontrolera, w przypadku aplikacji projektowej, w formacie JSON i aktualizuje wyświetlane komponenty.

### 3.1.2. Baza danych MySQL

MySQL[9] to jeden z najpopularniejszych systemów zarządzania bazą danych. Został wykorzystany w projekcie, ponieważ bardzo dobrze współpracuje z wykorzystanymi technologiami w aplikacji serwerowej, nie wymaga skomplikowanej konfiguracji oraz umożliwia bezproblemowe przeniesienie całej bazy na np. zewnętrzny serwer.

### 3.1.3. Aplikacja serwerowa Spring

Spring[2] to szkielet, służący do tworzenia aplikacji głównie w języku Java. Oferuje rozwiązania usprawniające prace nad rozwijaniem oprogramowania, których wykorzystanie znacznie redukuje ilość kodu konieczną do napisania przez programistę. M.in. usprawnia obsługiwanie wyjątków, komunikację z bazą danych oraz umożliwia wstrzykiwanie zależności. W projekcie, wykorzystane zostało dodatkowo narzędzie **Spring Boot**, umożliwiające automatyzację konfiguracji oraz instalacji paczek bibliotek, które zawierają potrzebne narzędzia.

W ramach Spring Boota zaimportowano następujące paczki bibliotek:

1. **Spring Security** – biblioteka do zarządzania bezpieczeństwem aplikacji, chroni zasoby przed nieautoryzowanym dostępem.
2. **Spring Web** – biblioteka dostarczająca zbiór funkcjonalności usprawniających tworzenie aplikacji webowych.
3. **Spring Data JPA** – biblioteka usprawniająca dostęp do źródła danych, która na podstawie oficjalnego standardu JPA (Java Persistence API), umożliwia uproszczone mapowanie na obiekty zawartości bazy danych.
4. **Spring Mail** – biblioteka umożliwiająca komunikację z użytkownikami za pomocą wiadomości e-mail.
5. **Spring WebSocket** – biblioteka zawierająca narzędzia do obsługi czatu w aplikacji.

#### 3.1.3.1. Project Lombok

Biblioteka ograniczająca generowanie wielu linii kodu powtarzającego się kodu podczas definiowania klas, szczególnie klas modelu. Za pomocą adnotacji możliwe jest automatycznie tworzenie np. konstruktora bezargumentowego (`@NoArgsConstructor`), metod zwracających pola klasy (`@Getter`), metod ustawiających pola klasy (`@Setter`) czy zabezpieczenie przed brakiem wartości pola (`@NonNull`).

### 3.1.3.2. JSON Web Token

Standard generowania tokenów uwierzytelniających sesje użytkownika. Token[4] jest przechowywany po stronie klienta i przesłany w zapytaniach do serwera, który na podstawie klucza weryfikuje, czy zezwolić na wykonanie zapytania i dostęp do zasobów. Jak nazwa wskazuje, token jest oparty na formacie JSON i składa się z trzech części – nagłówka, zawartości i sygnatury. W nagłówku znajdują się informacje o wykorzystanym algorytmie szyfrującym. W drugiej części przechowywane są informacje o użytkowniku, m.in. nazwa użytkownika oraz czas sesji. W sygnaturze przechowywany jest cyfrowy podpis serwera. Szczegóły implementacji w aplikacji zostały opisane w rozdziale 4.3. „Zabezpieczenia”. Poniżej znajduje się odpowiednio zakodowany i zdekodowany przykładowy token wykorzystywany w aplikacji. Do prezentacji wykorzystano oficjalne narzędzie znajdujące się na stronie [jwt.io](https://jwt.io).

**Encoded**
PASTE A TOKEN HERE

```

eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJwbHVzZXI
iLCJzY29wZXMiOiI7ImF1dGhvcm10eSI6IlVTRVI
ifV0sImZcyI6ImxvY2FsaG9zdCI6Im1hdCI6MTU
0Mzc3MzkwMiwiaXhwIjoxNTQzNzIxOTYyLWV0
-hCX1ZmqxhLn2pcU4MvaHQs9MW4hBr_PfBAyk4s

```

**Decoded**
EDIT THE PAYLOAD AND SECRET

**HEADER: ALGORITHM & TOKEN TYPE**

```

{
  "alg": "HS256"
}

```

**PAYLOAD: DATA**

```

{
  "sub": "pluser",
  "scopes": [
    {
      "authority": "USER"
    }
  ],
  "iss": "localhost",
  "iat": 1543773902,
  "exp": 1543791902
}

```

**VERIFY SIGNATURE**

```

HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  lbrol1123
) ☒ secret base64 encoded

```

Rysunek 2 Kodowanie tokenu zabezpieczeń

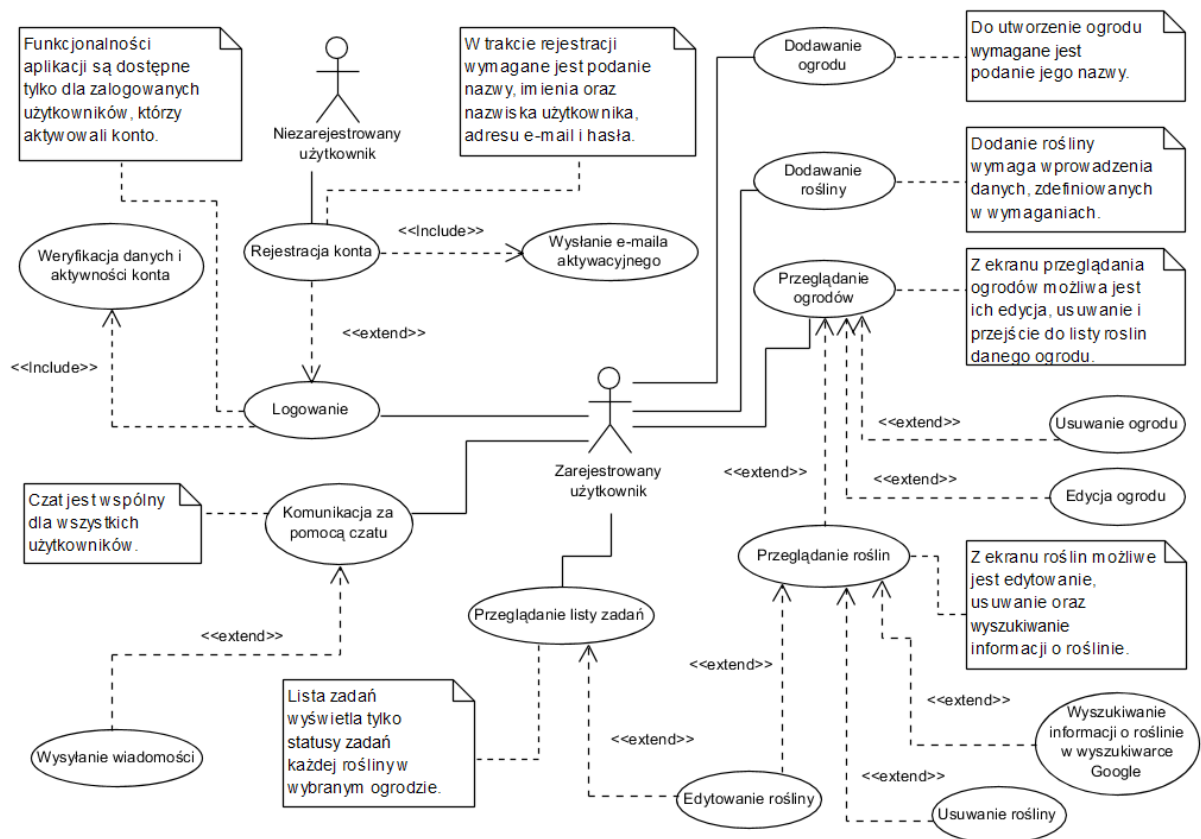
### 3.1.4. Aplikacja kliencka Angular

Angular[3] to biblioteka autorstwa Google, który ma spełniać podobne zadanie co Spring, jednak przeznaczony dla języka TypeScript. We wzorcu MVC odpowiada za Widok, czyli za interfejs użytkownika. Głównymi cechami charakteryzującymi Angulara są renderowanie po stronie serwera, zorientowania na komponenty oraz modularność. Wykorzystując dyrektywy, które są zagnieżdżone w kodzie HTML, możliwe jest łatwe kontrolowanie interfejsu użytkownika na podstawie logiki napisanej w języku TypeScript. Wiele metod i funkcji jest już zdefiniowanych w szkieletcie Angulara, przez co programista nie musi ręcznie tworzyć każdej dyrektywy. W projekcie, w ramach Angulara, wykorzystane zostały następujące moduły:

1. **Angular Core** – główny rdzeń i silnik Angulara, który odpowiada za działanie biblioteki.
2. **Angular Forms** – biblioteka dostarczająca zestaw metod i funkcjonalności usprawniających tworzenie formularzy, ich walidację oraz wysyłanie.
3. **Angular Material**[5] – biblioteka zawierająca komponenty wizualne stylu Material Design, który polega na dostarczeniu prostego, czytelnego interfejsu.
4. **Angular Router** – biblioteka usprawniająca przekierowania wewnątrz aplikacji, wspierająca m.in. obsługę parametrów w adresie.
5. **RxJS** – zestaw narzędzi który rozszerza możliwości Angulara o programowanie reaktywne, m.in. operacje na kolekcjach.
6. **Angular Http Client** – biblioteka wspierająca komunikację z aplikacją serwerową poprzez protokół http.
7. **StompJs, SockJS, jQuery** – biblioteki wykorzystane do obsługi czatu w aplikacji klienckiej.

### 3.2. Diagram przypadków użycia

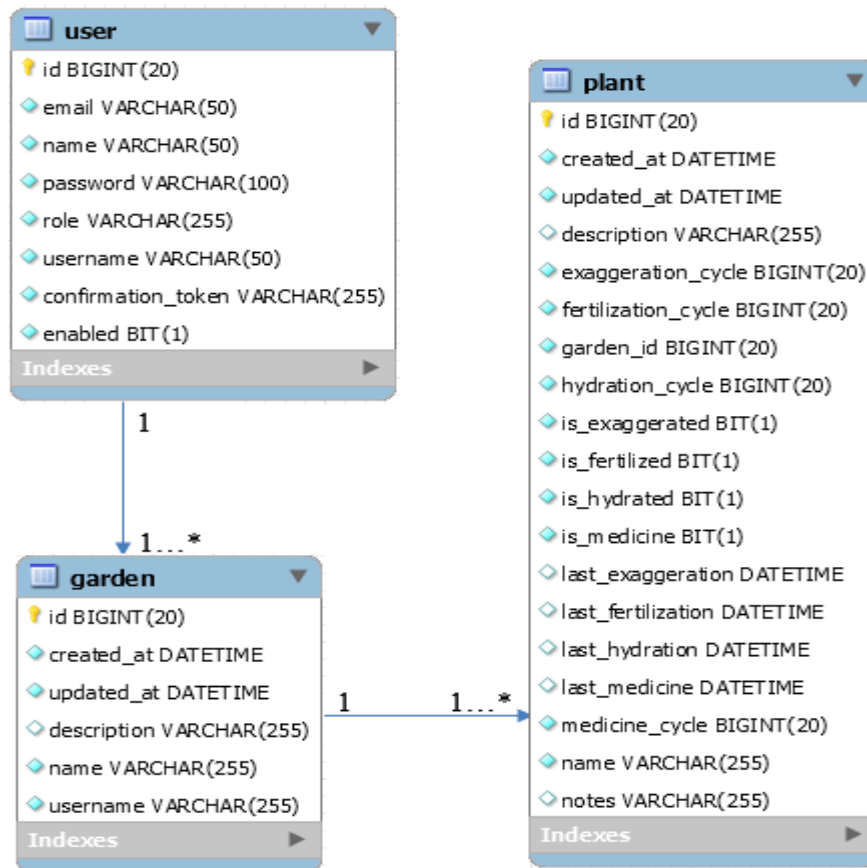
Po uwzględnieniu wszystkich wymagań aplikacji, możliwe było skonstruowanie diagramu przypadków użycia, który bez uwzględniania implementacji, określa możliwości jakie oferuje aplikacja.



Rysunek 3 Diagram przypadków użycia

### 3.3. Diagram encji

Diagram przedstawia encje bazy danych dla modelu klas, które są mapowane w aplikacji za pomocą biblioteki **Hibernate**[6].



Rysunek 4 Diagram encji

#### 3.3.1. Encja User

User, czyli użytkownik jest identyfikowany po polu `username` (nazwa użytkownika) oraz ID. Każdy użytkownik ma unikalny `username` oraz adres e-mail. Hasło użytkownika tj. pole `password` jest zakodowane za pomocą funkcji `bcrypt`. Domyślną rolą każdego użytkownika jest „USER”. Użytkownik może posiadać nieograniczoną liczbę ogrodów.

#### 3.3.2. Encja Garden

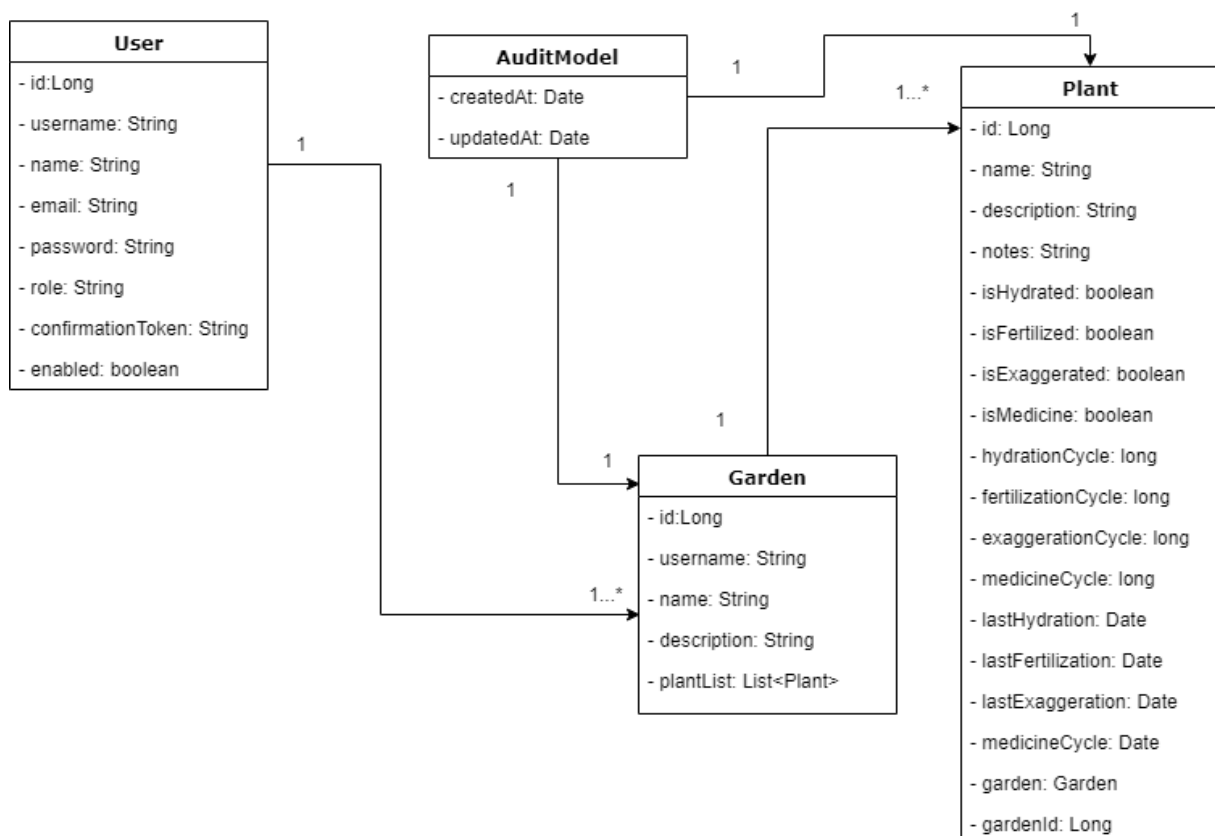
Garden, czyli ogród jest identyfikowany po polu ID. Może istnieć tylko w relacji z użytkownikiem. Zawiera informacje o nazwie, opisie, dacie powstania i ostatniej modyfikacji. Każdy ogród może posiadać nieograniczoną liczbę roślin.

### 3.3.3. Encja Plant

Plant, czyli roślina jest identyfikowana po ID. Jest powiązana z konkretnym ogrodem, czyli również użytkownikiem. Każda roślina musi posiadać informacje o pracy związanej z podlewaniem, nawożeniem, przesadzaniem i podawaniem środków ochrony roślin. Dodatkowo może mieć opis oraz notatki.

## 3.4. Diagram klas modelu

Na diagramie nie uwzględniono konstruktorów klas, getterów oraz setterów pól, które są generowane automatycznie z pomocą biblioteki Lombok. Nie zostały uwzględnione również klasy, które przechowują modele odpowiedzialne za autoryzację i zabezpieczenia aplikacji tj. AuthToken, Constants, LoginUser, ponieważ są to modele pomocnicze, które nie są mapowane na encje bazy danych.



Rysunek 5 Diagram klas modelu

Ze względu na wykorzystanie architektury MVC, nie został zaprojektowany diagram wszystkich klas w projekcie.

### 3.5. Narzędzia

Wybór narzędzi potrzebnych do utworzenia aplikacji, opierał się na zagwarantowaniu wsparcia dla wykorzystywanych technologii oraz języków programowania. Jednym z warunków była również darmowa licencja na oprogramowanie. Zdecydowano się wykorzystać następujące aplikacje:

1. **JetBrains IntelliJ IDEA** – rozbudowane zintegrowane środowisko programistyczne oferujące wsparcie podczas tworzenia aplikacji serwerowej oraz klienckiej. Dla studentów wersja aplikacji Ultimate jest darmowa.
2. **MySQL Workbench** – graficzny interfejs bazy danych MySQL, umożliwiający projektowanie, administrację oraz kontrolę bazy danych.
3. **Postman** – aplikacja służąca do testowania oprogramowania, wysyłająca zapytania do kontrolerów aplikacji serwerowej.
4. **Maven** – narzędzie służące do automatyzacji procesu budowy oprogramowania oraz ułatwiające korzystanie z niezależnych bibliotek w aplikacji.
5. **Node.js** – środowisko i silnik działania aplikacji opartych o JavaScript ( w tym TypeScript), w skład, którego wchodzi również **npm**, czyli menadżer pakietów dla rozwiązań interfejsu użytkownika.
6. **Git** – system kontroli wersji zapewniający możliwość kontrolowania procesu wytwarzania oprogramowania.
7. **Google Chrome** – przeglądarka internetowa na której testowana była budowa warstwy klienckiej aplikacji.

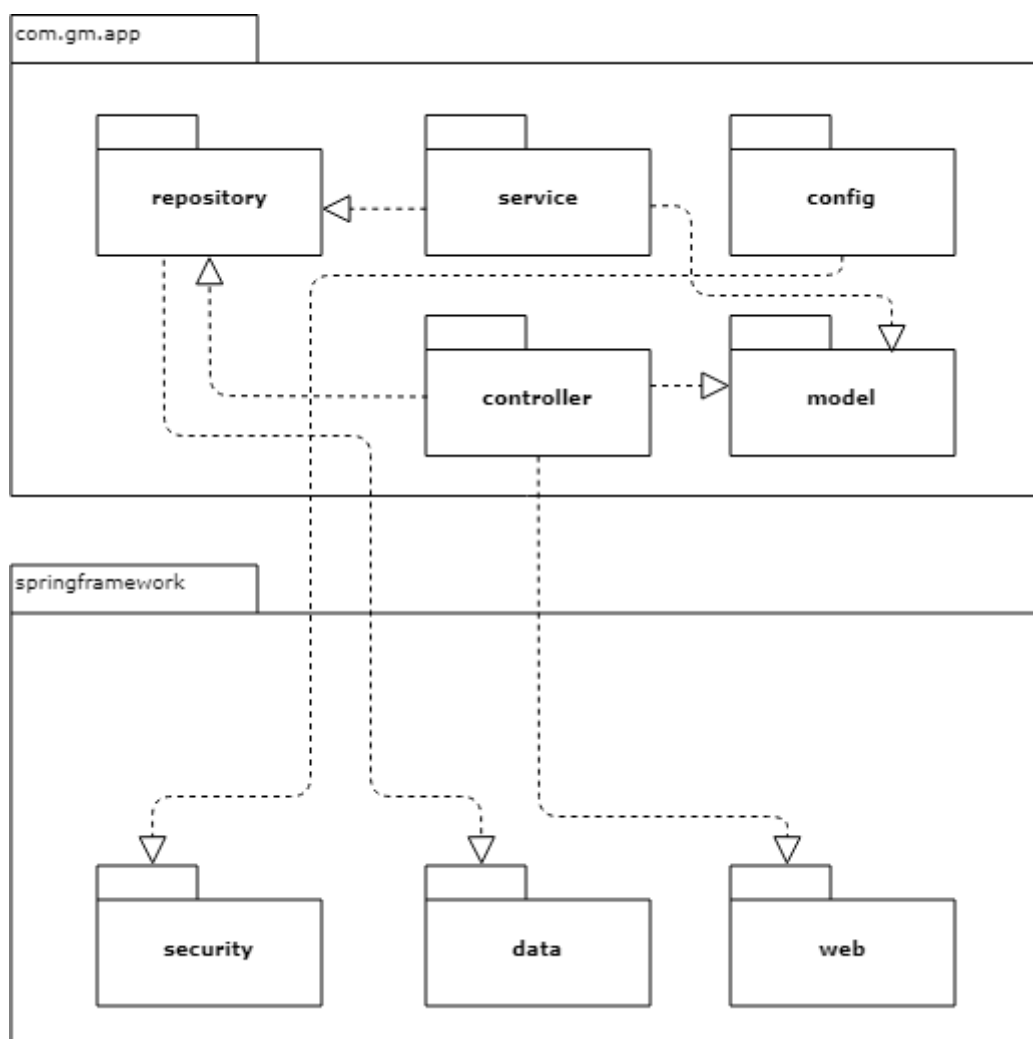


## 4. Implementacja

Proces implementacji aplikacji podzielony został między aplikację serwerową oraz kliencką. Korzystając z biblioteki **Hibernate**, nie było konieczne ręczne implementowanie bazy danych, ponieważ tabele i pola zostały zmapowane automatycznie podczas uruchamiania aplikacji. Opisane zostały głównie implementacje aplikacji serwerowej, ponieważ ta wymagała najwięcej uwagi podczas tworzenia.

### 4.1. Pakiety

Struktura aplikacji serwerowej oparta jest o pakiety, które zawierają klasy zgodne z przeznaczeniem pakietu. Wydzielenie pakietów według ich zastosowania ułatwia organizację projektu i uporządkowane rozwijanie nowych funkcjonalności w aplikacji.



Rysunek 6 Diagram pakietów aplikacji serwerowej

Na rysunku zostały przedstawione pakiety aplikacji oraz ich powiązania z pakietami frameworka Spring.

- Pakiet `com.gm.app.model` zawiera klasy reprezentujące modele, na których wykonywane są operacje.
- Pakiet `com.gm.app.controller` zawiera klasy kontrolerów dla obiektów w aplikacji oraz częściowo odpowiada za operacje na danych. W tym pakiecie są również zdefiniowane adresy metod, do których odwołuje się klient Angulara, dlatego jest on połączony z pakietem `com.springframework.web`.
- Pakiet `com.gm.app.service` zawiera implementacje metod serwisowych, które wykonują operacje na danych, głównie dla operacji związanych z logowaniem oraz rejestracją użytkownika.
- Pakiet `com.gm.app.repository` zawiera interfejsy wykorzystujące JPA Repository, za pomocą których komunikuje się z bazą danych.
- Pakiet `com.gm.app.config` zawiera metody odpowiedzialne za konfigurację aplikacji, głównie związane z zabezpieczeniami danych za pomocą JWT.

## 4.2. Komponenty

W przypadku aplikacji klienckiej, struktura opiera się na komponentach. Poszczególne funkcjonalności zaimplementowane w aplikacji serwerowej mają swoje odniesienie w komponentach aplikacji klienckiej. Każdy komponent zawiera 4 typy plików:

- `.ts` – pliki zawierające klasy z metodami do przetwarzania danych,
- `.spec.ts` – przygotowane klasy testowe dla komponentów,
- `.html` – pliki odpowiedzialne za budowę interfejsu danego komponentu.
- `.css` – pliki przechowujące klasy stylów dla interfejsu w pliku html.

Aplikacja składa się z następujących komponentów:

- `app` – główny komponent, zawierający konfigurację aplikacji, importowane biblioteki, adresowania oraz wykorzystane komponenty i klasy serwisowe,
- `chat` – komponent odpowiadający za widok czatu,
- `register` – komponent wyświetlający ekran rejestracji,
- `confirm-registration` – komponent odpowiadający za aktywację użytkownika,
- `login` – komponent wyświetlający panel logowania,

- `home` – widok ekranu głównego aplikacji,
- `garden-list` – komponent wyświetlający listę ogrodów,
- `garden-edit` – komponent zawierający metody związane z edycją ogrodu,
- `plant-list` – komponent wyświetlający listę roślin,
- `plant-edit` – komponent zawierający metody związane z edycją rośliny,
- `tasks-list` – komponent wyświetlający listę zadań.

Osobną grupą klas są komponenty serwisowe, które bezpośrednio komunikują się z kontrolerami aplikacji serwerowej. Należą do nich:

- `auth.service` – serwis autoryzujący logowanie użytkownika,
- `garden.service` – serwis obsługujący kontroler ogrodów,
- `plant.service` – serwis obsługujący kontroler roślin,
- `user.service` – serwis obsługujący kontroler użytkownika.

## 4.3. Adresowanie

W aplikacji adresowanie zaimplementowane zostało za pomocą adnotacji `@RequestMapping`, która jest jedną z funkcjonalności bibliotek Springa. Każdy kontroler obsługuje jeden adres w aplikacji. Przykładowe adresowanie kontrolera, który zwraca listę wszystkich ogrodów użytkownika, wygląda następująco:

Listing 1 Przykładowe adresowanie kontrolera

```
@GetMapping("/{username}/gardens")
public List<Garden> getAllGardensByUsername(@PathVariable (value =
"username") String username) {
    return gardenRepository.findByUsername(username);
}
```

Adnotacja `@GetMapping` jest skrótem adnotacji `@RequestMapping`, które wyglądałoby następująco:

```
@RequestMapping(value="/{username}/gardens", method = RequestMethod.GET)
```

- Właściwość `value` definiuje obsługiwany adres, którego parametrem jest `{username}`. Za pomocą adnotacji `@PathVariable`, parametr jest mapowany na argument funkcji.
- Właściwość `method` określa jakiego typu jest żądanie protokołu http. Określenie typu żądania umożliwia wykorzystanie tego samego adresu dla różnych metod.

### 4.3.1. Adresowanie w aplikacji serwerowej

Dla powiązanych ze sobą stron, zastosowany został ten sam prefiks adresowania, celem uporządkowania interfejsu komunikacji.

- Wszystkie adresy funkcji kontrolerów związanych z dostępem do danych ogrodów zaczynają się prefiksem parametru nazwy użytkownika: `/ {username} /`.
- Zgodnie z zależnościami, kolejnym poziomem adresowania są ogrody, więc pod uwagę brany jest typ działania na ogrodzie, odpowiednio jest to: zwracanie listy ogrodów użytkownika, informacje o danym ogrodzie, dodawanie ogrodu, edytowanie ogrodu oraz usuwanie ogrodu:
  - `/ {username} /gardens,`
  - `/ {username} /get-garden/ {gardenId},`
  - `/ {username} /add-gardens,`
  - `/ {username} /update-garden/ {gardenId},`
  - `/ {username} /delete-garden/ {gardenId}.`
- Dla kontrolera roślin prefiksem jest `/gardens`, ponieważ rośliny bezpośrednio należą do ogrodów. Odpowiednio jest to: zwracanie wszystkich roślin w ogrodzie, informacje o pojedynczej roślinie, dodanie nowej rośliny, edytowanie istniejącej oraz usunięcie rośliny:
  - `/gardens/ {gardenId} /plants,`
  - `/gardens/ {gardenId} /plant/ {plantId},`
  - `/gardens/ {gardenId} /add-plant,`
  - `/gardens/ {gardenId} /update-plant/ {plantId},`
  - `/gardens/ {gardenId} /delete-plant/ {plantId}.`
- Osobną grupą są adresy związane z zabezpieczeniami aplikacji. Adres `/register` odpowiada za rejestrację nowych użytkowników, `/generate-token` za utworzenie tokena uwierzytelniającego oraz zalogowanie do systemu oraz `/confirm-acc/ {token}` do aktywacji konta.
- Adres serwera czatu to `/webchat`, a adres, na którym nasłuchiwanie są wiadomości czatu to `/app/ chat`.

### 4.3.2. Adresowanie w aplikacji klienckiej

Adresy w aplikacji klienckiej dotyczą tych, które są bezpośrednio wykorzystywane przez użytkownika. Dopiero odpowiednie metody serwisowe odwołują się do zapytań kontrolerów aplikacji serwerowej. Dla każdego komponentu aplikacji klienckiej zdefiniowane zostały adresy, które obsługuje. W Angularze, lista typu `Routes` zawiera adresy, a parametry przekazywane są za pomocą dwukropka.

Listing 2 Konfiguracja adresowania w aplikacji klienckiej

```
const appRoutes: Routes = [
  {path: '', component: HomeComponent},
  {path: 'garden-list', component: GardenListComponent },
  {path: 'tasks-list', component: TasksListComponent },
  {path: 'plant-list/:id', component: PlantListComponent},
  {path: 'garden-add', component: GardenEditComponent},
  {path: 'garden/:gid/plant-add', component: PlantEditComponent},
  {path: 'garden/:gid/plant-edit/:pid', component: PlantEditComponent},
  {path: 'garden-edit/:id', component: GardenEditComponent},
  {path: 'register', component: RegisterComponent},
  {path: 'confirm/:code', component: ConfirmRegistrationComponent},
  {path: 'login', component: LoginComponent},
  {path: 'chat', component: ChatComponent},
];
```

## 4.4. Zabezpieczenia

Dostęp do zasobów aplikacji serwerowej zabezpieczony został za pomocą opisanego wcześniej JSON Web Token. Klasa `WebSecurityConfig` rozszerza klasę `WebSecurityConfigurerAdapter` z pakietu Spring, gdzie zdefiniowane są m.in adresy do funkcji, które nie wymagają autoryzacji.

Listing 3 Konfiguracja zabezpieczeń aplikacji

```
protected void configure(HttpSecurity http) throws Exception {
    http.cors().and().csrf().disable().
        authorizeRequests()
            .antMatchers("/generate-token", "/register",
"/confirm-acc/*", "/webchat/*").permitAll()
            .anyRequest().authenticated()
            .and()
            .exceptionHandling().
authenticationEntryPoint(unauthorizedHandler).and()
            .sessionManagement().
sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    http.addFilterBefore(authenticationTokenFilterBean(),
UsernamePasswordAuthenticationFilter.class);
}
```

Metody odpowiedzialne za rejestrację oraz generowanie tokenu z logowaniem nie są sprawdzane pod kątem zabezpieczeń, ponieważ to one decydują o dostępie do zasobów.

Token zabezpieczony jest algorytmem HS256 z kluczem `SIGNING_KEY = "lbroll123"`. Czas ważności tokenu ustawiony jest na 5 godzin. W sekcji zawartości przekazywana jest nazwa użytkownika (`user`), jego uprawnienia (`scopes`), emitent (`issuer`), data utworzenia (`issuedAt`) oraz wygaśnięcia (`expiration`). Konfiguracja metody `doGenerateToken` wygląda następująco:

Listing 4 Konfiguracja tokenu zabezpieczającego

```
private String doGenerateToken(String user) {
    Claims claims = Jwts.claims().setSubject(user);
    claims.put("scopes", Collections.singletonList(new SimpleGrantedAuthority("ROLE_USER")));

    return Jwts.builder()
        .setClaims(claims)
        .setIssuer("localhost")
        .setIssuedAt(new Date(System.currentTimeMillis()))
        .setExpiration(new Date(System.currentTimeMillis() +
ACCESS_TOKEN_VALIDITY_SECONDS*1000))
        .signWith(SignatureAlgorithm.HS256, SIGNING_KEY)
        .compact();
}
```

Hasło w bazie danych przechowywane jest za pomocą funkcji szyfrującej `bCrypt`. W Angularze zabezpieczenia polegają na ukryciu opcji w menu, które są dostępne tylko po zalogowaniu, natomiast `id` ogrodów i roślin w aplikacji klienckiej jest niezależne względem `id` wykorzystywanego w bazie danych.

#### 4.4.1. Autoryzacja konta przez e-mail

Celem uniknięcia zakładania masowej liczby kont przez użytkowników, nowo utworzone konto należy aktywować przed możliwością korzystania z niego. Po rejestracji, na podany adres e-mail wysyłany jest link zawierający kod uwierzytelniający do aktywacji konta. Dopiero otworzenie linku aktywuje konto i umożliwia korzystanie z aplikacji. Do implementacji rozwiązania wykorzystane zostały biblioteki z pakietu `spring-boot-starter-mail`, oraz serwis `userService`:

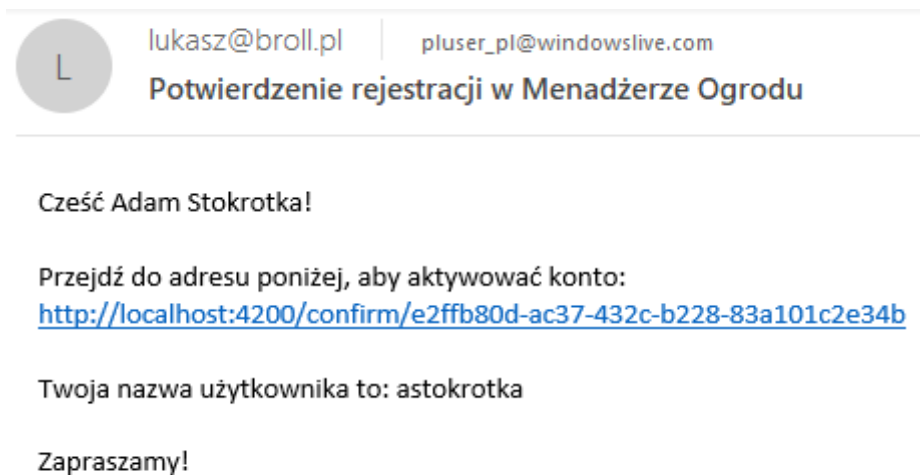
Listing 5 Metoda rejestracji nowego użytkownika

```
@RequestMapping(value="/register", method = RequestMethod.POST)
public User saveUser(@RequestBody User user){
    user.setEnabled(false);
    user.setConfirmationToken(UUID.randomUUID().toString());
    userService.save(user);
    SimpleMailMessage registrationEmail = new SimpleMailMessage();
```

```
registrationEmail.setTo(user.getEmail());
registrationEmail.setSubject("Potwierdzenie rejestracji w Menadżerze Ogrodu");
registrationEmail.setText("Przejdź do adresu poniżej, aby aktywować konto:\n"
+ "http://localhost:4200/confirm/" + user.getConfirmationToken());
registrationEmail.setFrom("lukasz@broll.pl");

emailService.sendEmail(registrationEmail);
return user; }
```

Na adres /register wysyłane jest zapytanie do metody saveUser z argumentem obiektu User. W pierwszej kolejności konto jest blokowane oraz generowany jest kod uwierzytelniający. Następnie userService dodaje nowego użytkownika do bazy danych oraz za pomocą emailService generowany i wysyłany jest następujący email do użytkownika:



Rysunek 7 Email potwierdzający rejestrację konta

Po przejściu na podany adres, klient Angulara rozpoczyna obsługę żądania. W tym przypadku, na ekranie, z pliku .html, wyświetlany jest tylko krótki komunikat o aktywacji konta:

#### Listing 6 Implementacja powiadomienia o aktywacji konta

```
<mat-card >
  <mat-card-header class="do-center">
    <mat-card-title><h2>Konto aktywowane!</h2></mat-card-title>
  </mat-card-header>
</mat-card>
```

Jednocześnie wywoływana jest metoda `ngOnInit()` z pliku `.ts`, która przekazuje kod uwierzytelniający do klasy serwisowej `user.service`, dzięki wykorzystaniu pakietu `@angular/router`.

Listing 7 Metoda przechwytyująca kod uwierzytelniający

```
ngOnInit() {
  this.sub = this.route.params.subscribe(params => {
    this.code = params['code'];
    this.userService.confirm(this.code).subscribe(result => {
    }, error => console.error(error));
  });
}
```

Serwis już bezpośrednio wysyła żądanie http do kontrolera aplikacji serwerowej. W tym przypadku jest to żądanie typu GET, a kontroler ma zwrócić obiekt przetworzonego użytkownika.

Listing 8 Metoda potwierdzająca konto, komunikująca się z kontrolerem.

```
public API = '://localhost:8080';
confirm(code: string) {
  return this.http.get(this.API + '/confirm-acc/' + code);
}
```

Następnie serwer przyjmuje żądanie na odpowiednim kontrolerze, który wyszukuje w bazie danych użytkownika z przekazanym kodem uwierzytelniającym i wywołuje metodę, która bezpośrednio aktualizuje status konta. Adnotacja `@Transactional` oznacza, że zostanie wykonana transakcja do bazy danych.

Listing 9 Kontroler obsługujący aktywację konta

```
@Transactional
@GetMapping("/confirm-acc/{code}")
public User processConfirmationForm(@PathVariable(value = "code")
String code)
{
    User user = userService.findByConfirmationToken(code);
    userRepository.activateUser(user.getId());
    return user;
}
```

Właściwa metoda zmieniająca status konta oznaczona jest adnotacją `@Modifying`, ponieważ wysyła zapytanie modyfikujące bazę danych. Treść zapytania oznaczona jest adnotacją `@Query`. Parametr `id` metody przekazywany jest za pomocą dwukropka `:id`. Metody wykonujące działania bezpośrednio na bazie danych muszą być w klasach oznaczonych adnotacją `@Repository`.



Listing 10 Metoda aktywująca konto bezpośrednio w bazie danych

```
@Modifying
@Query("UPDATE User t SET t.active = true WHERE t.id = :id")
void activateUser(@Param("id") Long id);
```

## 4.5. Automatyczne sprawdzanie statusu pracy

Jedną z głównych funkcjonalności aplikacji, jest kontrolowanie statusu wykonania prac na rośliną. Bieżące dane są przedstawiono w tabeli dla każdego ogrodu. Dla każdej z czynności wyświetlany jest aktualny status wykonania zadania. Do zaimplementowania tej funkcjonalności wykorzystany został kontroler pobierający listę roślin dla danego ogrodu oraz kontroler aktualizujący informacje o roślinie. Dla każdej wczytywanej rośliny w liście sprawdzany jest jej aktualny status wykonania prac. Jeśli cykl danego zadania minął, jego status się zmienia.

Listing 11 Sprawdzanie statusu nawodnienia rośliny

```
if (plant.getLastHydration().getTime() + plant.getHydrationCycle() -
curr.getTime() < 0 && plant.isHydrated()) {
    plant.setHydrated(false);
    plantRepository.save(plant);
}
```

Na przykładzie nawodnienia rośliny, sprawdzany jest warunek czy od ostatniego podlewania nie minął zbyt duży okres oraz czy roślina ma status podlanej. Jeśli warunki są spełnione, status wykonania pracy jest aktualizowany. Gdy użytkownik wykona daną pracę, pole zawierające datę jej wykonania jest aktualizowane:

Listing 12 Ustawianie nowego czasu wykonania pracy

```
if (plant.isHydrated() != plantRequest.isHydrated()) {
    plant.setHydrated(plantRequest.isHydrated());
    plant.setLastHydration(new Date());
}
```

Konstruktor `new Date()` domyślnie zwraca aktualną datę i godzinę.

## 4.6. Cykliczny e-mail z listą ogrodów wymagających uwagi

W związku z dużym ryzykiem, że użytkownik może zapomnieć o sprawdzeniu statusu zadań w aplikacji, zaimplementowana została funkcjonalność, która cyklicznie wysyła do użytkownika e-mail z przypomnieniem, w których ogrodach ma oczekujące zadania. Celowo nie jest wysyłana dokładna lista roślin, żeby zachęcić użytkownika do uruchomienia aplikacji.



lukasz@broll.pl

lukasz@broll.pl

Przypomnienie o pracy w ogrodzie

Cześć!

W następujących ogrodach masz oczekujące zadania do wykonania:

[Kuchnia, Salon, Ogród, Ganek]

Zadbaj o rośliny i nie zapomnij zaktualizować statusu w aplikacji!

Rysunek 8 Wiadomość przypominająca o pracy w ogrodzie

Do implementacji rozwiązania wykorzystywano wbudowaną w Springa obsługę zadań cyklicznych, która za pomocą adnotacji `@EnableScheduling` oraz `@Scheduled(fixedRate = 21600000)`, wymuszają na aplikacji serwerowej wykonanie metody. Parametr `fixedRate` to cykl wykonywania metody w milisekundach. Cykl w aplikacji został ustalony na 6 godzin (21600000 milisekund), ponieważ tyle wynosi najmniejszy możliwy cykl pracy nad rośliną. Z bazy danych pobierana jest lista wszystkich ogrodów i dla każdego sprawdzane są rośliny pod względem statusu zadań:

Listing 13 Metoda sprawdzająca statusy prac roślin w ogrodzie

```
public boolean pendingJobs(Garden garden){
    AtomicBoolean isJob = new AtomicBoolean(false);
    List<Plant> plants =
plantRepository.findByGardenId(garden.getId());
    plants.forEach(plant -> {
        if(!plant.isHydrated() && plant.getHydrationCycle()!=0)
isJob.set(true);
        else if(!plant.isMedicine() && plant.getMedicineCycle()!=0)
isJob.set(true);
        else if(!plant.isExaggerated() &&
plant.getExaggerationCycle()!=0) isJob.set(true);
        else if(!plant.isFertilized() &&
plant.getFertilizationCycle()!=0) isJob.set(true);
    });
    return isJob.get();    }
```

Jeśli dla któregoś z typów zadania wymaga jest akcja, metoda zwraca `isJob=true`. Celem uniknięcia prac, które nie są wykonywane dla rośliny, sprawdzany jest dodatkowo cykl wykonania, który dla opcji ‘nie dotyczy’ wynosi 0.

W głównej metodzie, wszystkie ogrody wymagające uwagi są kolekcjonowane. Następnie w kolekcji typu Set zapisywane są adresy e-mail właścicieli ogrodów. Kolekcję Set wykorzystano, żeby uniknąć zduplikowanych rekordów. Na końcu do każdego użytkownika wysyłany jest e-mail:

Listing 14 Metoda wysyłająca e-maile o oczekujących zadaniach w ogrodzie

```
@Scheduled(fixedRate = 21600000) //6h
public void scheduleFixedRateTask() {
    List<Garden> gardens = gardenRepository.findAll();
    List<String> jobGardens = new ArrayList<>();
    Set<String> owners = new HashSet<>();
    gardens.forEach(garden -> {
        if(pendingJobs(garden)) {jobGardens.add(garden.getName());
            owners.add(userService.findOne(garden.getUsername()).getEmail());
        }
    });
    owners.forEach(owner ->{
        SimpleMailMessage toDoListEmail = new SimpleMailMessage();
        toDoListEmail.setTo(owner);
        toDoListEmail.setSubject("Przypomnienie o pracy w ogrodzie");
        toDoListEmail.setText("Cześć!\nW następujących ogrodach masz
oczekujące zadania do wykonania:\n"
+ jobGardens.toString()+
"\nZadbaj o rośliny i nie zapomnij zaktualizować statusu w aplikacji!");
        toDoListEmail.setFrom("lukasz@broll.pl");
        emailService.sendEmail(toDoListEmail);
    });
}
```

## 4.7. Czat ogrodników

W celu wymiany informacji między ogrodnikami, zaimplementowany został wspólny pokój czatu dla wszystkich użytkowników aplikacji. Użytkownicy są identyfikowani po nazwie użytkownika podanej przy rejestracji, a razem z wiadomością przesyłany jest czas jej wysłania.

Do implementacji rozwiązania na serwerze wykorzystano zbiór bibliotek Spring Boot dla biblioteki WebSockets - spring-boot-starter-websocket. Aplikacja kliencka komunikuje się z serwerem czatu za pomocą SockJS[8], biblioteki JavaScript, która operuje na obiektach zgodnych z WebSockets. Konfiguracja serwera wiadomości wymaga zdefiniowania adresu, na którym ma działać oraz adresu, na którym będzie otrzymywać wiadomości z czatu.

Listing 15 Metoda konfiguracyjna adres serwera czatu

```
public void registerStompEndpoints(StompEndpointRegistry registry)
{
    registry.addEndpoint("/webchat")
        .setAllowedOrigins("*")
        .withSockJS();
}
```

W pierwszej kolejności definiowany jest adres serwera, następnie reguła zabezpieczeń umożliwiająca kontakt z serwerem z dowolnego adresu, a na końcu informacja o wykorzystaniu SockJS do komunikacji z serwerem.

Listing 16 Metoda konfiguracyjna brokera wiadomości czatu

```
public void configureMessageBroker(MessageBrokerRegistry registry)
{
    registry.setApplicationDestinationPrefixes("/app")
        .enableSimpleBroker("/chat");
}
```

W konfiguracji brokera wiadomości, wystarczy zdefiniować tylko adres, na który wysyłane będą wiadomości.

## 5. Testy

Przeprowadzone zostały podstawowe testy aplikacji pod kątem poprawnego działania modeli, kontrolerów oraz testy funkcjonalne podstawowych funkcji aplikacji. Testy modeli zostały utworzone za pomocą narzędzia JUnit, testy kontrolerów za pomocą aplikacji Postman, a testy funkcjonalne zostały wykonane na aplikacji z wykorzystaniem przeglądarki Google Chrome.

**JUnit**[11] jest to popularne narzędzie do tworzenia testów jednostkowych w języku Java. Jego główne cechy to m.in. oddzielenie testów od kodu, tworzenie raportów oraz wiele mechanizmów uruchamiania.

**Postman**[7] to aplikacja komputerowa do testowania API, tj. interfejsu programowania aplikacji, czyli sposobu w jaki komunikują się programy. Symuluje zapytania, które aplikacja kliencka wysyła do aplikacji serwerowej protokołem http i wyświetla wynik zapytania. Umożliwia to sprawdzenia poprawności działania kontrolerów, bez zaimplementowanego interfejsu użytkownika.

### 5.1. Testy jednostkowe modeli

Przetestowane zostały najważniejsze modele w aplikacji, których dane przechowywane są w bazie danych – User, Garden oraz Plant. Podstawowy test sprawdza, czy dane są prawidłowo zapisywane do pól i z nich odczytywane.

Listing 17 Test jednostkowy pola Id klasy User

```
@Test
public void setId() {
    Long id = (long)1;
    user.setId(id);
    assertEquals(user.getId(), id);
}
```

Bardziej zaawansowany test dla listy roślin w ogrodzie, sprawdza jednocześnie parametry rośliny oraz prawidłowe przypisanie do listy.

Listing 18 Test jednostkowy listy roślin w ogrodzie

```
@Test
public void setPlantsList() {
    Plant plant1 = Plant.builder()
        .id((long)1)
        .name("Róża")
        .description("Czerwona")
        .notes("od babci")
        .isHydrated(true)
        .hydrationCycle((long)10000)
```

```
        .lastHydration(firstDate)
        .garden(garden)
        .gardenId((long) 1).build();
Plant plant2 = Plant.builder()
    .id((long) 2)
    .name("Tulipan")
    .description("Żółty")
    .notes("")
    .isHydrated(true)
    .hydrationCycle((long) 20000)
    .lastHydration(secondDate)
    .garden(garden)
    .gardenId((long) 1).build();

List<Plant> plants = new ArrayList<>();
plants.add(plant1);
plants.add(plant2);
garden.setPlantsList(plants);
assertEquals(garden.getPlantsList(), plants);
}
```

Metoda `assertEquals` porównuje przekazane jej argumenty. Test jest pozytywny, jeśli oba argumenty mają taką samą zawartość.

## 5.2. Testy funkcjonalne

Testy funkcjonalne polegają na mniej technicznym podejściu do aplikacji. Tester na podstawie scenariusza wykonuje czynności z poziomu aplikacji i sprawdza czy oczekiwany efekt kroku został spełniony. Przeprowadzone scenariusze uwzględniały:

- logowanie z prawidłowymi danymi, fałszywymi oraz dla nieistniejącego konta,
- rejestrację z walidacją pól, aktywację konta i próbę dostępu do zasobów aplikacji przed aktywowaniem konta,
- dodawanie, edytowanie, usuwanie ogrodów i roślin z walidacją pól wymaganych i niewymaganych,
- korzystanie z czatu,
- korzystanie z listy zadań.

### 5.3. Testy kontrolerów

Poniżej zaprezentowano test kontrolera odpowiedzialnego za aktywację konta za pomocą aplikacji Postman. Na adres `/confirm-acc/{token}` wysyłany jest kod uwierzytelniający, a zadaniem kontrolera jest znalezienie użytkownika na podstawie kodu, jego aktywacja oraz zwrócenie obiektu z danymi w postaci JSON.

The screenshot shows a Postman interface for a GET request to `http://localhost:8080/confirm-acc/637a42b8-0c5e-4c2a-b9ff-02b08f45b895`. The response status is 200 OK, with a time of 319 ms and a size of 548 B. The response body is displayed in JSON format:

```

1 {
2   "id": 34,
3   "name": "lukluk",
4   "email": "m3046895@nwytg.net",
5   "username": "lukluk",
6   "password": "$2a$10$KE4FI4aG2ahTrzvzjbYa40n7F8o2zZtIYmPDvEiPuX0NFAKfp0tEy",
7   "role": "USER",
8   "enabled": true,
9   "confirmationToken": "637a42b8-0c5e-4c2a-b9ff-02b08f45b895"
10 }
```

Rysunek 9 Test API kontrolera aktywacji konta

## 6. Aplikacja

Wykorzystanie biblioteki Angular Material umożliwiło utworzenie funkcjonalnego i nowoczesnego interfejsu z responsywnymi tabelami ogrodów i roślin oraz okna czatu. Poniżej przedstawiono ekran główny aplikacji, zawierający krótki tekst o roślinach. Dla niezalogowanego użytkownika w menu możliwa jest wyłącznie opcja logowania. Dla zalogowanego użytkownika możliwe są opcje czatu, listy zadań, menadżera ogrodów i wylogowania. Wykorzystane tło<sup>1</sup> aplikacji udostępnione jest na darmowej licencji Creative Commons.

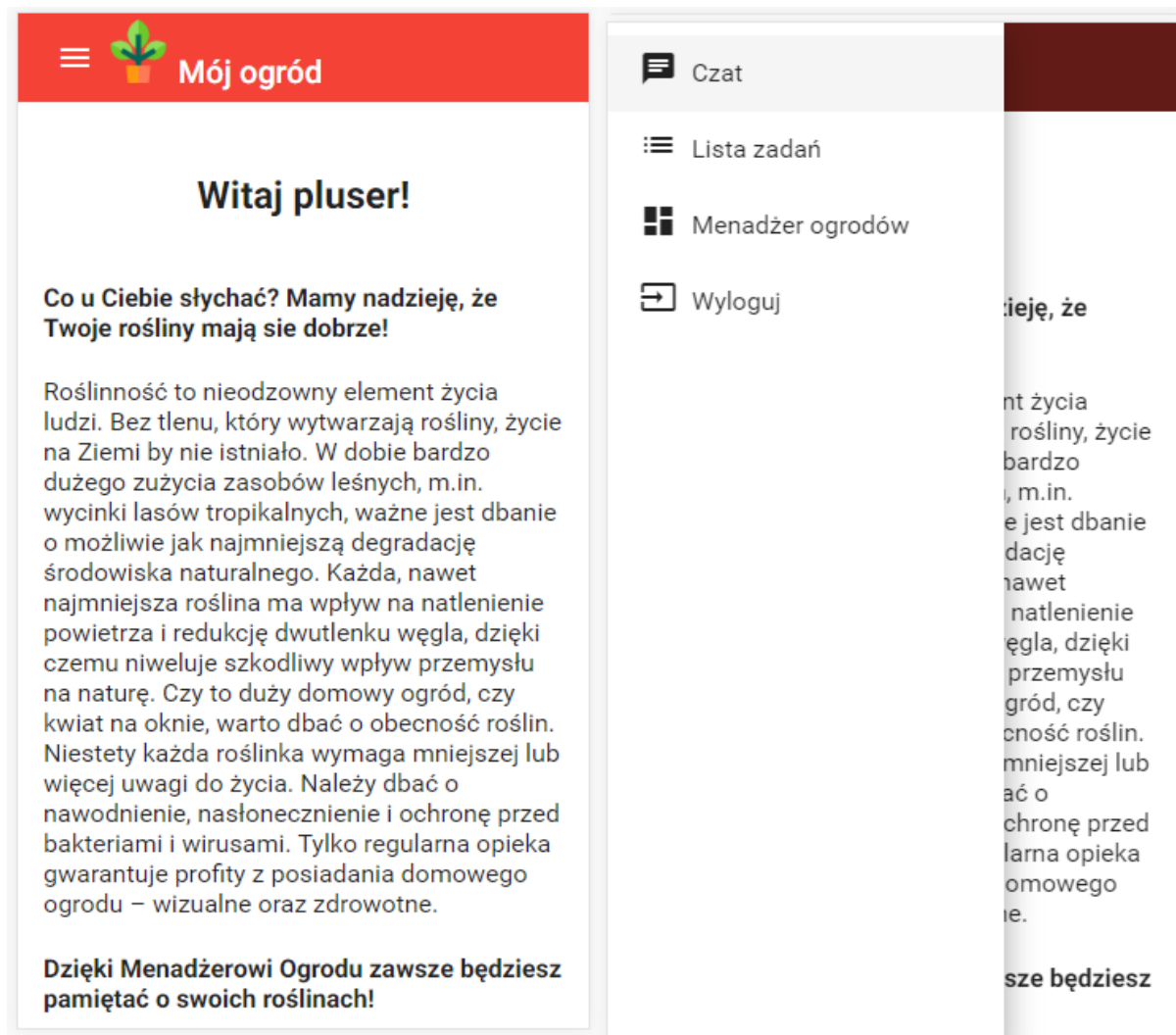


Rysunek 10 Ekran główny

<sup>1</sup> Źródło tła: <https://pixabay.com/pl/chabry-polne-kwiaty-%C5%82%C4%85ka-3432162>

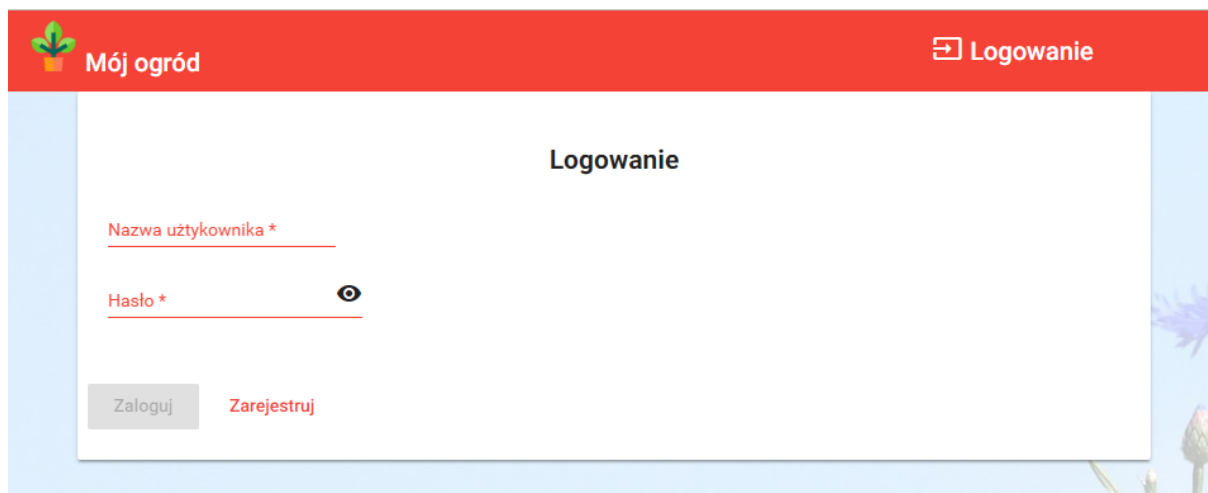


Do implementacji menu wykorzystano elementy `mat-sidenav`, `mat-toolbar` oraz `fxLayout`. W zależności od rozmiaru ekranu, menu jest dostępne w postaci wysuwanego panelu bocznego dla ekranów mobilnych lub belki na górze strony dla ekranów o rozdzielczości większej niż 960 pikseli.



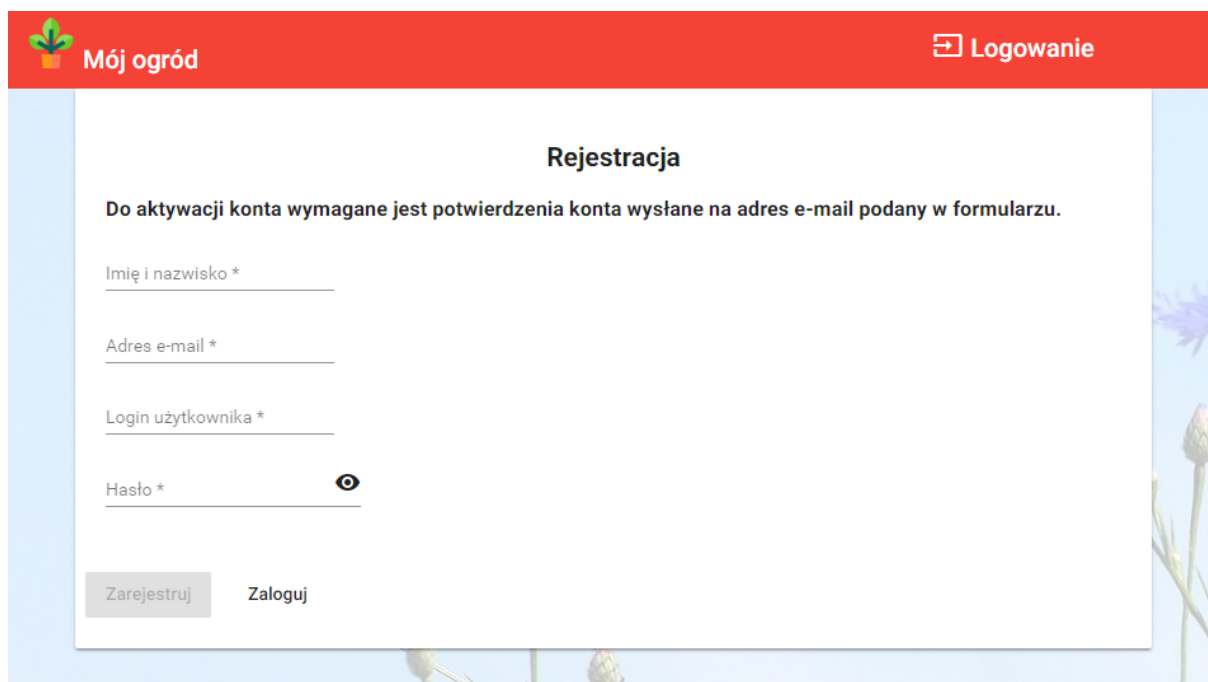
Rysunek 11 Strona główna z menu bocznym na ekranie urządzenia mobilnego

Panel logowania składa się z pola nazwy użytkownika oraz hasła. Do utworzenia formularza wykorzystano elementy `mat-input`, `mat-form-field` oraz `mat-button`. Użytkownik ma możliwość podejrzenia wprowadzanego hasła, a przycisk logowania jest zablokowany do czasu wprowadzenia minimalnej ilości znaków w obu polach. Z ekranu logowania możliwe jest przejście do ekranu rejestracji. Po pomyślnym zalogowaniu, użytkownik jest przekierowany do ekranu głównego aplikacji oraz odblokowane są wszystkie opcje w menu, w przeciwnym razie wyświetlany jest komunikat o nieudanym logowaniu.



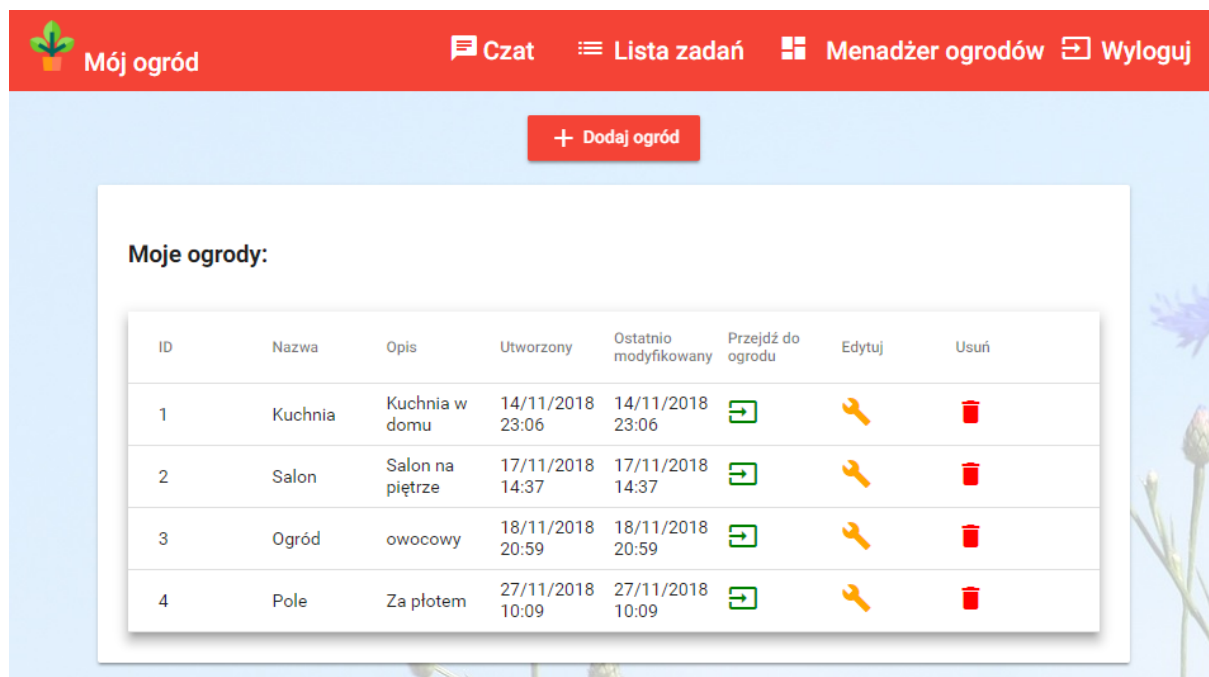
Rysunek 12 Ekran logowania

Na ekranie rejestracji wykorzystano te same elementy, co na ekranie logowania. Przycisk Zarejestruj uaktywnia się po wypełnieniu prawidłowo wszystkich pól. Pole adresu e-mail musi zawierać znak '@', a inne pola muszą mieć minimalną długość 6 znaków.



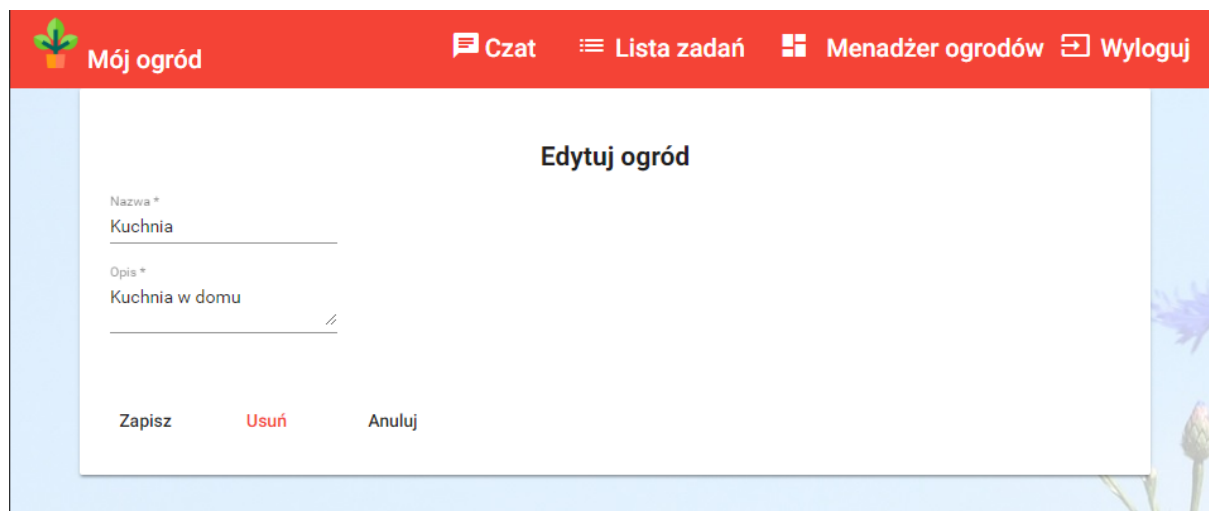
Rysunek 13 Ekran rejestracji

Do implementacji ekranu listy ogrodów wykorzystano bibliotekę `fxLayout`, która umożliwia tworzenie responsywnych, dostosowanych do każdego rozmiaru ekranu tabel. W tabeli ogrodów użytkownika wyświetlane są wszystkie możliwe dane i akcje związane z ogrodem, zgodnie z opisami kolumn.



Rysunek 14 Ekran listy ogrodów

Ekran edycji ogrodu wykorzystuje te same elementy biblioteki Angular Material, co ekran logowania. W zależności czy wybrano edycję istniejącego ogrodu, czy tworzenie nowego, dostosowany jest nagłówek ekranu oraz możliwość usuwania ogrodu.

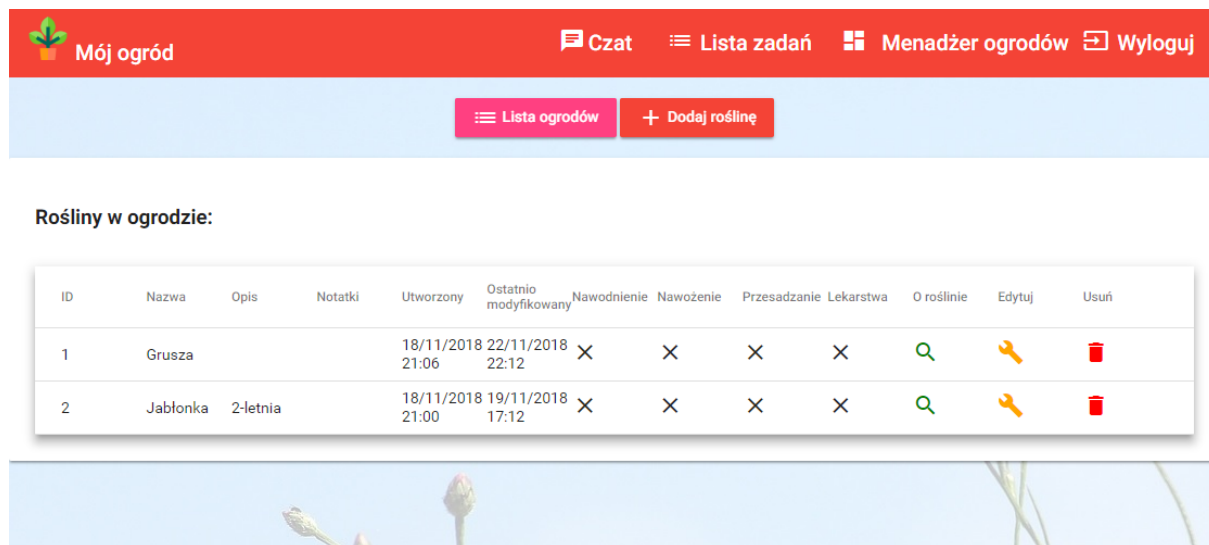


Rysunek 15 Ekran edycji lub dodawania ogrodu

Ekran listy ogrodów to główna funkcjonalność całej aplikacji. W tym miejscu wyświetlane są najważniejsze informacje dotyczące statusu prac związanych z rośliną, informacje o utworzeniu i ostatniej dacie edycji, a także możliwe akcje do wykonania. Przycisk zielonej lupy wyszukuje w wyszukiwarce Google informacje o roślinie na podstawie jej nazwy.

Aplikacja webowa umożliwiająca zarządzanie domowym ogrodem

Kolejne przyciski umożliwiają odpowiednio edytowanie oraz usuwanie rośliny. Tabela jest responsywna dzięki wykorzystaniu biblioteki `fxLayout`.

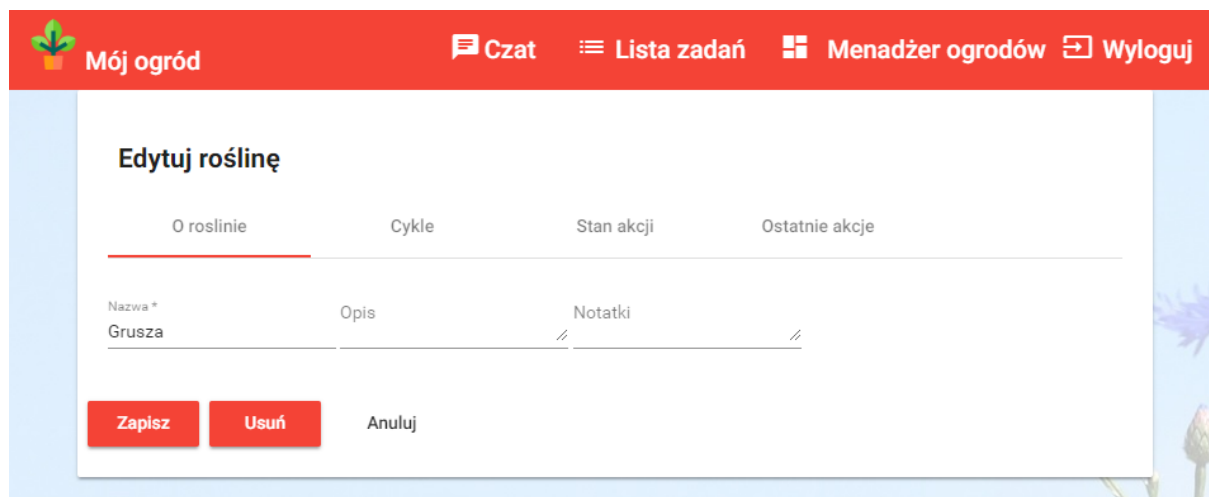


The screenshot shows the 'Mój ogród' (My Garden) application interface. At the top, there is a red navigation bar with a plant icon and the text 'Mój ogród'. To the right of the bar are links: 'Czat', 'Lista zadań', 'Menadżer ogrodów', and 'Wyloguj'. Below the navigation bar, there is a light blue section with two buttons: 'Lista ogrodów' and '+ Dodaj roślinę'. The main content area is titled 'Rośliny w ogrodzie:' and contains a table with two rows of plant data. The table has columns for ID, Nazwa, Opis, Notatki, Utworzony, Ostatnio modyfikowany, Nawodnienie, Nawożenie, Przesadzanie, Lekarstwa, O roślinie, Edytuj, and Usuń. The first row shows a plant with ID 1, named 'Grusza', with a creation date of 18/11/2018 21:06 and a last modification date of 22/11/2018 22:12. The second row shows a plant with ID 2, named 'Jabłonka', with a creation date of 18/11/2018 21:00 and a last modification date of 19/11/2018 17:12. The table is followed by a decorative image of a plant.

ID	Nazwa	Opis	Notatki	Utworzony	Ostatnio modyfikowany	Nawodnienie	Nawożenie	Przesadzanie	Lekarstwa	O roślinie	Edytuj	Usuń
1	Grusza			18/11/2018 21:06	22/11/2018 22:12	×	×	×	×	🔍	🔧	🗑️
2	Jabłonka	2-letnia		18/11/2018 21:00	19/11/2018 17:12	×	×	×	×	🔍	🔧	🗑️

Rysunek 16 Ekran listy roślin w ogrodzie

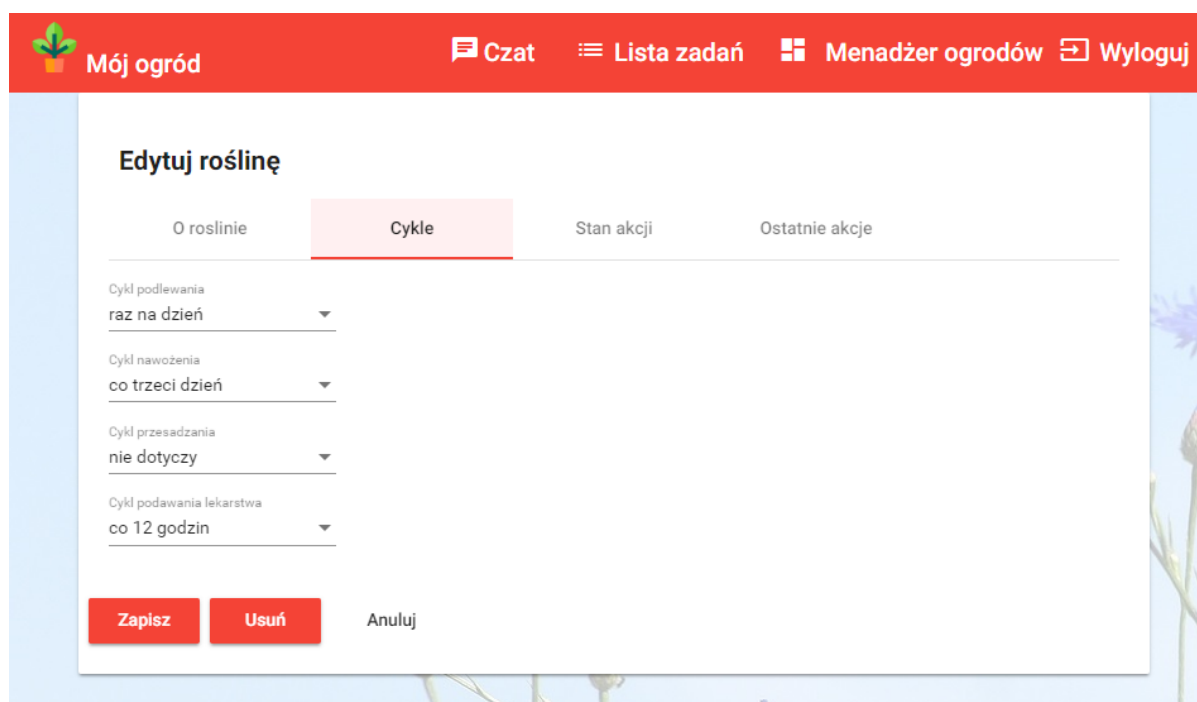
Ekran edycji informacji składa się z czterech paneli utworzonych za pomocą elementów `mat-tab-group` oraz `mat-tab`. Na pierwszym, zatytułowanym 'O roślinie' można edytować nazwę, opis oraz notatki za pomocą elementów `mat-input` oraz `textarea`.



The screenshot shows the 'Edytuj roślinę' (Edit Plant) application interface. At the top, there is a red navigation bar with a plant icon and the text 'Mój ogród'. To the right of the bar are links: 'Czat', 'Lista zadań', 'Menadżer ogrodów', and 'Wyloguj'. Below the navigation bar, there is a light blue section with a title 'Edytuj roślinę' and four tabs: 'O roślinie', 'Cykle', 'Stan akcji', and 'Ostatnie akcje'. The 'O roślinie' tab is selected and shows a form with three input fields: 'Nazwa \*' (Name), 'Opis' (Description), and 'Notatki' (Notes). The 'Nazwa \*' field contains the text 'Grusza'. Below the input fields, there are three buttons: 'Zapisz' (Save), 'Usuń' (Delete), and 'Anuluj' (Cancel).

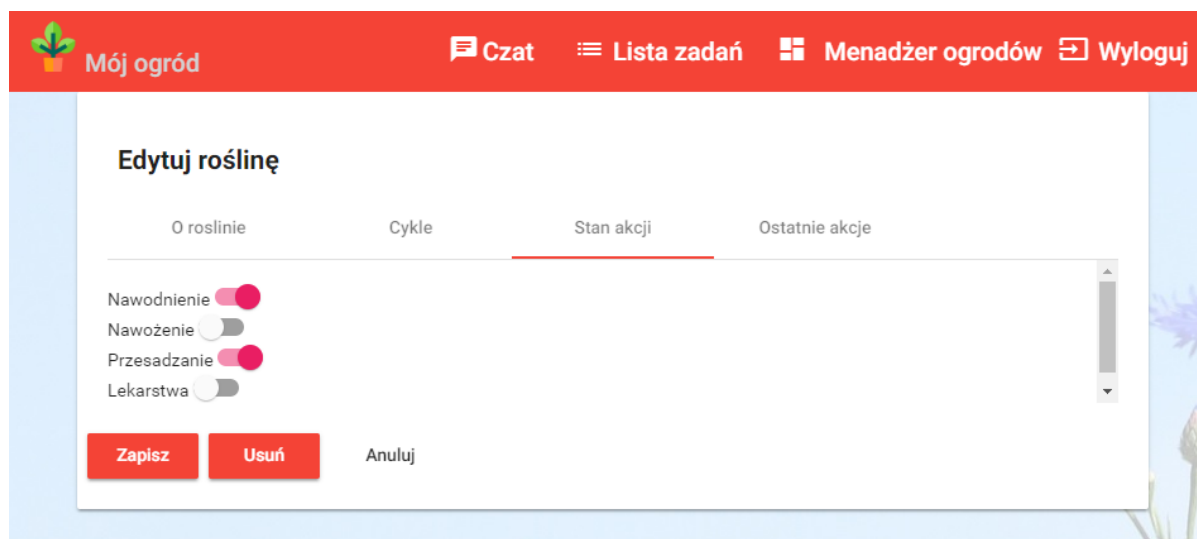
Rysunek 17 Ekran edycji podstawowych informacji o roślinie

W panelu 'Cykle' za pomocą `mat-select` i `mat-option` możliwy jest wybór cyklu wykonywania danej czynności. Z rozwijanej listy elementów do wyboru jest kilkanaście możliwych terminów – od sześciu godzin do trzech lat.



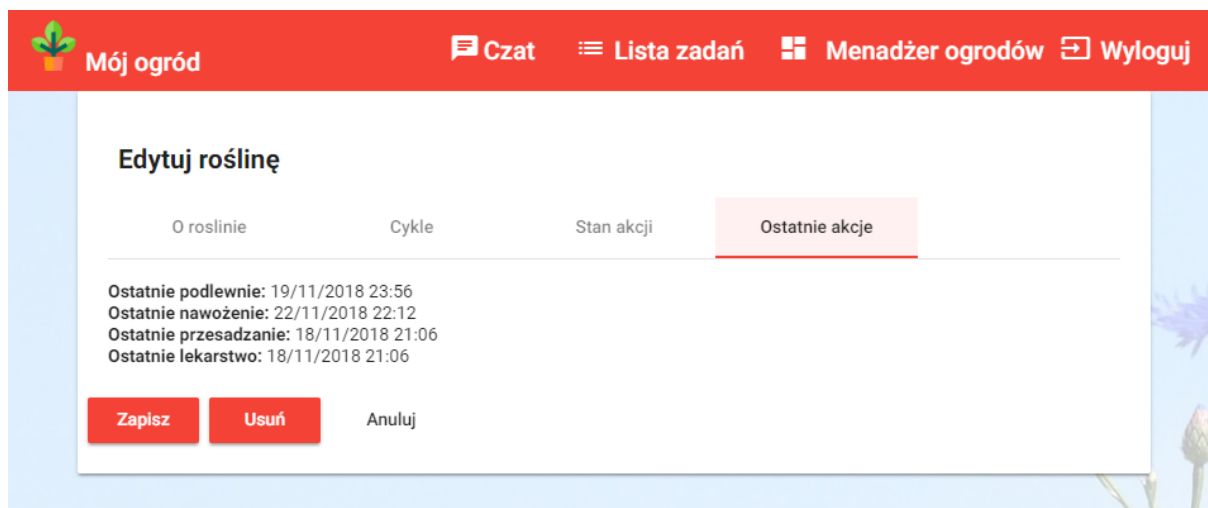
Rysunek 18 Ekran edycji cykli prac rośliny

Na trzecim panelu możliwe jest ustawianie statusu wykonania czynności za pomocą `mat-slide-toggle`. Jeśli użytkownik ustawi wskaźnik w prawo i zapisze edycję, czas cyklu wykonania zostanie zresetowany do aktualnej daty i godziny.



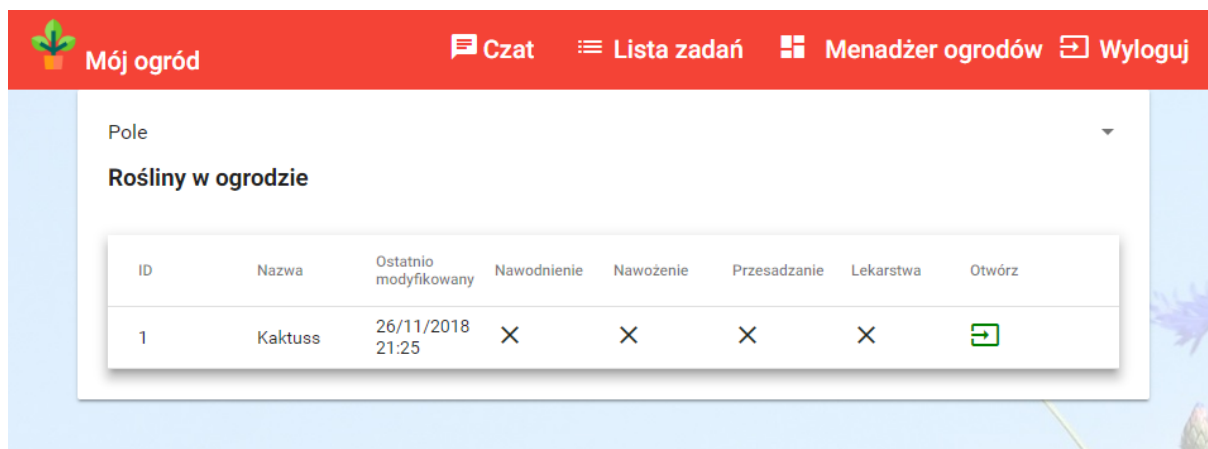
Rysunek 19 Ekran edycji stanów akcji rośliny

Na ostatnim panelu edycji rośliny widoczne są czasy ostatniego wykonania danej czynności, na podstawie których można określić, kiedy dokładnie należy wykonać akcję ponownie.



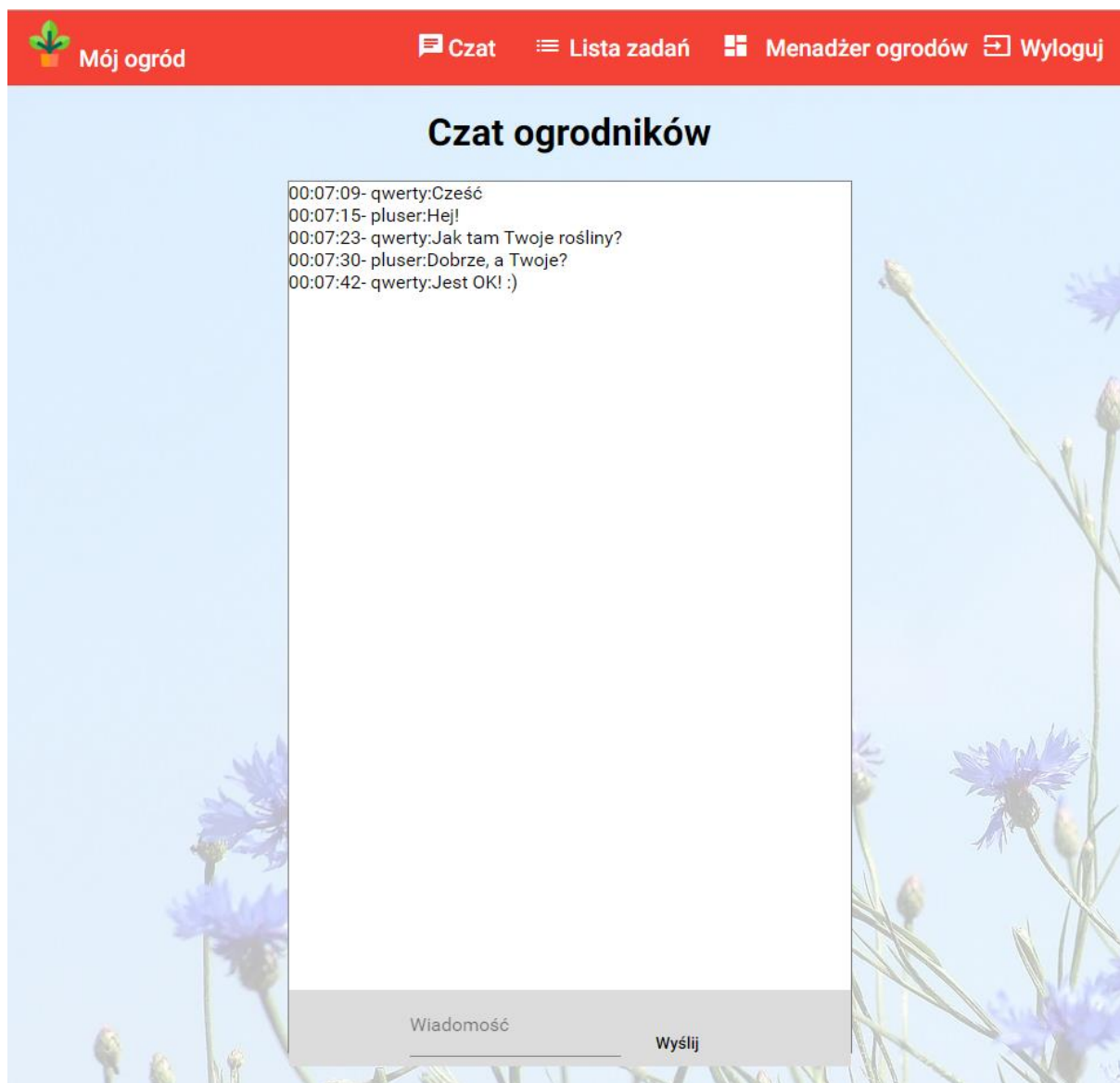
Rysunek 20 Ekran statusu ostatnich akcji rośliny

Komponent listy zadań został zaimplementowany, żeby uprościć aktualizowanie akcji dla danej rośliny. Z rozwijanej listy użytkownik wybiera jeden ze swoich ogrodów, a tabela z roślinami aktualizuje się bez odświeżania strony. W tabeli wyświetlane są tylko informacje o statusach wykonania czynności oraz przycisk przekierowujący do edycji danej rośliny.



Rysunek 21 Ekran listy zadań ogrodu

Ekran Czat składa się z okna wiadomości, pola do wprowadzania tekstu oraz przycisku wysyłania wiadomości. Użytkownik w momencie uruchomienia czatu nie ma dostępu do historii rozmów. Dla każdej wiadomości wyświetlany jest kolejno czas jej wysłania, autor oraz treść.



Rysunek 22 Ekran czatu

## 7. Podsumowanie

Efektem pracy jest aplikacja spełniająca wymagania funkcjonalne i нефункционалне projektu. Dodatkową, nieplanowaną funkcjonalnością jest pokój czatu dla ogrodników. Aplikacja nadaje się do realnego kontrolowania prac związanych z ogrodem, komunikacji między użytkownikami za pomocą czatu oraz wysyłania przypomnień na adres e-mail.

Jednym z trudniejszych problemów w trakcie implementacji, było poprawne skonfigurowanie zabezpieczeń JSON Web Token z dostępem do zasobów, a szczególnie z biblioteką WebSockets obsługującą chat, która wymagała indywidualnego sposobu przesyłania tokenu zabezpieczeń. Jestem zadowolony, że zaimplementowane zostały wszystkie zakładane funkcjonalności oraz że większość widoków wyświetla się poprawnie na ekranach urządzeń mobilnych. Minusem aplikacji jest to, że widoki edycji roślin oraz listy zadań nie są responsywne. Technologie, które zostały wykorzystane w projekcie dają satysfakcjonujące efekty i wykorzystałbym je ponownie, gdybym implementował aplikację jeszcze raz.

Możliwe kierunki rozwoju aplikacji:

- Utworzenie bazy roślin zawierającej informacje i odpowiedzi na temat ich pielęgnacji,
- Rozszerzony system bezpośredniej komunikacji między użytkownikami,
- Wizualne projektowanie rozmieszczenia roślin w ogrodzie,
- Możliwość edycji danych użytkownika,
- Wspólne zarządzanie ogrodem przez wielu użytkowników.



# Spis rysunków

Rysunek 1 Architektura systemu.....	8
Rysunek 2 Kodowanie tokenu zabezpieczeń .....	11
Rysunek 3 Diagram przypadków użycia.....	13
Rysunek 4 Diagram encji .....	14
Rysunek 5 Diagram klas modelu .....	15
Rysunek 6 Diagram pakietów aplikacji serwerowej .....	17
Rysunek 7 Email potwierdzający rejestrację konta.....	23
Rysunek 8 Wiadomość przypominająca o pracy w ogrodzie .....	26
Rysunek 9 Test API kontrolera aktywacji konta.....	31
Rysunek 10 Ekran główny .....	32
Rysunek 11 Strona główna z menu bocznym na ekranie urządzenia mobilnego .....	33
Rysunek 12 Ekran logowania.....	34
Rysunek 13 Ekran rejestracji.....	34
Rysunek 14 Ekran listy ogrodów .....	35
Rysunek 15 Ekran edycji lub dodawania ogrodu.....	35
Rysunek 16 Ekran listy roślin w ogrodzie .....	36
Rysunek 17 Ekran edycji podstawowych informacji o roślinie .....	36
Rysunek 18 Ekran edycji cykli prac rośliny .....	37
Rysunek 19 Ekran edycji stanów akcji rośliny .....	37
Rysunek 20 Ekran statusu ostatnich akcji rośliny .....	38
Rysunek 21 Ekran listy zadań ogrodu .....	38
Rysunek 22 Ekran czatu .....	39

# Spis listingów

Listing 1 Przykładowe adresowanie kontrolera .....	19
Listing 2 Konfiguracja adresowania w aplikacji klienckiej .....	21
Listing 3 Konfiguracja zabezpieczeń aplikacji .....	21
Listing 4 Konfiguracja tokenu zabezpieczającego .....	22
Listing 5 Metoda rejestracji nowego użytkownika .....	22
Listing 6 Implementacja powiadomienia o aktywacji konta.....	23
Listing 7 Metoda przechwytyjąca kod uwierzytelniający .....	24
Listing 8 Metoda potwierdzająca konto, komunikująca się z kontrolerem.....	24
Listing 9 Kontroler obsługujący aktywację konta.....	24
Listing 10 Metoda aktywująca konto bezpośrednio w bazie danych .....	25
Listing 11 Sprawdzanie statusu nawodnienia rośliny .....	25
Listing 12 Ustawianie nowego czasu wykonania pracy.....	25
Listing 13 Metoda sprawdzająca statusy prac roślin w ogrodzie .....	26
Listing 14 Metoda wysyłająca e-maile o oczekujących zadaniach w ogrodzie .....	27
Listing 15 Metoda konfiguracyjna adres serwera czatu .....	27
Listing 16 Metoda konfiguracyjna brokera wiadomości czatu .....	28
Listing 17 Test jednostkowy pola Id klasy User .....	29
Listing 18 Test jednostkowy listy roślin w ogrodzie .....	29

# Literatura

- [1] Seth Ladd, Keith Donald., Expert Spring MVC and Web Flows, apress, USA 2006
- [2] Felipe Gutierrez, "Wprowadzenie do Spring Framework dla programistów Java", Helion 2015
- [3] Nate Murray, Felipe Coury, Ari Lerner, Carlos Taborda, ng-book The Complete Book on Angular 5, Fullstack.io, San Francisco, California, 2018
- [4] Sebastian Peyrott. JWT Handbook, Auth0, 2017
- [5] Biblioteka Angular Material <https://material.angular.io/> [dostęp dnia 4 listopada 2018]
- [6] Dokumentacja Hibernate <https://hibernate.org/orm/documentation/5.3/> [dostęp dnia 3 października 2018]
- [7] Dokumentacja Postmana <https://www.getpostman.com/docs/v6/> [dostęp dnia 4 listopada 2018]
- [8] Dokumentacja SockJS <https://github.com/sockjs/sockjs-client> [dostęp dnia 20 listopada 2018]
- [9] Dokumentacja MySQL <https://dev.mysql.com/doc/> [dostęp dnia 3 października 2018]
- [10] Dokumentacja bcrpyt <https://www.npmjs.com/package/bcrypt> [dostęp dnia 8 października 2018]
- [11] Dokumentacja JUnit <https://junit.org/junit5/docs/current/user-guide/> [dostęp dnia 14 listopada 2018]