POLITECHNIKA WROCŁAWSKA

WYDZIAŁ ELEKTRONIKI

KIERUNEK: INFORMATYKA (INF)

SPECJALNOŚĆ: INŻYNIERIA SYSTEMÓW INFORMATYCZNYCH (INS)

PRACA DYPLOMOWA

INŻYNIERSKA

Aplikacja webowa umożliwiająca zarządzanie domowym ogrodem

A web application that allows you to manage your home garden

AUTOR:

Łukasz Broll

PROWADZĄCY PRACĘ:

Dr inż, Marek Woda, W4/K9

OCENA PRACY:

WROCŁAW, 2018

**Spis treści**

[1. Wstęp 4](#_Toc531079810)

[1.1. Wprowadzenie 4](#_Toc531079811)

[1.2. Cel pracy 4](#_Toc531079812)

[1.3. Zakres pracy 5](#_Toc531079813)

[1.4. Układ pracy 5](#_Toc531079814)

[2. Specyfikacja wymagań 6](#_Toc531079815)

[2.1. Wymagania funkcjonalne 6](#_Toc531079816)

[2.2. Wymagania niefunkcjonalne 7](#_Toc531079817)

[3. Projekt 8](#_Toc531079818)

[3.1. Architektura 8](#_Toc531079819)

[3.1.1. Wzorzec MVC 8](#_Toc531079820)

[3.1.2. Baza danych MySQL 9](#_Toc531079821)

[3.1.3. Aplikacja serwerowa Spring 9](#_Toc531079822)

[3.1.4. Aplikacja kliencka Angular 10](#_Toc531079823)

[3.2. Diagram encji 11](#_Toc531079824)

[3.2.1. Encja User 11](#_Toc531079825)

[3.2.2. Encja Garden 11](#_Toc531079826)

[3.2.3. Encja Plant 12](#_Toc531079827)

[3.3. Diagram klas 12](#_Toc531079828)

[3.4. Narzędzia 13](#_Toc531079829)

[4. Implementacja 14](#_Toc531079830)

[4.1. Pakiety 14](#_Toc531079831)

[4.2. Adresowanie 15](#_Toc531079832)

[4.2.1. Adresowanie w aplikacji 16](#_Toc531079833)

[4.3. Zabezpieczenia 17](#_Toc531079834)

[4.3.1. Autoryzacja konta przez e-mail 17](#_Toc531079835)

[4.4. Automatyczne sprawdzanie statusu pracy 19](#_Toc531079836)

[4.5. Cykliczny e-mail z listą ogrodów wymagających uwagi 20](#_Toc531079837)

[4.6. Czat ogrodników 21](#_Toc531079838)

[5. Testy 22](#_Toc531079839)

[6. Aplikacja 22](#_Toc531079840)

[7. Podsumowanie 22](#_Toc531079841)

[8. Sekcja 22](#_Toc531079842)

[8.1. Sekcja poziomu 1 22](#_Toc531079843)

[8.1.1. Sekcja poziomu 2 22](#_Toc531079844)

[8.2. Kolejna sekcja poziomu 1 23](#_Toc531079845)

[8.2.1. Kolejna sekcja poziomu 2 23](#_Toc531079846)

[9. Podsumowanie i wnioski 24](#_Toc531079847)

[Literatura 25](#_Toc531079848)

[Spis rysunków 26](#_Toc531079849)

[Spis tabel 27](#_Toc531079850)

[Spis listingów 28](#_Toc531079851)

[Skróty 29](#_Toc531079852)

[Dodatek A 30](#_Toc531079853)

1. Wstęp
   1. Wprowadzenie

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 roślinka wymaga mniejszej lub więcej 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. Cel pracy

Celem pracy jest utworzenie narzędzia wspomagającego zarządzanie domowym ogrodem, które wspomagać będzie regularne prace i obowiązki do utrzymania zdrowia roślin. Dzięki przechowywaniu w jednym miejscu informacji o wszystkich swoich roślinach i ogrodach, łatwiej będzie kontrolować bieżące obowiązki, a zautomatyzowane 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 program będzie responsywny, tj. będzie poprawnie skalować się na każdym typie urządzenia i rozmiarze ekranu, obsługa będzie intuicyjna, a interfejs nowoczesny.

* 1. Zakres pracy

Początkowym etapem pracy będzie specyfikacja wymagań funkcjonalnych oraz niefunkcjonalnych 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. Układ pracy

W kolejnych rozdziałach umieszczone zostaną konkretne treści. W Rozdziale 5 będzie ...

1. Specyfikacja wymagań

W rozdziale opisane są wymagania, jakie musi spełniać aplikacja do zarządzani domowym ogrodem. Podstawowym wymaganiem jest udostępnienie użytkownikami aplikacji do zarządzania domowym ogrodem, która będzie dostępna w języku polskim. Poniżej wymienione zostały wymagania funkcjonalne oraz niefunkcjonalne.

* 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 email 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.
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 informacje o utworzeniu i ostatniej edycji ogrodu.
5. W ogrodzie przechowywana jest lista roślin, każda roślina składa się na:
   1. nazwę,
   2. opis,
   3. notatki,
   4. prace związane z rośliną, tj. podlewanie, nawożenie, przesadzanie, dawkowanie środków ochrony roślin – w skład każdej pracy wchodzi:
      1. cykl wykonywania czynności,
      2. ostatni termin jej wykonania,
      3. aktualny status czynności.
   5. data utworzenia oraz ostatniej modyfikacji rośliny,
6. Wszystkie dane rośliny, oprócz ostatniego wykonywania każdej z czynności, można edytować lub usuwać roślinę oraz każde pole w formularzu jest walidowane przed zapisaniem.
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. Dla każdej rośliny zdefiniowany jest terminarz prac oraz w postaci listy zadań, w której znajdują się aktualne stany prac.
11. Do użytkownika cyklicznie wysyłany jest e-mail z listą ogrodów w których są oczekujące prace.
    1. Wymagania niefunkcjonalne
12. Aplikacja powinna działać poprawnie na poniższych przeglądarkach:

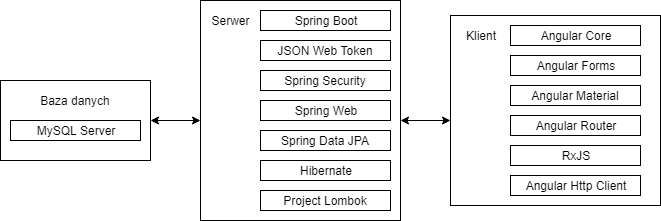
* Google Chrome w wersji 60 lub wyższej,
* Mozilla Firefox w wersji 54 lub wyższej,
* Microsoft Edge w wersji 14 lub wyższej,

1. Technologie oraz rozwiązania systemów informatycznych, wykorzystane do utworzenia aplikacji, powinny być darmowe.
2. Aplikacja powinna działać, niezależnie od sprzętu i systemu operacyjnego użytkownika.
3. Obsługa aplikacji powinna być intuicyjna i wygodna dla każdego użytkownika.
4. Projekt

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

* 1. Architektura

Aplikacja została podzielona na trzy niezależne od siebie moduły: bazę danych, serwer oraz aplikację kliencką. Dzięki takiemu rozwiązaniu łatwiej jest kontrolować działanie całego systemu oraz modyfikacje nad poszczególnymi komponentami. W związku z tym, schemat systemu razem z wykorzystanymi technologiami, wygląda następująco:



Rysunek 1 Architektura systemu

* + 1. Wzorzec MVC

Rozwinięciem skrótu MVC to Model-View-Controler, czyli Model-Widok-Kontroler. 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 opisywanej aplikacji, w formacie JSON i aktualizuje wyświetlane komponenty.
    1. Baza danych MySQL

MySQL to jeden z najpopularniejszych relacyjnych systemów zarządzania bazą danych. Został wykorzystany w projekcie, ponieważ bardzo dobrze współpracuje z wykorzystanymi technologiami w aplikacji serwerowej, jest łatwy do skonfigurowania oraz umożliwia bezproblemowe przeniesienie całej bazy na np. zewnętrzny serwer.

* + 1. Aplikacja serwerowa Spring

Spring 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 programisty. M.in. usprawnia obsługiwanie wyjątków, komunikację z bazą danych oraz umożliwia wstrzykiwanie zależności. W projekcie, w ramach Springa, wykorzystane zostały następujące moduły:

1. Spring Boot – narzędzie umożliwiające automatyzację konfiguracji oraz instalacji potrzebnych modułów.
2. Spring Security – biblioteka do zarządzania bezpieczeństwem aplikacji, chroni zasoby przed nieautoryzowanym dostępem.
3. Spring Web – biblioteka dostarczająca zbiór funkcjonalności usprawniających tworzenie aplikacji webowych.
4. 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.
   * + 1. Project Lombok

Biblioteka ograniczająca generowanie wielu linijek 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).

* + - 1. JSON Web Token

Standard generowania tokenów uwierzytelniających sesje użytkownika. 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. Wszystkie dane wewnątrz tokena są zakodowane algorytmem Base64. 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.

* + 1. Aplikacja kliencka Angular

Angular to framework autorstwa Google, który ma spełniać podobne zadanie co Spring, jednak przeznaczony dla języka TypeScript. We wzorcu MVC odpowiada za Widok, czyli to co widzi użytkownik. 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 logi napisanej w języku TypeScript. Wiele metod i funkcji jest już zdefiniowanych w szkielecie 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 frameworka.
      2. Angular Forms – biblioteka dostarczająca zestaw metod i funkcjonalności usprawniających tworzenie formularzy, ich walidację oraz wysyłanie.
      3. Angular Material – biblioteka zawierająca komponenty wizualne stylu Material Design, który polega na dostarczeniu prostego, czytelnego interfejsu dostosowanego do różnych urządzeń.
      4. Angular Router – biblioteka usprawniająca przekierowania wewnątrz aplikacji, wspierająca m.in. parametry 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.
  1. Diagram encji

Diagram przestawia encje bazy danych dla modelu klas w aplikacji.



Rysunek 2 Diagram encji

* + 1. Encja User

User, czyli użytkownik jest identyfikowany po polu username (nazwa użytkownika). Każdy użytkownik ma unikalny ‘username’ oraz ‘eamil’. Hasło użytkownika tj. pole ‘password’ jest zakodowane za pomocą funkcji bcrypt. Domyślną rolą każdego użytkownika jest „ROLE\_USER”. Użytkownik może posiadać nieograniczoną liczbę ogrodów.

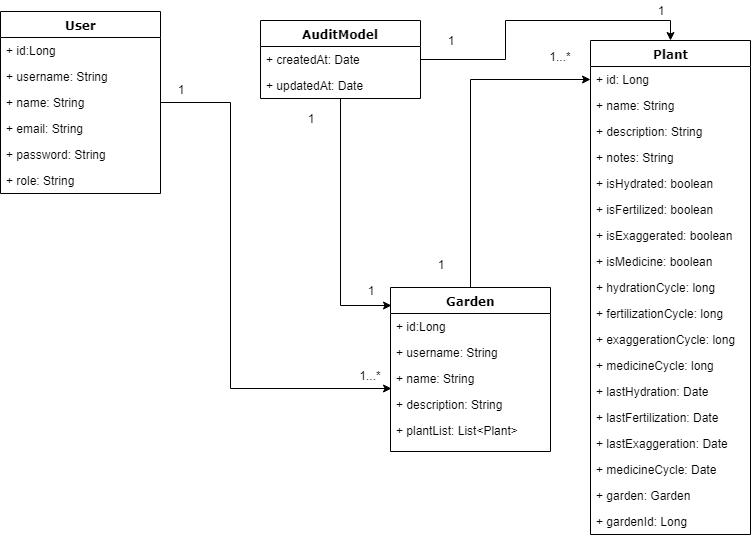
* + 1. Encja Garden

Garden, czyli ogród jest identyfikowany po 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.

* + 1. 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.

* 1. Diagram klas

Na diagramie nie uwzględniono konstruktorów klas, getterów oraz seterów pól, które są generowane automatycznie zza pomocą biblioteki Lombok oraz klas, które przechowują modele odpowiedzialne za autoryzację i zabezpieczenia aplikacji tj. AuthToken, Constants, LoginUser oraz RegisterUser,

Rysunek 3 Diagram klas

* 1. Narzędzia

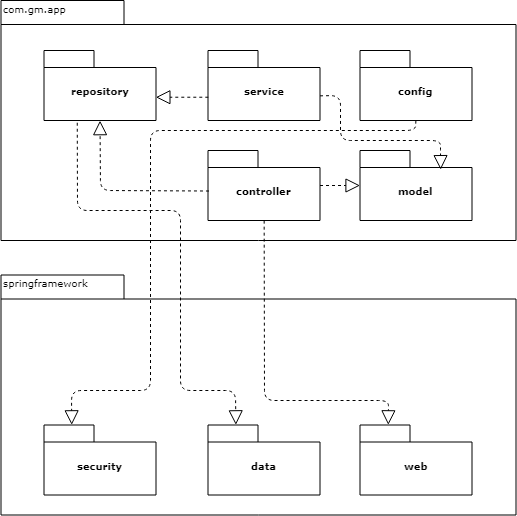
Wybór narzędzi potrzebnych do utworzenia aplikacji, opierał się na zagwarantowaniu wsparcia dla wykorzystywanych technologii oraz języków programowania. Jednym z warunków była również darmowa licencja na oprogramowanie. Zdecydowano się wykorzystać następujące aplikacje:

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, symulująca 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 za pomocą którego zarządza się wartswą front-endu
6. Git – system kontroli wersji zapewniający możliwość kontrolowania procesu wytwarzania oprogramowania.
7. Google Chrome – przeglądarka internetowa na której testowana była budowa warstwy klienckiej aplikacji.
8. Implementacja

Opis implementacji.

* 1. Pakiety

Struktura aplikacji 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 4 Diagram pakietów aplikacji

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

* Pakiet com.gm.app.model zawiera klasy reprezentujące dane, na których wykonywane są operacje.
* Pakiet com.gm.app.controller zawiera klasy kontrolerów dla obiektów w aplikacji oraz częściowo odpowiada za operacje na bazie 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 bazie danych, głównie dla operacji związanych z danymi 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.
  1. Adresowanie

W aplikacji adresowanie zaimplementowane zostało za pomocą adnotacji @RequestMapping, która jest jedną z funkcjonalności bibliotek Spring’a. 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, Pageable pageable) {

return gardenRepository.findByUsername(username, pageable);

}

* 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} (nazwa użytkownika). 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.
  + 1. Adresowanie w aplikacji

Dla powiązanych ze sobą stron, zastosowany został ten sam prefiks adresowania, zgodnie z zastosowanymi regułami zabezpieczeń.

* 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.
  1. Zabezpieczenia

Dostęp do zasobów aplikacji serwerowej zabezpieczony został za pomocą opisanego wcześniej JSON Web Token. Głównymi parametrami, na podstawie których dokonywana jest autoryzacja, są: nazwa oraz hasło użytkownika, token uwierzytelniający, prefiks tokenu oraz czas ważności tokenu.

Klasa WebSecurityConfig rozszerza klasę WebSecurityConfigurerAdapter z pakietu Spring, gdzie zdefiniowane są m.in adresy do funkcji, które nie wymagają autoryzacji.

Listing 2 Konfiguracja zabezpieczeń aplikacji

protected void configure(HttpSecurity http) throws Exception {

http.cors().and().csrf().disable().

authorizeRequests()

.antMatchers("/generate-token", "/register", "/confirm-acc/\*").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.
* 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.
  + 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 3 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());

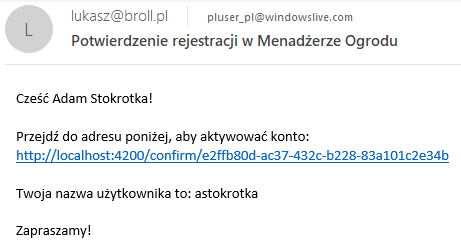
registrationEmail.setFrom("lukasz@broll.pl");

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 SimpleMailService oraz emailService generowany i wysyłany jest następujący email do użytkownika:



Rysunek 5 Email potwierdzający rejestrację konta

Po przejściu na podany adres wywoływana jest metoda kontrolera, który wyszukuje w bazie danych użytkownika z przekazanymi kodem uwierzytelniającym i aktywuje konto:

Listing 4 Metoda aktywująca konto użytkownika

@GetMapping("/confirm-acc/{code}")

public User processConfirmationForm(@PathVariable(value = "code") String code) {

User user = userService.findByConfirmationToken(code);

user.setEnabled(true);

return userService.save(user);

}

* 1. 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 wykorzystane zostało zapytanie pobierające listę roślin dla danego ogrodu oraz zapytanie aktualizujące 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 5 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 czas oraz czy roślina ma status podlanej. Jeśli warunki są spełnione, status pracy jest zmieniany.

Gdy użytkownik wykona daną pracę, pole zawierające czas jej wykonania jest aktualizowane:

Listing 6 Ustawianie nowego czasu wykonania pracy

if(plant.isHydrated()!=plantRequest.isHydrated()){

plant.setHydrated(plantRequest.isHydrated());

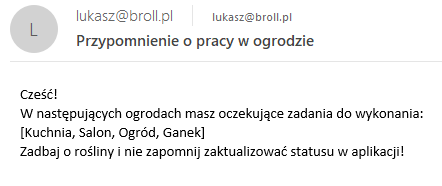
plant.setLastHydration(new Date());

}

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

* 1. Cykliczny e-mail z listą ogrodów wymagających uwagi

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



Rysunek 6 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 7 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 któregokolwiek typu zadania wymaga jest akcja, metoda zwraca isJob=true. Celem uniknięcia prac, które nie są wykonywane dla rośliny, sprawdzany jest dodatkowo cykl wykonania, który dla statusu pracy ‘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 8 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);

});

}

* 1. Czat ogrodników

W celu wymiany informacji między ogrodnikami, zaimplementowany został wspólny pokój czatu dla wszystkich użytkowników aplikacji. Użytkownicy są identyfikowani po nazwie użytkownika podaje 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, 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 9 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 10 Metoda konfuracyjna broker 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.

1. Testy
2. Aplikacja
3. Podsumowanie
4. Sekcja
   1. Sekcja poziomu 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris id dapibus enim. Etiam lobortis pulvinar enim in maximus.

* + 1. Sekcja poziomu 2

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris id dapibus enim. Etiam lobortis pulvinar enim in maximus.

* + - 1. Sekcja poziomu 3

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris id dapibus enim. Etiam lobortis pulvinar enim in maximus.

Tab. 1. Przykład podpisu tabeli

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris id dapibus enim. Etiam lobortis pulvinar enim in maximus.

* 1. Kolejna sekcja poziomu 1
     1. Kolejna sekcja poziomu 2

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris id dapibus enim. Etiam lobortis pulvinar enim in maximus.

Rys. 1. Przykład podpisu rysunku

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris id dapibus enim. Etiam lobortis pulvinar enim in maximus.

Listing. 1. Początkowe żądanie HTTP

GET /script/Articles/Latest.aspx HTTP/1.1  
Host: www.codeproject.com  
Connection: keep –alive  
Cache -Control: max-age=0  
Accept: text/html ,application/xhtml+xml,application/xml|  
User -Agent: Mozilla/5.0 ...  
Accept -Encoding: gzip ,deflate ,sdch  
Accept -Language: en-US...  
Accept -Charset: windows -1251,utf -8...

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris id dapibus enim. Etiam lobortis pulvinar enim in maximus.

Przykład listy numerowanej:

1. Podsumowanie i wnioski

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris id dapibus enim. Etiam lobortis pulvinar enim in maximus. Aliquam erat volutpat. Integer maximus est turpis, ut bibendum ligula accumsan et. Ut eget vestibulum libero. Aliquam erat volutpat. Nullam placerat mauris a lectus tincidunt, et aliquet turpis aliquam. Etiam in malesuada lacus. Proin dignissim augue sit amet auctor elementum. Suspendisse potenti. Vivamus suscipit vulputate massa ac molestie. Suspendisse a justo porttitor, commodo mi at, placerat risus. Integer lobortis augue ac neque suscipit, vel sodales lacus fringilla.

# Literatura

[1] M. Bickley, C. Slominski. A MySQL-based data archiver: preliminary results. Proceedings of ICALEPCS07, Paz. 2007. http://www.osti.gov/scitech/servlets/purl/922267 [dostęp dnia 20 czerwca 2015].

[2] J. Jędrzejczyk, B. Sródka. Segmentacja obrazów metodą drzew decyzyjnych. Raport instytutowy, Politechnika Wrocławska, Wydział Elektroniki, 2007.

# Spis rysunków

[Rys. 1. Przykład podpisu rysunku 18](#_Toc530334916)

# Spis tabel

[Tab. 1. Przykład podpisu tabeli 2](#_Toc465685652)

# Spis listingów

[Listing. 1. Początkowe żadanie HTTP 2](#_Toc465685644)

# Skróty

**OGC** (ang. *Open Geospatial Consortium*)

**XML** (ang. *eXtensible Markup Language*)

**SOAP** (ang. *Simple Object Access Protocol*)

**WSDL** (ang. *Web Services Description Language*)

**UDDI** (ang. *Universal Description Discovery and Integration*)

**GIS** (ang. *Geographical Information System*)

**SDI** (ang. *Spatial Data Infrastructure*)

**ISO** (ang. *International Standards Organization*)

**WMS** (ang. *Web Map Service*)

**WFS** (ang. *Web Feature Service*)

**WPS** (ang. *Web Processing Service*)

**GML** (ang. *Geography Markup Language*)

**SRG** (ang. *Seeded Region Growing*)

**SOA** (ang. *Service Oriented Architecture*)

**IT** (ang. *Information Technology*)

# Dodatek A