P O L I T E C H N I K A Ś W I Ę T O K R Z Y S K A
Wydział Elektrotechniki, Automatyki i Informatyki

Pavlo Dediu
Numer albumu: 087992

Tytuł pracy dyplomowej w języku polskim

Praca dyplomowa
na studiach I-go stopnia
na kierunku Informatyka

Promotor pracy dyplomowej:
dr inż. Karol Wieczorek
Katedra Systemów Informatycznych

Kielce 2023

Tutaj ma być zamieszczona kopia (skan) zadania na pracę

P O L I T E C H N I K A Ś W I Ę T O K R Z Y S K A
WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI I INFORMATYKI
Studia stacjonarne/niestacjonarne
Kierunek: Elektrotechnika, Informatyka, Automatyka i Elektrotechnika Przemysłowa, Teleinformatyka

Zatwierdzam: **Rok akademicki: 2022/23**

.....
Prodziekan ds. studenckich i dydaktyki

ZADANIE NA PRACĘ DYPLOMOWĄ STUDIÓW PIERWSZEGO/DRUGIEGO STOPNIA

Wydano studentowi:

Imię Nazwisko

nr albumu: 99999

I. Temat pracy:

Tytuł pracy dyplomowej w języku polskim

Title of the thesis in English

II. Cel pracy:

Celem pracy jest tu nie powinno pojawić się powtórzenie tytułu pracy, lecz określenie (w dwóch - trzech wierszach) celu jej realizacji

III. Plan pracy (zakres pracy):

W tej części zadania winny być wyszczególnione zagadnienia, będące przedmiotem analizy w ramach pracy, a nie gotowy plan (spis treści) pracy dyplomowej, takie części jak wstęp, czy wnioski są tutaj zbędne. Strukturalizacja pracy (budowa planu PD) winna być efektem wspólnej pracy promotora pracy i studenta. Plan pracy może być opisowy lub w punktach.

IV. Uwagi dotyczące pracy:

(Uwagi pojawiają się tylko w szczególnych sytuacjach – np. gdy praca powstaje na konkretne zamówienie lub w ramach projektu)

V. Termin oddania pracy: zgodnie z Regulaminem Studiów.

VI. Konsultant: Imię i Nazwisko lub wpisać *Praca nie wymaga konsultanta.*

Opiekun merytoryczny	**Promotor pracy dyplomowej**
wpisać tytuł/stopień, imię i nazwisko	wpisać tytuł/stopień, imię i nazwisko
.....	.....
(podpis)	*(podpis)*

Temat pracy dyplomowej celem jej wykonania otrzymałem(am):

Kielce, dnia .....r. .....
czytelny podpis studenta

Załącznik Nr 1
do Antyplagiatowej procedury sprawdzania prac dyplomowych
i elektronicznej archiwizacji prac dyplomowych w Politechnice Świętokrzyskiej
wprowadzonej Zarządzeniem Rektora Nr 21/23

Kielce, dnia .....

.....
Imię i nazwisko studenta, nr albumu

.....
Adres zamieszkania

.....
Studia pierwszego/drugiego* stopnia, forma studiów stacjonarne/niestacjonarne*

.....
Kierunek, zakres

.....
Promotor pracy dyplomowej

OŚWIADCZENIE

Przedkładając w roku akademickim 20...../..... promotorowi pracy dyplomowej studiów pierwszego/drugiego* stopnia, powołanemu przez Dziekana Wydziału .....
Politechniki Świętokrzyskiej, pracę dyplomową pod tytułem: .....

.....

.....

oświadczam, że:

1) przedstawiona praca dyplomowa została opracowana przeze mnie samodzielnie, stosownie do wskazówek merytorycznych opiekuna pracy,
2) przy wykonywaniu pracy dyplomowej wykorzystano materiały źródłowe, w granicach dozwolonego użytku wymieniając autora, tytuł pozycji i miejsce jej publikacji,
3) praca dyplomowa nie zawiera żadnych danych, informacji i materiałów, których publikacja nie jest prawnie dozwolona,
4) przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego/stopnia naukowego w wyższej uczelni,
5) niniejsza wersja pracy jest identyczna z treścią elektroniczną w systemie Archiwum Prac Dyplomowych.

Przyjmuję do wiadomości, że w przypadku ujawnienia w mojej pracy dyplomowej, stanowiącej podstawę nadania tytułu zawodowego, przypisania sobie przeze mnie autorstwa istotnego fragmentu lub innych elementów cudzego utworu lub ustalenia naukowego, rektor, w drodze decyzji administracyjnej, stwierdzi nieważność dyplomu.

Zostałem uprzedzony:

1) o odpowiedzialności karnej wynikającej z art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t. j. Dz. U. z 2022 r. poz. 2509 ze zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.",
2) o odpowiedzialności dyscyplinarnej na podstawie art. 307 ust. 1 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (t. j. Dz. U. z 2022 r. poz. 574, ze zm.): „Student podlega odpowiedzialności dyscyplinarnej za naruszenie przepisów obowiązujących w uczelni oraz za czyn uchybiający godności studenta."

.....
Czytelny podpis studenta

**niepotrzebne skreślić*

Oświadczenie autora pracy – jeśli jest wymagane przez promotora

Załącznik Nr 2
do Antyplagiatowej procedury sprawdzania prac dyplomowych
i elektronicznej archiwizacji prac dyplomowych w Politechnice Świętokrzyskiej
wprowadzonej Zarządzeniem Rektora Nr 21/23

Kielce, dnia .....

.....
Imię i nazwisko studenta, nr albumu

.....
Adres zamieszkania

.....
Studia pierwszego/drugiego stopnia, forma studiów stacjonarne/niestacjonarne

.....
Kierunek, zakres

.....
Promotor pracy dyplomowej

OŚWIADCZENIE AUTORA PRACY

Zgodnie z ustawą z dnia 4 lutego 1994r. o prawie autorskim i prawach pokrewnych (t.j. Dz. U. 2022 poz. 2509), wyrażam zgodę na udostępnianie mojej pracy dyplomowej dla celów naukowych i dydaktycznych.

.....
Czytelny podpis studenta

Strona 8 (tył oświadczenia autora pracy jeżeli wypełniono) – pusta, bez numeru - ten tekst należy usunąć

Tytuł pracy w języku polskim

Streszczenie

Celem

Słowa kluczowe: - do 6

Title in English

Summary

Summary of the thesis in English - 4 to 10 lines

Keywords: - up to 6

Strona 8 lub 10 (tył streszczeń) – pusta, bez numeru - ten tekst należy usunąć

SPIS TREŚCI

1. Wstęp

W obliczu globalnych migracji, uchodźstwa oraz zmieniającej się natury współczesnych konfliktów, istnieje coraz większa potrzeba stworzenia odpowiednich narzędzi i platform, które mogą skutecznie wspierać uchodźców w procesie adaptacji, integracji społecznej oraz zapewnieniu im niezbędnych informacji.

W 2022 roku globalne wydarzenia, w tym konflikt na Ukrainie, spowodowały wzrost migracji. Szacuje się, że w wyniku tego konfliktu liczba uchodźców z Ukrainy przekroczyła 5 milionów osób [1], a w roku 2023 ta liczba wzrosła niemal do 6 milionów [1]. W związku z tym istnieje pilna potrzeba zapewnienia tym osobom szybkiego dostępu do takich infromacji jak lokalizacja najbliższych źródeł pomocy humanitarnej oraz możliwość monitorowania obszarów dotkniętych wysokim napływem uchodźców.

Warto zauważyć, że w Polsce zaangażowanie społeczne, w tym praca wolontaryjna, odgrywa istotną rolę w różnych obszarach, w tym również w wsparciu dla uchodźców. W pierwszym kwartale 2022 roku aż 8.5 miliona osób uczestniczyło w działaniach wolontariackich [2]. To znacząca liczba, która świadczy o gotowości społeczeństwa do angażowania się w pomoc dla potrzebujących, w tym dla uchodźców.

Celem pracy jest opracowanie aplikacji webowej „Portalu Infromacyjnego Wspomagającego uchodżcom”, zapewni zgromadzony dostęp do informacji dotyczącej pomocy dla uchodźców oraz pozwoli wolontariuszom w efektywny sposób dostarczać informacje dotyczące pomocy. Praca obejmuje 5 rozdziałów, w których będą omówione kluczowe elementy związane z opracowaną aplikacją webową.

W drugim rozdziale zostanie przedstawiony opis projektu oraz założenia aplikacji, zostaną także omówione istniejące aplikacje webowe mające na celu wsparcie uchodźców. Trzeci rozdział jest poświęcony implementacji wykonanej aplikacji, przedstawi funkcjonalności udostępnione dla uchodżcow oraz wolontariuszów,

Czwarty rozdział zawiera opis do testów aplikacji. Rozdział piąty zawiera podsumowujące wnioski aplikacji oraz dalsze plany rozwoju.

2.Opis projektu

2.1. Przegląd istniejących rozwiązań

Obecnie istnieje nie wiele inicjatyw oraz aplikacji webowych mających na celu wsparcie uchodźców, większość takich aplikacji nie spełniają w pełni ich potrzeb czy nie są odpowiednio dostosowane do zmieniających się warunków.Najbardziej polularnym przykładem wspierającym uchodźców jest aplikacja mobilna RefAid [3] która jest dostępna dla urządzeń mobilnych z systemem iOS oraz Android.

Po utworzeniu konta użytkownika, aplikacja wyświetla główne menu zawierające listę kategorii pomocy oraz panel nawigacyjny (Rys.2.1).

Po wybraniu konkretnej kategorii, aplikacja wyświetla listę ofert pomocy wyszukiwanych według wybranej kategorii (Rys.2.2). Aby przeczytać szczegółowe informacje dotyczące wybranej oferty pomocy, należy nacisnąć na interesującą nas ofertę pomocy.

Następnie aplikacja wyświetla takie informacje jak tytuł, opis pomocy, dane kontaktowe oraz adres do którego jest przypisana dana pomoc.

Panel nawigacyjny zawiera trzy przyciski, pierwszy przekierowuje nas do menu głównego (Rys.2.1), drugi wyświetla interaktywną mapę (Rys.2.3), a trzeci otwiera powiadomienia dotyczące ofert z pomocą (Rys.2.4). Dzięki mapie, użytkownik może zobaczyć lokalizacje różnych ofert pomocy oraz przefiltrować te oferty według kategorii.

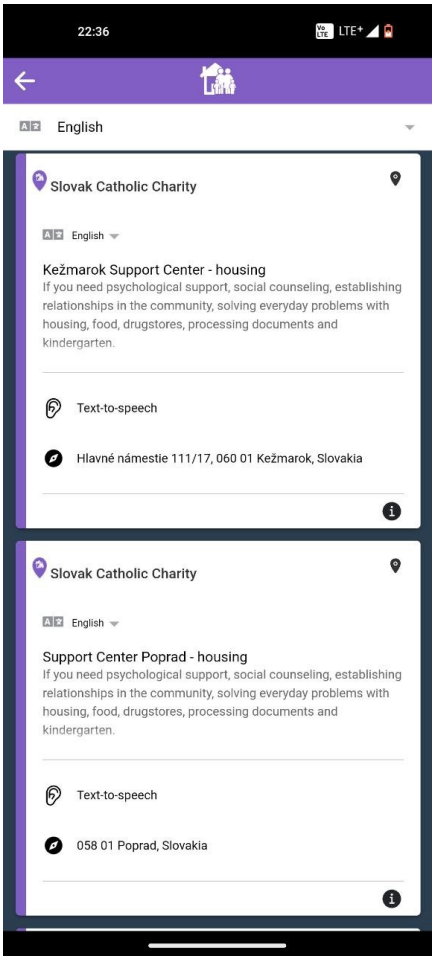
Zalety:

- Przyjazny interfejs
- Możliwość obserwowania pomocy na mapie
- Filtrowanie pomocy według kategorii

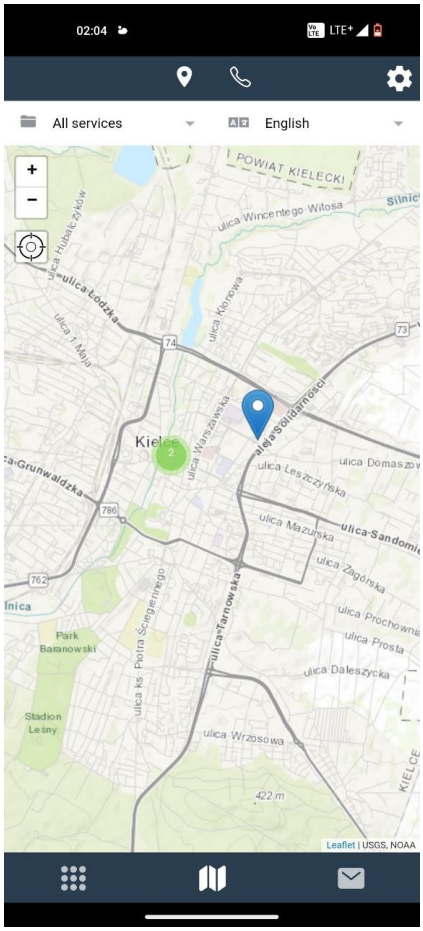
Wady:

- Brak wyszukiwarki
- Brak szybkiego dostępu do listy zaproponowanej pomocy w przypdku gdy użytkowknik jest wolontariuszem
- Brak pomiaru odległości pomiędzy wyszukiwaną oferta a użykownikiem

Rys.2.1 Ekran główny aplikacji RefAid



Rys.2.2 Lista wyszukiwanych ofert pomocy według wybranej kategorii



Rys.2.3 Mapa interaktywna

Powiadomienia dotyczące ofert z pomocą

2.1. Założenia aplikacji

Poniżej znajduje się zestawienie wszystkich przyjętych założeń wykonanych dla aplikacji webowej:

1. Aplikacja powinna zawierać dwa rodzaje konta, dla uchodźcy oraz wolontariusza, zapewniając dostęp do różnych funkcjonalności w zależności od roli użytkownika. Wówczas każdy użytkownik może posiadać wyłącznie jedną role.
2. Aplikacja w celu bezpieczeństwa powinna dokonywać szyfrowanie haseł użytkowników.
3. Zgromadzona informacja dotycząca pomocy powinna zostać przedstawiona w postaci oferty utworzonej przez wolontariusza
4. Aplikacja powinna zawierać mapę interaktywną do wyświetania ofert pomocy.Mapa stanowi intuicyjny interfejs dla użytkowników, pozwalając im na łatwe lokalizowanie ofert w ich otoczeniu.
5. Funkcjonalności dotyczące interakcji pomiędzy użytkownikami a ofertami pomocy powinny być zabiezpieczone uprawnieniami na podstawie roli użytkowników.
6. Uprawnienia do zarządzania tworzonej oferty pomocy są przedzielane wyłącznie do twórcy danej oferty. Jest to kluczowy element w kontroli dostępu do aktualizacji, usunięcia i zarządzania ofertami pomocy.
7. Aplikacjia powinna udzielać użytkownika statystykami dotyczącymi ofert pomocy do prowadzenia analizy.
8. Strona główna aplikacji powinna posiadać czytelny i zrozumiały interfejs zapewniając łatwy dostęp do funkcjonalności użytkownika
9. Użytkownicy-uchodżcy w celu uzyskania szczegółowej informacji dotyczącej oferty pomocy powinni posiadać mozliwości zgłoszenia żądania oferty.

3. Implementacja

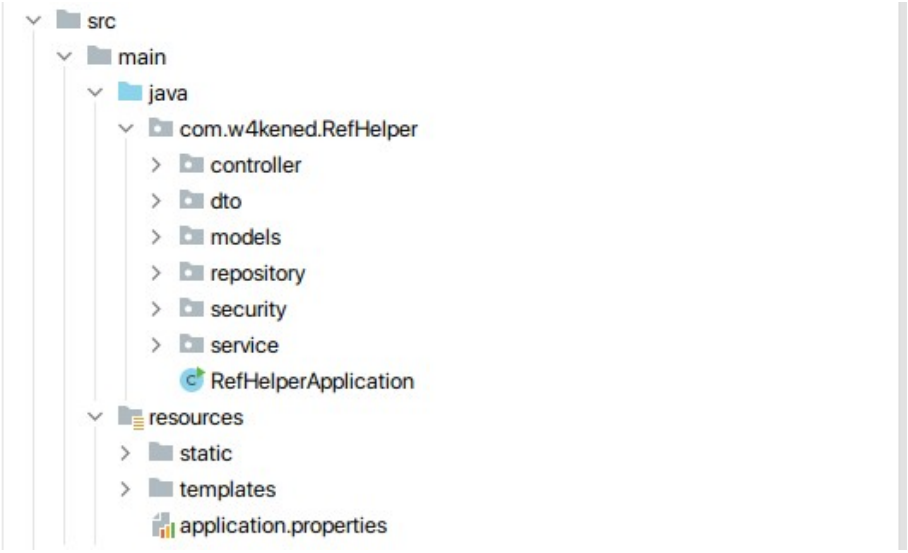
3.1 Użyte technologie

W procesie tworzenia aplikacji skorzystano z technologii, takich jak Java Spring Framework jako główny język backendu ze względu na skalowalność i zapewnione bezpieczeństwo aplikacji. Dodatkowo użyto JavaScript oraz Thymeleaf jako silnik szablonów, umożliwiający budowę elementów interfejsu użytkownika po stronie serwera. W celu przechowywania danych zastosowano bazę danych MariaDB. Dodatkowo, wykorzystano Hibernate Framework do komunikacji z bazą danych.

3.2 Struktura aplikacji

Dana aplikacja jest oparta na wzorcu projektowym MVC (Model-View-Controller), który jest powszechnie stosowany w tworzeniu interfejsów użytkownika, danych i logiki sterującej. Ten wzorzec projektowy podkreśla separację logiki biznesowej oprogramowania i wyświetlania. "Separacja obowiązków" zapewnia lepsze podział pracy i poprawę obsługi.

Poniżej znajduje się struktura projektu:



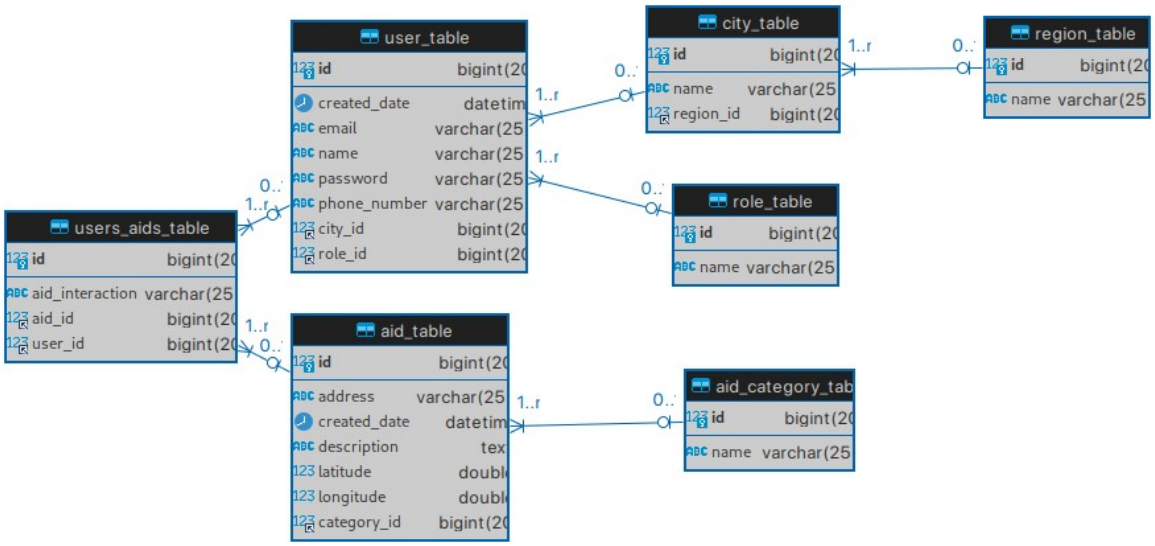
Rys. 3.2.1 Struktura projektu

- Pakiet controller odpowiada za obsługę żądań HTTP i kierowanie ich do odpowiednich metod w modelu lub widoku.
- Pakiet dto (Data Transfer Object) jest odpowiedzialny za przechowywanie danych, które są transferowane między różnymi warstwami aplikacji.
- Pakiet models odpowiada za reprezentację danych biznesowych aplikacji. Każda klasa w tym pakiecie jest zwykle pojedynczą klasą reprezentującą (POJO - Plain Old Java Object), która przechowuje stan jednego obiektu biznesowego.

- Pakiet repository jest odpowiedzialny za obsługę operacji CRUD (Create, Read, Update, Delete) na danych biznesowych.
- Pakiet security jest odpowiedzialny za obsługę zabezpieczeń aplikacji.
- Pakiet service w projekcie Spring MVC jest odpowiedzialny za obsługę logiki biznesowej aplikacji.
- Folder resources służy do przechowywania plików HTML oraz XML

3.3 Bazy Danych

Każda tabela zawiera kluczowe dane wykorzystywane przez stronę internetową. Te informacje są używane między innymi do prezentowania ofert pomocy, lub żądania ich przez uchodźcy .Oto krótki opis wszystkich tabel znajdujących się w bazie danych:



Rys. 3.3.1 Diagram encji

region_table – tabela przechowująca regiony/województwa.

city_table – tabela przechowująca miasta, które należą do jednego regionu/województwa.

role_table – tabela przechowująca roli użytkowników. Istniejące role to:

- ROLE_VOLUNTEER – rola wolontariusza.
- ROLE_REFUGEE – rola uchodźcy.

System może być rozszerzony o dodatkowe role.

user_table – tabela przechowująca użytkownika, każdy użytkownik jest przypisany do jednego miasta oraz posiada jedną rolę.

aid_category_table – tabela przechowująca kategorie pomocy, do kategorii należą takie kategorie jak:

- Basic Necessities Aid – dana kategoria reprezuntje pomocy z wyżywieniem oraz mieszkaniem.
- Healthcare Aid – kategoria reprezentująca pomoc z opieką zdrowotną.
- Education Aid – kategoria reprezentująca pomoc z edukacją oraz , korepytycjami.

- Employment Aid – kategoria reprezentująca pomoc w zatrudnieniu.
- Legal Aid – kategoria reprezentująca pomoc prawną.
- Community Aid – kategoria reprezentująca pomoc społeczna.

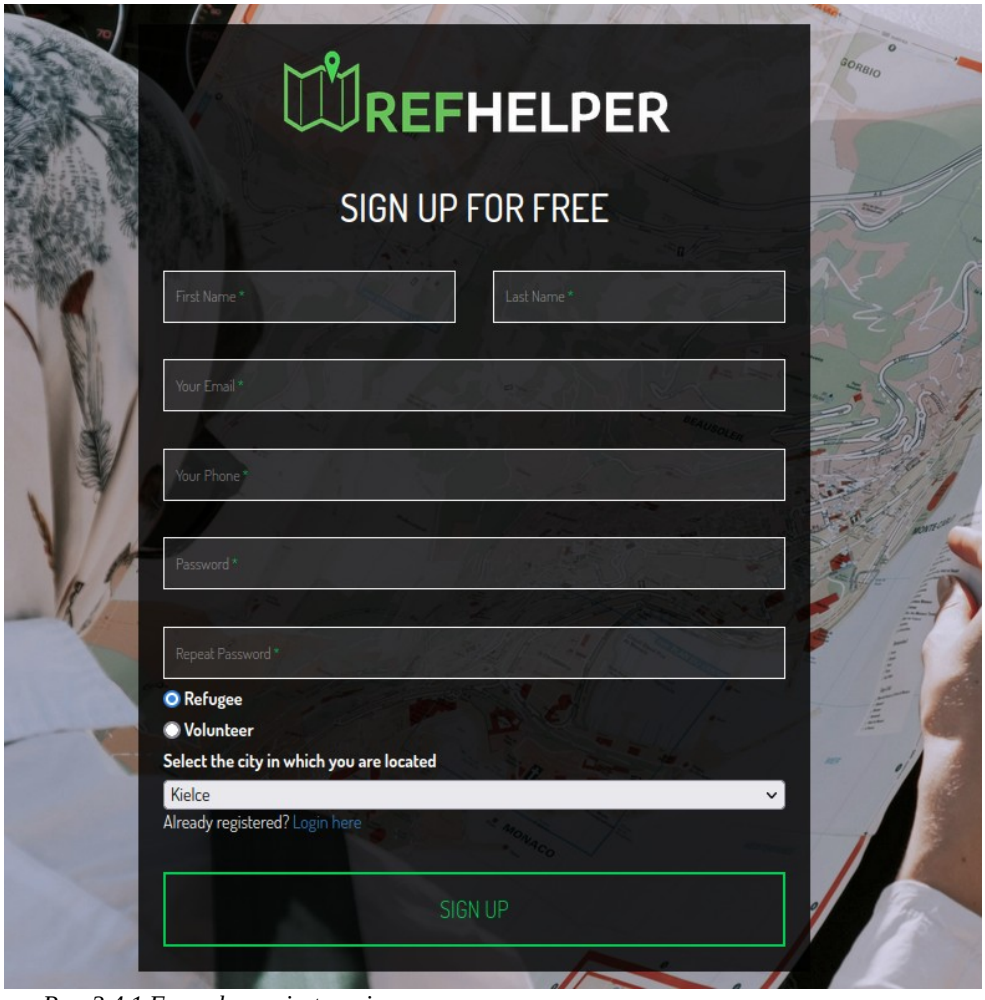
System może być rozszerzony o dodatkowe kategorie.

aid_table – tabela przechowująca oferty pomocy które posiadają jedną kategorie

users_aids_table – tabela przechowująca interakcje pomiędzy użytkownikiem a ofertą pomocy

3.4 Autoryzacja użytkowników

Aplikacja internetowa wykorzystuje mechanizm autoryzacji, który odpowiada za zapewnienie dostępu użytkownikowi do usług oferowanych przez Portal Informacyjny dla Uchodźców. Aby jednak użytkownik mógł z nich skorzystać, musi przejść przez szereg etapów walidacyjnych, z których pierwszym jest **rejestracja.**



Rys. 3.4.1 Formularz rejestracyjny

Dane wprowadzone do pól tekstowych są najpierw weryfikowane po stronie klienta, a następnie również po stronie serwera, zapewniając podwójne sprawdzenie przed zapisaniem ich do bazy danych.

```
const passwordField = document.getElementById('password');
const repeatPasswordField = document.getElementById('repeatPassword');
const mismatchAlert = document.getElementById('passwordMismatch');
const submitButton = document.getElementById('sumbitButtonId');

function checkPasswordMatch() {
    if (passwordField.value !== repeatPasswordField.value &&
                          passwordField.value.length < 8) {
        mismatchAlert.style.display = 'block';
        submitButton.style.display = 'none';

    } else {
        mismatchAlert.style.display = 'none';
        submitButton.style.display = 'block';
    }
}
passwordField.addEventListener('input', checkPasswordMatch);
repeatPasswordField.addEventListener('input', checkPasswordMatch);
```

Listing 3.1 Walidacja hasła z użyciem JavaScript

Funckja checkPasswordMatch (Listing 3.1) sprawdza, czy podczas wpisywania danych, wprowadzone hasła w obu polach są zgodne i czy są one wystarczającej długośći.

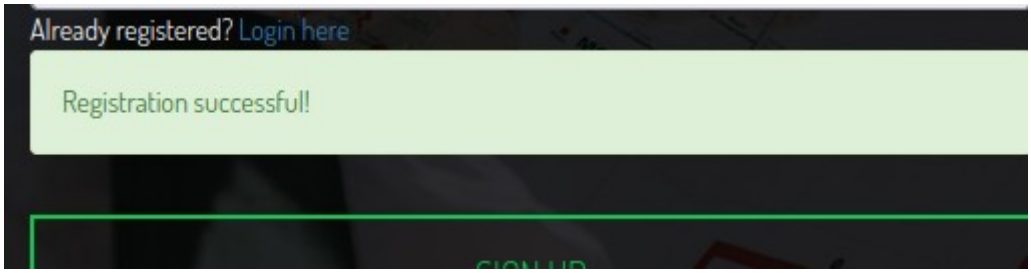
```
@PostMapping("/register/save")
public String register(@Valid @ModelAttribute("user") UserDto user,
BindingResult result, Model model) {
    if (!user.getPassword().equals(user.getRepeatPassword())) {
        result.rejectValue("repeatPassword", "error.user", "Passwords do not
match");
        return "redirect:/register";
    }
    UserEntity existingUserEmail = userService.findByEmail(user.getEmail());
    if(existingUserEmail != null &&
       existingUserEmail.getEmail() != null &&
       existingUserEmail.getEmail().isEmpty()) {
        return "redirect:/register?fail";
    }
    if(result.hasErrors()) {
        model.addAttribute("user", user);
        return "authSignup";
    }
    userService.saveUser(user);
    model.addAttribute("message", "Registration successful!");
    return "redirect:/register?success";
}
```

Listing 3.2 Metoda obsługiwająca żądanie POST pod adresem "/register/save"

Po kliknięciu przycisku Sign Up, jest wykonywana metoda POST pod adresem /register/save która jest obsługiwana przez metodę register (Listing 3.2), w którym parametr 'user' stanowi obiekt transferu danych. Ten obiekt przechowuje wprowadzone dane z formularza.

Wprowadzone użytkownikiem dane przechodzą przez szereg dodatkowej walidacji oraz zostają sprawdzane, czy użytkownik o podanym adresie e-mail istnieje w bazie danych.

W przypadku sukcesu dane użytkownika są zapisywane do bazy danych poprzez wywołanie metody saveUser (Listing 3.3) z warstwy pośredniej, a na stronie rejestracji zostaje wyświetlony komunikat informujący o sukcesie (Rys 3.4.2).



Rys. 3.4.2 Komunikat informujący o sukcesie rejestracji.

```java
@Override
public boolean saveUser(UserDto userDto) {
    UserEntity userEntity = new UserEntity();
    userEntity.setName(userDto.getFirstName()+" "+userDto.getLastName());
    userEntity.setEmail(userDto.getEmail());
    userEntity.setPassword(passwordEncoder.encode(userDto.getPassword()));
    userEntity.setPhoneNumber(userDto.getPhoneNumber());
    LocalDateTime now = getCurrentTimeStamp();
    userEntity.setCreatedDate(now);
    RoleEntity roleEntity;
    switch (userDto.getSelectedRole()) {
        case 2 -> {
            roleEntity = roleRepository.findByName("ROLE_REFUGEE");
            userEntity.setRoleEntity(roleEntity);}
        case 3 -> {
            roleEntity = roleRepository.findByName("ROLE_VOLUNTEER");
            userEntity.setRoleEntity(roleEntity);}
    }
    CityEntity cityEntity =cityRepository.findByName(userDto.getCityName());
    userEntity.setCityEntity(cityEntity);
    try {
        UserEntity existingUser =
                        userRepository.findByEmail(userDto.getEmail());
        if (existingUser != null || userDto.getPassword().length() < 8) {
            return false;
        }
        userRepository.save(userEntity);
        return true;
    } catch (Exception e) {
        return false;
    }
}
```

Listing 3.3 Metoda zapisująca użytkownika do bazy danych

Metoda saveUser (Listing 3.3) przyjmująca parametr obiektu transferu danych użytkownika, szyfruje hasło użytkownika za pomocą algorytmu BCrypt, przepisuje dane z pól obiektu podanego w parametrze do obiektu encji użytkownika a następnie taki obiekt

zostaje zapisany za pomocą podstawowej metody save zdefiniowanej w interfejsie CrudRepository (Listing 3.4)

```
@NoRepositoryBean
public interface CrudRepository<T, ID> extends Repository<T, ID> {
    <S extends T> S save(S entity);
      ...
}
```

Listing 3.4 Interfejs CrudRepository

Następnym etapem autoryzacji jest **logowanie**.

Gdy użytkownik wprowadza swoje dane do logowania, aplikacja przesyła te informacje do systemu uwierzytelniania (Listing 3.5) w celu weryfikacji poprawności podanych danych

```
@Override
public UserDetails loadUserByUsername(String email) throws
UsernameNotFoundException {
UserEntity user = userRepository.findByEmail(email);
if (user != null) {
    if (user.getEmail() == null || user.getEmail().isEmpty()) {
        throw new IllegalArgumentException("Email is null or empty for user
with ID: " + user.getId());
    }
    if (user.getPassword() == null || user.getPassword().isEmpty()) {
        throw new IllegalArgumentException("Password is null or empty for
user with ID: " + user.getId());
    }
    SimpleGrantedAuthority authority = new
SimpleGrantedAuthority(user.getRoleEntity().getName());
    User authUser = new User(user.getEmail(),user.getPassword(),
            Collections.singletonList(authority));
    return authUser;
} else {
    throw new UsernameNotFoundException("Invalid username or password");
}
}
```

Listing 3.5 Metoda obsługiwająca proces uwierzytelniania

Jeśli wprowadzone przez użytkownika dane zgadzają się z danymi znajdującymi się w bazie danych, aplikacja przekierowuje użytkownika do strony głównej.

Dostęp do strony głównej jest zabezpieczony za pomocą modułu Spring Security, poprzez rozszerzanie klasy WebSecurityConfigurerAdapter (Listing 3.6) i zdefiniowanie reguł dotyczących uwierzytelniania. Dzięki temu zabezpieczamy dane przed nieautoryzowanym dostępem.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig  extends WebSecurityConfigurerAdapter {
    private CustomUserDetailsService userDetailsService;
    @Autowired
    public SecurityConfig(CustomUserDetailsService userDetailsService){
        this.userDetailsService = userDetailsService;
    }
    @Bean
    public static PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
                .authorizeRequests()
                .antMatchers("/login", "/addAid," "/editAid/*","/register",
"/css/**", "/js/**")
                .permitAll()
                .antMatchers("/home")
                .authenticated()
                .and()
                .formLogin(form -> form
                    .loginPage("/login")
                    .usernameParameter("email")
                    .defaultSuccessUrl("/home")
                    .loginProcessingUrl("/login")
                    .failureUrl("/login?error=true")
                    .permitAll()
                )
                .logout(logout -> logout
                        .logoutRequestMatcher(
                                new AntPathRequestMatcher("/logout"))
                        .permitAll()
                );
    }
    @Override
    public void configure(AuthenticationManagerBuilder builder) throws
Exception {
        builder.userDetailsService(userDetailsService)
                        .passwordEncoder(passwordEncoder());
    }
}
```

Listing 3.6 Klasa definiująca opcje konfiguracji bezpieczeństwa aplikacji

3.5 Wyświetlanie ofert pomocy

Dla uchodźców zapewnienie dostępu do informacji o ofertach pomocy stanowi kluczowe wyzwanie. Wykorzystanie mapy jako narzędzia wizualnego mogłoby pomóc w prezentacji informacji dotyczących lokalizacji i rodzaju pomocy. W ramach prezentacji ofert pomocy zostało wykorzystano darmową bibliotekę Leaflet do inicjalizacji mapy (Listing 3.7)

```
<div id="map"></div>
...
<script th:src="@{app/leaflet/leaflet.js}"></script>
<link rel="stylesheet"
href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css" />
<script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"></script>
...
var standardMapLayer =
L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
   attribution: '&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors'
});
...
var map = L.map('map', {
   layers: [standardMapLayer]
}).setView([51.87801, 20.61224], 7);
```

Listing 3.7 Inicjalizacja biblioteki Leaflet

Proces wyświetlania pomocy polega na pobieraniu obiektów ofert pomocy z określonego adresu URL (Listing 3.8). Każdy element z listy jest przekształcany do formatu JSON, a następnie dodawany do mapy za pomocą funkcji addMarker.

```
let wrappedMarkers = [];
function displayAidMarkersFromServer() {
   const url = '/getAllAidMarkers';
   fetch(url)
      .then(response => response.json())
         .then(aid => {
            aid.forEach(marker => {
               addMarker(marker);
               wrappedMarkers.push(marker);
            });
         }).catch(error => {
         console.error('Wystąpił błąd podczas pobierania ofert pomocy z
serwera:', error);
      });
}
displayAidMarkersFromServer();
```

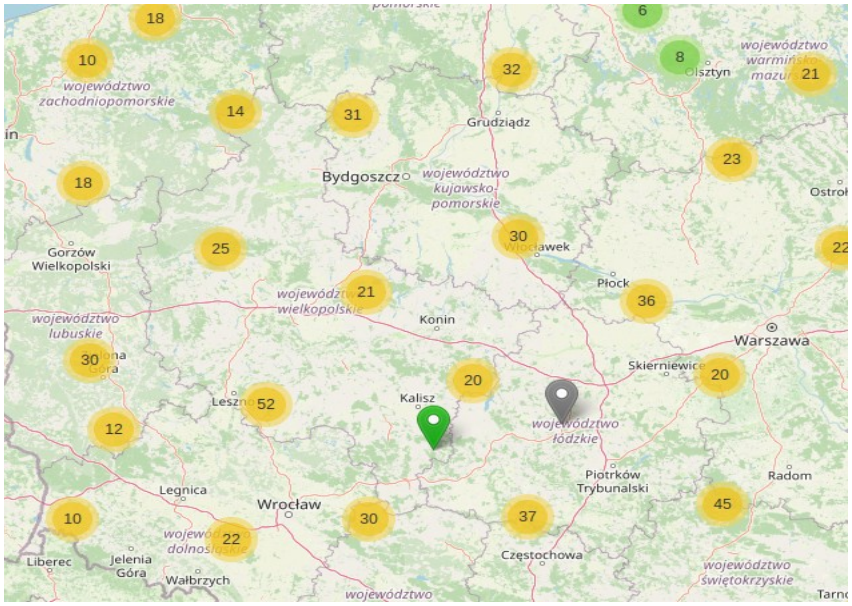
Listing 3.8 Funckja odpowiedzialna za pobieranie listy ofert z serwera

Podczas implementacji wyświetlania ofert pomocy zostało napotkano problem z organizacją znaczników na mapie, ponieważ duża liczba takich znaczników mogłaby spowodować zawieszenie przeglądarki, co jest nieakceptowalne.
Na szczęście, biblioteka Leaflet posiada moduł MarkerCluster , który rozwiązuje opisany wyżej problem przez grupowanie znaczników znajdujących się blisko siebie.
W związku z tym, ten moduł został wykorzystany w funkcji addMarker (Listing 3.9)

```
<link rel="stylesheet"
href="https://unpkg.com/leaflet.markercluster@1.5.0/dist/MarkerCluster.css" /
>
<link rel="stylesheet"
href="https://unpkg.com/leaflet.markercluster@1.5.0/dist/MarkerCluster.Defaul
t.css" />
<script
src="https://unpkg.com/leaflet.markercluster@1.5.0/dist/leaflet.markercluster
.js"></script>
var markerCluster = L.markerClusterGroup();
function addMarker(marker) {
   let color = convertIdToColor(marker.aidCategoryEntity.id);
   if (marker.description) {
      addToSidebar(marker);
      theMarker = L.marker([marker.latitude, marker.longitude], { icon:
markerIcons[color] });
      theMarker.bindPopup(
...
      );
      markerCluster.addLayer(theMarker);
      markerlist.push(theMarker);
      map.addLayer(markerCluster);
   }
}
```

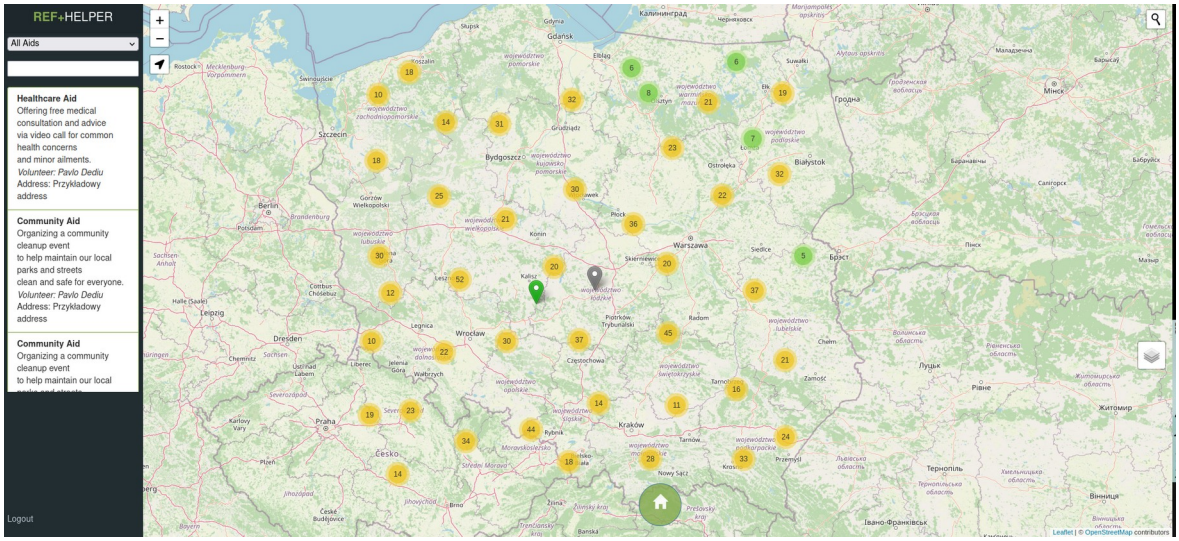
Listing 3.9 Funckja dokonująca dodawanie znaczników pomocy do mapy

Jako rezultat zastosowania tego modułu, uzyskujemy funkcjonalność grupowania oraz wyświetlania ofert pomocy na mapie, co pozwala na klarowne prezentowanie tych informacji użytkownikowi.(Rys. 3.5.1).



Rys. 3.5.1 Wyświetlanie ofert pomocy na mapie

Niezależnie od roli użytkownika, strona główna oprócz mapy interaktywnej zawiera panel boczny oraz umieszczony na dole ekranu przycisk wyświetlający modal z menu głównym. (Rys.3.6.1)



Rys. 3.6.1 Interfejs graficzny strony głównej

Panel boczny zawiera listę informacyjną zawierającą oferty pomocy, wyszukiwarkę wyszukiwająca oferty na liście według słów kluczowych oraz filtr znaczników.

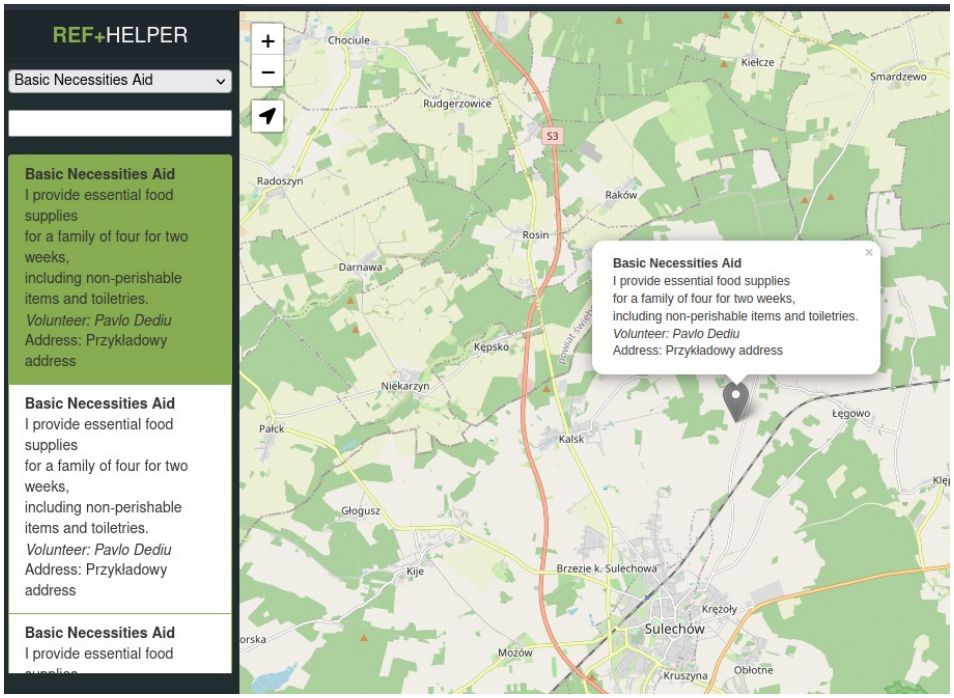
Elementy listy reprezentują owrapowane obiekty znaczników ofert pomocy, zawierające opis pomocy, adres oraz dane wolontariusza oferującego wsparcie.

Są one dodawane do listy za pomocą funkcji addToSidebar (Listing 3.10).

```
function addToSidebar(marker) {
...
    subheading.textContent = convertIdToName(marker.aidCategoryEntity.id);
...
    descriptionDiv.textContent = marker.description;
...
    creatorDiv.textContent = "Volunteer: "
              +  marker.usersAidsEntities[0].userEntity.name;
...
    addressDiv.textContent = "Address: "+ marker.address;
    listItem.addEventListener('click', function(e) {
        for (var i  = 0; i < markerlist.length; i++) {
            if (markerlist[i]._latlng.lat == marker.latitude &&
markerlist[i]._latlng.lng == marker.longitude) {
                map.setView([marker.latitude, marker.longitude], 12);
                markerlist[i].openPopup();
                break;
            }
        }
    });
...
    listItem.appendChild(div);
    sidebarList.appendChild(listItem);
}
```

Listing 3.0 Funckja dokonująca umieszczanie znaczników pomocy do listy informacyjnej

Każdy element listy posiada zdarzenie kliknięcia, które powoduje przeniesienie widoku użytkownika na mapie do miejsca, w którym dana oferta pomocy jest umiejscowiona (Rys. 3.6.2).



Rys. 3.6.2 Zdarzenie kliknięcia elementa listy

Rol użytkownika jest kluczowym atrybutem konta użytkownika który definiuje dostęp do róznego rodzaju informacji oraz możliwośći interakcji z ofertami pomocy. Sprawdzenie roli zastępuje w metodzie getHomePage która odpowiada za ładowanie szablonu strony głównej oraz umieszczanie infomacji dotyczących ofert do zmiennych Thymeleaf w celu wyświetlania tej informacji w plikach HTML (Listing 3.11).

```
@GetMapping("/home")
public String getHomePage(Model model) throws NotFoundException {
    UserEntity user = new UserEntity();
    String email = SecurityUtil.getSessionUser();
...
    String userFullName = userEntity.getName();
    model.addAttribute("userFullName", userFullName);
    if (Objects.equals(roleName, "ROLE_VOLUNTEER")) {
        List<AidEntity> aids = new ArrayList<>();
        aids = aidService.findByCreatorUserId(userEntity.getId());
        List<Long> aidIds = aids.stream().map(AidEntity::getId).toList();
...
        model.addAttribute("aidsOfferedCount", aids.toArray().length);
        model.addAttribute("aidsRequestedCount", requests.toArray().length);
        model.addAttribute("aidsList", aids);
        model.addAttribute("requestedAidsList", requests);
        model.addAttribute("layout", "layout");
    } else { ... }
    return "home";
}
```

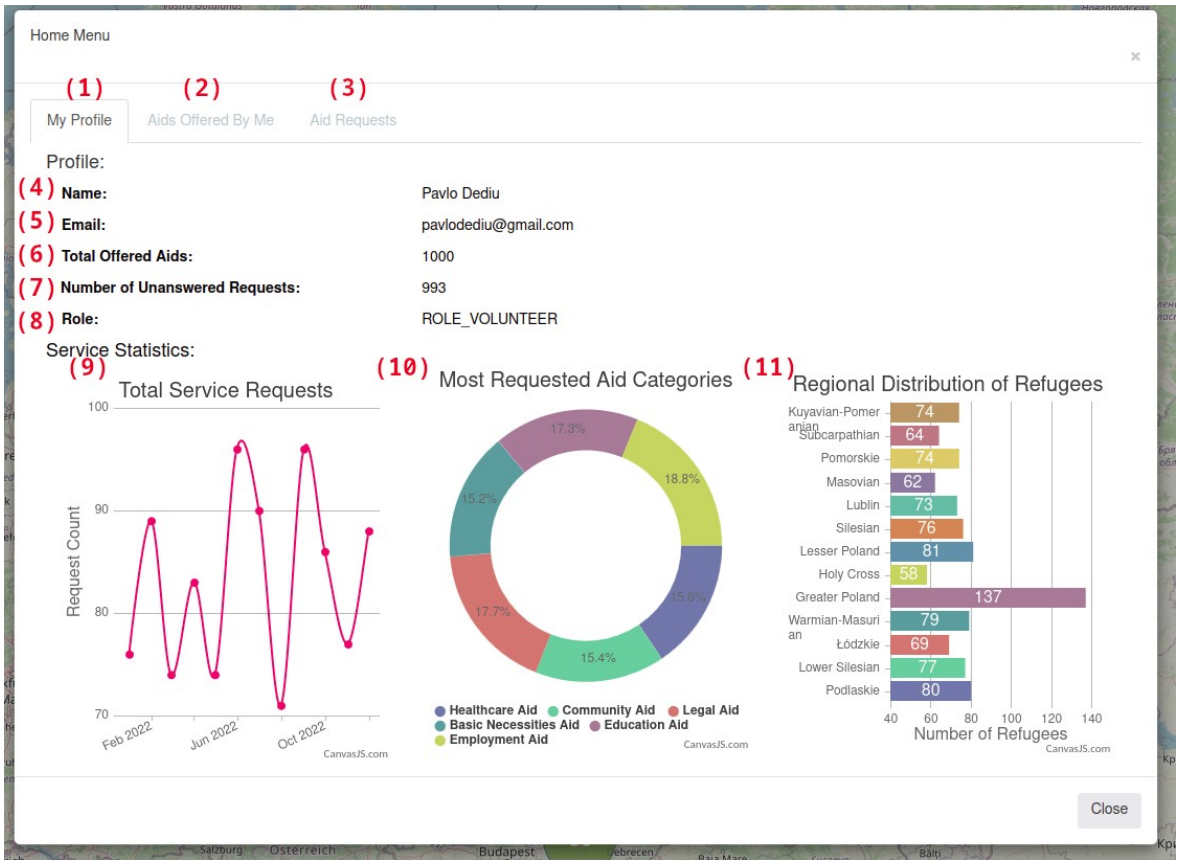
Listing 3.11 Metoda ładująca stronę główną

3.6 Funkcjonalność wolontariuszy

Jeśli użytkownik posiada rolę z nazwą 'ROLE_VOLUNTEER', aplikacja udostępnia mu informacje dotyczące wcześniej utworzonych przez niego ofert, daje możliwość tworzenia nowej oferty pomocy oraz możliwość aktualizacji istniejących ofert.

Aby skorzystać z wyżej opisanych funkcjonalności, należy wyświetlić modal z menu głównym za pomocą przycisku na dole ekranu (Rys. 3.6.2)

Rys. 3.6.2 Przycisk wyświetlający menu główne



Rys. 3.6.3 Modal zawierający menu główne użytkownika-wolontariuszy

W menu głównym (Rys. 3.6.3) są umieszczone takie elementy jak:

1. Zakładka z profilem użytkownika
2. Zakładka z ofertami pomocy utworzonych przez użytkownika
3. Zakładka z żądaniami ofert utworzonych przez użytkownika
4. Imię i nazwisko przypisane do konta użytkownika
5. Email przypisany do konta użytkownika
6. Liczba utworzonych ofert pomocy przez użytkownika
7. Liczba żądań bez odpowiedzi ofert utworzonych przez użytkownika
8. Rola użytkownika
9. Wykres pokazujący ilość żądań pomocy w czasowym przedziale
10. Wykres pokazujący ilość najczęściej żądanych kategorii pomocy w serwisie
11. Wykres pokazujący ilość użytkowników-uchodźców w każdym województwie

Dla wykresów **9-11** dane są pobierane za pomoców określonych endpointów (Listing 3.12). Taki sposób dostarczania informacji dla wykresów umożliwi pobieranie danych asynchronicznie, co przyspieszy ładowanie strony głównej aplikacji.

```
@GetMapping("/getTotalServiceRequests")
public ResponseEntity<Map<String, Long>> getTotalServiceRequests() {
    Map<String, Long> dataForChart =
                    usersAidsService.getOverallDataForChart();
    return ResponseEntity.ok().body(dataForChart);
}
@GetMapping("/getMostRequestedServices")
public ResponseEntity<Map<String, Long>> getMostRequestedServices() {
    Map<String, Long> dataForChart =
                    usersAidsService.getMostRequestedDataForChart();
    return ResponseEntity.ok().body(dataForChart);
}
@GetMapping("/getRegionalDistributionOfRefugees")
public ResponseEntity<Map<String, Long>> getRegionalDistributionOfRefugees()
{
    Map<String, Long> dataForChart =
                    userService.getRegionalDistributionOfRefugeesForChart();
    return ResponseEntity.ok().body(dataForChart);
}
```

Listing 3.12 Endpointy zwracające dane dla wykresów

Przedstawione endpointy zwracają mapy które gromadzą poszczególne wyniki natywnych zapytań SQL które są wywołane za pomocą warstwy pośredniej (Listing 3.13), (Listing 3.14). Natywne zapytania SQL pozwalają na bezpośrednie korzystanie z języka zapytań SQL, co daje większą kontrolę nad wykonywanymi operacjami.

```
@Query(value = "SELECT DATE_FORMAT(created_date, '%m-%Y') AS date, COUNT(id)
AS count FROM users_aids_table WHERE aid_interaction = 'REQUESTING' "+
"GROUP BY MONTH(created_date), YEAR(created_date)", nativeQuery = true)
List<Object[]> getCountOfAidRequestsByMonthAndYear();
@Query(value = "SELECT act.name AS Category, count(uat.id) AS count " +
"FROM users_aids_table uat " +
"INNER JOIN aid_table at ON uat.aid_id  = at.id " +
"INNER JOIN aid_category_table act ON at.category_id = act.id " +
"WHERE uat.aid_interaction = 'REQUESTING' GROUP BY act.name ",nativeQuery =
true)
List<Object[]> getCountOfAidRequestsByMostRequestedCategory();
```

Listing 3.13 Używane natywne zapytania SQL w menu głównym wolontariuszy. Część 1/2

```
@Query(value = "SELECT rt2.name AS Region, COUNT(ut.id) AS Count " +
        "FROM user_table ut " +
        "INNER JOIN role_table rt ON ut.role_id = rt.id " +
        "INNER JOIN city_table ct ON ut.city_id = ct.id " +
        "INNER JOIN region_table rt2  ON ct.region_id = rt2.id " +
        "WHERE rt.name = 'ROLE_REFUGEE' " +
        "GROUP BY rt2.name "
        , nativeQuery = true)
List<Object[]> getRegionalDistributionOfRefugees();
```

Listing 3.14 Używane natywne zapytania SQL w menu głównym wolontariuszy. Część 2/2

Wynik zapytania getCountOfAidRequestsByMonthAndYear (Listing 3.13) pozwoli użytkownikowi śledzić, w jakim konkretnym okresie czasu odnotowano największą liczbę żądań pomocy w serwisie.

Natomiast wynik zapytania getCountOfAidRequestsByMostRequestedCategory (Listing 3.13) udzieli użytkownika informacją która kateogoria pomocy jest najbardziej żądana w serwisie.

Ostatnie zapytanie SQL getRegionalDistributionOfRefugees (Listing 3.14) umożliwia użytkownikom śledzenie, w którym województwie zarejestrowano największą liczbę użytkowników-uchodźców.

Tworzenie oferty jest zrealizowane za pomocą formularza który wyświetla się za pomocą metody showAddAidForm (Listing 3.15).

```
<a style="color: white !important;" class="btn btn-primary"
th:href="@{/addAid}">Offer Aid</a>
...
@GetMapping("/addAid")
public String showAddAidForm(Model model) {
    model.addAttribute("aidDto", new AidDto());
    return "addAidForm";
}
```

Listing 3.15 Metoda wyświetlająca formularz do tworzenia oferty

Formularz zawiera funkcje validateForm(Listing 3.16) która uniemożliwia zapisanie pustych pól adresu oraz opisu

```
function validateForm() {
    var addressTmp = document.getElementById('address').value;
    var descriptionTmp = document.getElementById('description').value;
    if (addressTmp == "" || descriptionTmp == "") {
        alert("Please fill out the Description and Address");
        return false;
    }
}
```

Listing 3.16 Funckja uniemożliwiająca zapisanie pustych pól adresu oraz opisu

Do poprawnej walidacji adresu został użyty moduł Leaflet Control Geocoder który zapewnia geowyszukiwanie adresu za pomocą pola zamieszczonego na minimapie (Listing 3 .17)

```
...
<div class="form-group">
    <label for="minimap">Address:</label>
    <div id="minimap" style="height: 20rem;"></div>
</div>
<div class="form-group">
    <input type="text" class="form-control" id="address" name="address"
           th:field="*{address}" readonly
           placeholder="Address will populate here after founding on map"
    >
</div>
...
var geocoder = L.Control.geocoder({
    defaultMarkGeocode: false
}).on('markgeocode', function(e) {
    var latlng = e.geocode.center;
...
   const categoryId = document.getElementById('aidCategory').value;
    const color = convertIdToColor(categoryId)
...
    map.fitBounds(e.geocode.bbox);
    document.getElementById('address').value = e.geocode.name;
    document.getElementById('latitude').value = parseFloat(latlng.lat);
    document.getElementById('longitude').value = parseFloat(latlng.lng);
}).addTo(map);
```

Listing 3.17 Geowyszukiwanie za pomocą pola zamieszczonego na minimapie

Gdy użytkownik zatwierdza formularz, wywoływana jest metoda POST addAid (Listing 3.18), która pobiera uzupełnione pola formularza i dokonuje zapis oferty do bazy danych

```
@PostMapping("/addAid")
public String addAid(@ModelAttribute("aidDto") AidDto aidDto) {
    aidService.saveAid(aidDto);
    return "redirect:/home";
}
```

Listing 3.18 Metoda pobierająca uzupełnione pola formularza do zapisu oferty

W podobny sposób jest zrealizowane usuwanie oferty za pomocą metody GET deleteAid (Listing 3.19)

```
<a style="color: white !important;" th:href="@{'/deleteAid/'+ ${aid.id}}"
class="btn btn-danger" onclick="return confirmDelete()">Delete</a>
...
@GetMapping("/deleteAid/{id}")
public String deleteAid(@PathVariable("id") Long id) throws NotFoundException
{
    aidService.deleteAidById(id);
    return "redirect:/home";
}
```

Listing 3.19 Metoda pobierająca id oferty do usuwania z bazy danych

Aktualizacja oferty pomocy jest dokonywana za pomocą formularza zawierającego dane w obiekcie Model w metodzie GET showEditAidForm (Listing 3.20)

```
@GetMapping("/editAid/{id}")
public ModelAndView showEditAidForm(@PathVariable("id") Long id, Model model)
throws NotFoundException {
    AidEntity aidEntity = aidService.findByAidId(id);
    if (aidEntity == null) {
        throw new NotFoundException("Not aid with   ID " + id);
    }
    ModelAndView editView = new ModelAndView();
    editView.setViewName("editAidForm");
    AidDto aidDto = new AidDto(aidEntity);
    editView.addObject(aidDto);
    return editView;
}
```

Listing 3.20 Metoda wyświetlająca formularz do edytowania oferty

Zatwierdzanie formularza do edytowania powoduje wywołanie metody POST editAid (Listing 3.21)

```
@PostMapping("/editAid/{id}")
public ModelAndView editAid(@PathVariable("id") Long id,
@ModelAttribute("aidDto") AidDto aidDto) throws NotFoundException {
    ModelAndView editView = new ModelAndView();
    aidService.updateAid(aidDto, id);
    editView.addObject(aidDto);
    editView.setViewName("redirect:/home");
    return editView;
}
```

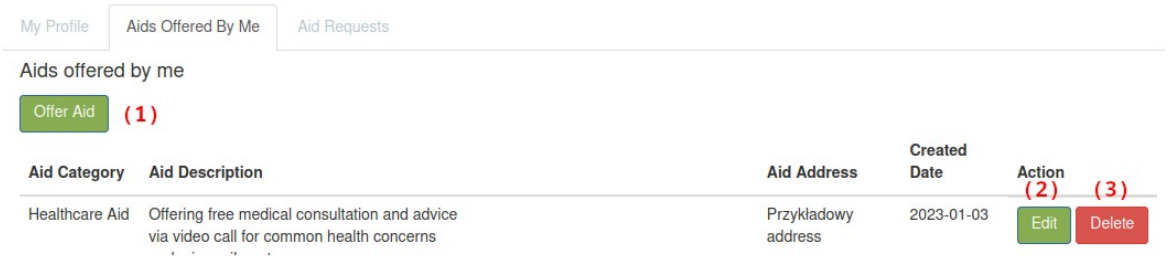
Listing 3.21 Metoda pobierająca aktualizowane pola formularza do zapisu oferty

Określona metoda odwołuje się do warstwy pośredniej w celu aktualizacji oferty za pomocą metody updateAid (Listing 3.22)

```
@Override
@Transactional
public boolean updateAid(AidDto aidDto, Long id) throws NotFoundException{
    UserEntity userEntity =
userService.findByEmail(SecurityUtil.getSessionUser());
    Optional<AidEntity> optionalAidEntity = aidRepository.findById(id);
    if (optionalAidEntity.isPresent()) {
        AidEntity existingAidEntity = optionalAidEntity.get();
        if (!usersAidsRepository.isCreatorOfAid(existingAidEntity.getId(),
                userEntity.getId()) ) {
            return false;
        }
        existingAidEntity.setDescription(aidDto.getDescription());
        existingAidEntity.setLatitude(aidDto.getLatitude());
        existingAidEntity.setLongitude(aidDto.getLongitude());
        existingAidEntity.setAddress(aidDto.getAddress());
        aidRepository.save(existingAidEntity);
        UsersAidsEntity usersAidsEntity  = new UsersAidsEntity();
        usersAidsEntity.setAidEntity(existingAidEntity);
        usersAidsEntity.setUserEntity(userEntity);
        usersAidsEntity.setAidInteraction(AidInteraction.MODIFYING);
        usersAidsEntity.setCreatedDate(getCurrentTimeStamp());
        usersAidsRepository.save(usersAidsEntity);
        return true;
    }
    else {
        throw new NotFoundException("AidEntity with ID " + id + " not
found");
    }
}
```

Listing 3.22 Metoda dokonująca aktualizacje oferty

Omówione powyżej funkcjonalności są dostępne w zakładce 'Aids Offered By Me' (Rys 3. 6. 4)



Rys. 3.6.4 Modal z otwartą zakładka Aids Offered By Me

Elementy zakładki Aids Offered By Me:

1. Przycisk wyświetlający formularz do utworzenia oferty pomocy
2. Przycisk wyświetlający formularz do aktualizacji oferty pomocy
3. Przycisk usuwający oferte pomocy z bazy danych serwera

Użytkownik-wolontariusz ma również możliwość dokonywania odpowiedzi na żądanie oferty pomocy od uchodźcy, czyli może zaakceptować takie żądanie lub odmówić.

Akceptowanie na złożone przez uchodźcy żądania ofert jest zrealizowane za pomocą metody acceptAidRequest (Listing 3.23), która przyjmuje takie parametry jak id żądanej oferty oraz id użytkownika-uchodźcy, zwracając komunikat o sukcesie, gdy operacja jest pomyślnie zakończona

```
@GetMapping("/acceptAidRequest/{aidId}/{userId}")
public String acceptAidRequest(@PathVariable("aidId") Long aidId,
@PathVariable("userId") Long userId) throws NotFoundException {
    try {
        aidService.acceptAidRequest(aidId, userId);
        return "redirect:/home?AidRequestedAccepted";
    } catch (Exception ex) {
        return ex.getMessage();
    }
}
```

Listing 3.23 Metoda przekazująca akceptowanie na żądanie oferty do serwisu

Powyższa metoda również wykorzystywuje warstwę pośrednią do zapisywania interakcji oznaczającej akceptowanie na żądanie oferty do bazy danych (Listing 3.24).

```
@Override
public void acceptAidRequest(Long aidId, Long userId) throws
NotFoundException {
    Optional<AidEntity> optionalAidEntity = aidRepository.findById(aidId);
    if (optionalAidEntity.isPresent()) {
        AidEntity existingAidEntity = optionalAidEntity.get();
        Optional<UserEntity> optionalUserEntity =
                                userRepository.findById(userId);
        if (optionalUserEntity.isPresent()) {
            UsersAidsEntity usersAidsEntity = new UsersAidsEntity();
            UserEntity existingUserEntity = optionalUserEntity.get();
            usersAidsEntity.setAidEntity(existingAidEntity);
            usersAidsEntity.setUserEntity(existingUserEntity);
            usersAidsEntity.setAidInteraction(AidInteraction.ACCEPTANCE);
            usersAidsEntity.setCreatedDate(getCurrentTimeStamp());
            usersAidsRepository.save(usersAidsEntity);
        }
    }
    else {
        throw new NotFoundException("user or aid ids not found");
    }
}
```

Listing 3.24 Metoda serwisowa dokonująca zapis akceptowania na ządanie oferty do bazy danych serwera

W podobny sposób jest zrealizowane odmówienie na żądanie za pomocą metody rejectAidRequest (Listing 3.25)

```
@GetMapping("/rejectAidRequest/{aidId}/{userId}")
public String rejectAidRequest(@PathVariable("aidId") Long aidId,
@PathVariable("userId") Long userId) throws NotFoundException {
    try {
        aidService.rejectAidRequest(aidId, userId);
        return "redirect:/home";
    } catch (Exception ex) {
        return ex.getMessage();
    }
}
```

Listing 3.25 Metoda przekazująca odmówienie na żądanie oferty do serwisu

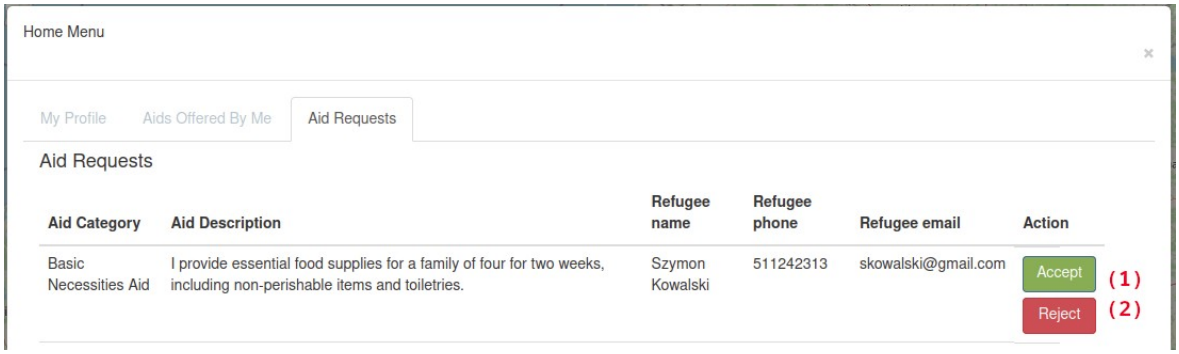
Interakcja odmówienia żądania oferty jest również zapisywana do bazy danych serwera przez warstwę pośrednią (Listing 3.26)

```
@Override
public void rejectAidRequest(Long aidId, Long userId) throws
NotFoundException {
    Optional<AidEntity> optionalAidEntity = aidRepository.findById(aidId);

    if (optionalAidEntity.isPresent()) {
        AidEntity existingAidEntity = optionalAidEntity.get();
        Optional<UserEntity> optionalUserEntity =
                                    userRepository.findById(userId);
        if (optionalUserEntity.isPresent()) {
            UsersAidsEntity usersAidsEntity = new UsersAidsEntity();
            UserEntity existingUserEntity = optionalUserEntity.get();
            usersAidsEntity.setAidEntity(existingAidEntity);
            usersAidsEntity.setUserEntity(existingUserEntity);
            usersAidsEntity.setAidInteraction(AidInteraction.REJECTION);
             usersAidsEntity.setCreatedDate(getCurrentTimeStamp());
```

```
            usersAidsRepository.save(usersAidsEntity);
        }
    }
    else {
        throw new NotFoundException("user or aid ids not found");
    }
}
```

Listing 3.26 Metoda serwisowa dokonująca zapis odmówienia na ządanie oferty do bazy danych serwera

Użytkownik-wolontariusz ma dostęp do ządania ofert utworzonych przez niego w zakładce 'Aids Requests' (Rys. 3.6.5).



Rys. 3.6.5 Modal z otwartą zakładka Aids Requests

Elementy interakcji z żadaniami w zakładce Aids Requests:

1. Przycisk akceptowania żądania oferty pomocy.
2. Przycisk odmówienia żądania oferty pomocy.

3.7 Funkcjonalność uchodźcy

W przypadku, gdy użytkownik posiada rolę 'ROLE_REFUGEE', aplikacja udostępnia informacje dotyczące ilości złożonych przez niego żądań ofert bez odpowiedzi oraz ilości złożonych żądań ofert z otrzymanymi odpowiedziami. Podobnie jak w przypadku funkcjonalności dostępnych dla wolontariuszy, dokonywanie akcji z ofertami pomocy dla użytkownika-uchodźcy jest dostępne w menu głównym (Rys 3.7.1).
W menu głównym użytkownika-uchodźcy pewne elementy są powtórzone z menu głównego użytkownika-wolontariuszy. Wówczas menu główne użytkownika-uchodźcy zawiera takie elementy jak

1. Zakładka z profilem użytkownika
2. Zakładka z rozszerzoną listą ofert pomocy
3. Zakładka z odpowiedziami na żądania ofert pomocy
4. Imię i nazwisko przypisane do konta użytkownika
5. Email przypisany do konta użytkownika
6. Rol przypisana do konta użytkownika
7. Liczba złożonych przez użytkownika-uchodźcy żądań bez odpowiedzi
8. Liczba złożonych przez użytkownika-uchodźcy żądań z odpowiedziami

9. Wykres pokazujący ilość żądań pomocy w czasowym przedziale
10. Wykres pokazujący ilość ofert pomocy w każdym województwie
11. Wykres pokazujący ilość użytkowników-uchodźców w każdym województwie

Rys. 3.7.1 Modal zawierający menu główne użytkownika-uchodźcy

Dane dla wykresu będącego **10** elementem menu-głownego użytkownika-uchodźcy są obliczane za pomocą metody getRegionalDistributionOfAidsForChart

(Listing 3.27) w warstwie serwisowej

```
@Override
public Map<String, Long> getRegionalDistributionOfAidsForChart() {
    List<AidEntity> aids = aidRepository.findAll();
    List<RegionEntity> regions = getRegions();
    Map<String, Long> mapOfAidsRegionalDistribution  = new HashMap<>();
    for (RegionEntity regionEntity : regions) {
        mapOfAidsRegionalDistribution.put(regionEntity.getName(), 0L);
    }
    for (AidEntity aidEntity : aids) {
        for (String region : mapOfAidsRegionalDistribution.keySet()) {
            if (aidEntity.getAddress().contains(region)) {
             mapOfAidsRegionalDistribution.compute(region, (k, v) -> v + 1);
            }
        }
    }
    return mapOfAidsRegionalDistribution;
}
```

Listing 3.27 Metoda serwisowa zliczająca ilośc ofert pomocy w każdym województwie

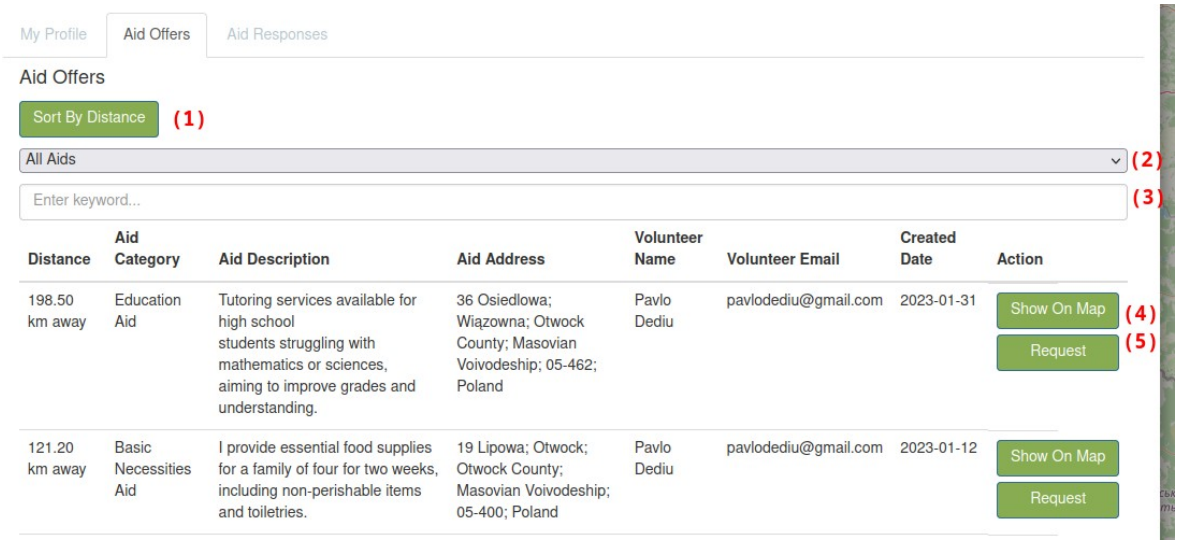
Zliczanie jest dokonane w programistyczny sposób iterując po liście ofert pomocy i sprawdzając zgodność pomiędzy polem zawierającym adres a nazwą województwa zawartą w bazie danych.

Wykresy zawarte w rozdziale 3.6, 3.7 są generowane za pomocą darmowej biblioteki canvasJS (Listing 3.28)

```
   fetch('/getRegionalDistributionOfAids')
      .then(response => response.json())
      .then(data => {
      const dataPoints = Object.keys(data).map(region => ({
         label: region,
         y: data[region]
      }));
      const chart = new CanvasJS.Chart("chartContainer2", {
         title: {
            text: "Regional Distribution of Aids ",
         },
...
         data: [{
            type: "bar",
            indexLabel: "{y}",
            dataPoints: dataPoints
         }]
      });
      chart.render();
   })
      .catch(error => {
      console.error('Error fetching data:', error);
   });
```

Listing 3.28 Przykład użycia biblioteki CanvasJS dla wykresu zliczającym ilość ofert w województwach

Użytkownilk-uchodźca ma możliwosc dokonywania żądania oferty pomocy, przeglądania oraz fitrowania listy takich oferty w zakładce Aid Offers (Rys. 3.7.2)



Rys. 3.7.2 Modal z otwartą zakładka Aid Offers

Elementy interakcji z listą rozszerzoną w zakładce Aids Offers:

1. Sortowanie ofert według odległości od użytkownika
2. Filtrowanie ofert według kategorii
3. Wyszukiwanie ofert według słów kluczowych
4. Przycisk wyszukiwania oferty na mapie
5. Przycisk żadania oferty

Dla pomiaru odległości pomiędzy ofertą pomocy a użytkownikiem jest używane Geolocation API który za pomocą przeglądarki pobiera współrzedne użytkownika,

a następnie na podstawie takich współrzednych jest obliczana odległość (Listing 3.29)

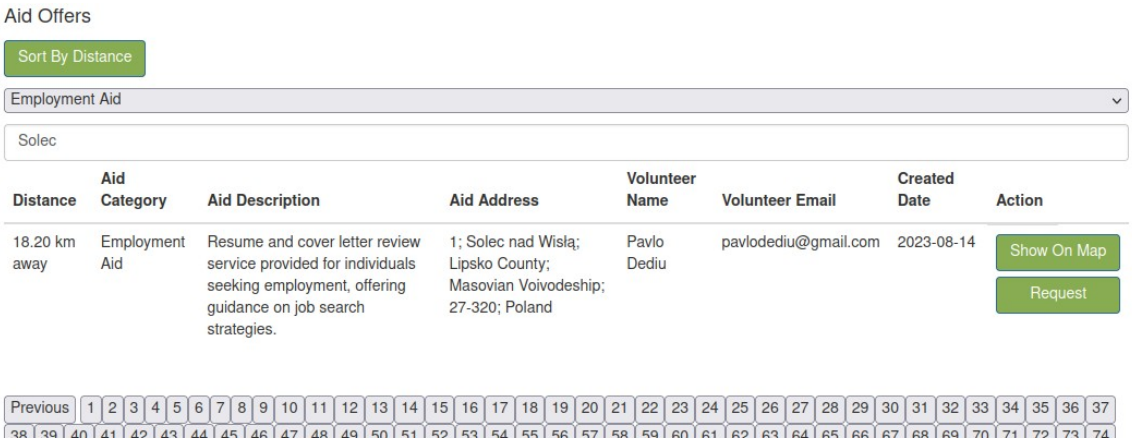
```
navigator.geolocation.getCurrentPosition(function(position) {
   var userLatLng = L.latLng(position.coords.latitude,
position.coords.longitude);
   markerlist.forEach(function(marker) {
      var markerLatLng = L.latLng(marker.getLatLng().lat,
marker.getLatLng().lng);
      var markerLatitude = marker._latlng.lat;
      var markerLongitude = marker._latlng.lng;
      var distanceCellId = 'distId-' + markerLatitude + '-' +
markerLongitude;
      var distanceCell = document.getElementById(distanceCellId);
      if (distanceCell) {
         var distance = userLatLng.distanceTo(markerLatLng);
         distanceCell.innerText = (distance/ 1000).toFixed(2) + ' km away';
      }
   });
});
```

Listing 3.29 Przykład użycia Geolocation API

Identyfikator elementu listy ofert jest generowany na podstawie współrzędnych samej oferty, takie rozwiązanie ułatwia wyświetlanie liczbę odległośći dla każdego elementu w liście, zatem sortwanie jest zrealizowanie za pomocą funkcji sortByDistance(Listing 3.30)

```
...
<td th:id="'distId-' + ${aid.latitude} + '-' + ${aid.longitude}">
<span></span></td>
...
function sortByDistance() {
   const tableBody = document.getElementById('tableBody');
   const rows = Array.from(tableBody.querySelectorAll('tr'));
   rows.sort(function(a, b) {
const distance1 = parseFloat(a.querySelector('td').innerText.replace(' km
away', ''));
const distance2 = parseFloat(b.querySelector('td').innerText.replace(' km
away', ''));
      return distance1 - distance2;
   });
   rows.forEach(function(row) {
      tableBody.appendChild(row);
   });
}
```

Listing 3.30 Przykład użycia Geolocation API



Rys. 3.7.3 Wynik wyszukiwania oferty

Żądanie oferty jest dokonywane za pomocą metody GET requestAid (Listing 3.31) zawierająca parametr id oferty pomocy

```
@GetMapping("/requestAid/{id}")
public String requestAid(@PathVariable("id") Long id) throws
NotFoundException  {
    if (aidService.countRequestedAidByUser(id) < 1) {
        aidService.requestAid(id);
        return "redirect:/home?AidRequestedSuccessfull";
    }
        return "redirect:/home";
}
```

Listing 3.31 Metoda dokonująca żądanie oferty pomocy

Następnie jest wywoływana metoda requestAid (Listing 3.23) dokonująca zapis interakcji żądania dla oferty pomocy z określonym id.

```
@Override
public void requestAid(Long id) throws NotFoundException {
    Optional<AidEntity> optionalAidEntity = aidRepository.findById(id);

    if (optionalAidEntity.isPresent()) {
        AidEntity existingAidEntity = optionalAidEntity.get();
        UserEntity userEntity =
                  userService.findByEmail(SecurityUtil.getSessionUser());
        UsersAidsEntity usersAidsEntity  = new UsersAidsEntity();
        usersAidsEntity.setAidEntity(existingAidEntity);
        usersAidsEntity.setUserEntity(userEntity);
        usersAidsEntity.setAidInteraction(AidInteraction.REQUESTING);
        usersAidsEntity.setCreatedDate(getCurrentTimeStamp());
        usersAidsRepository.save(usersAidsEntity);
    }
    else {
        throw new NotFoundException("AidEntity with ID " + id + " not
found");
    }
}
```

Listing 3.23 Metoda serwisowa dokonująca zapis żądania oferty pomocy

4. Testy aplikacji

Testowanie aplikacji jako proces pozwoli wykryć błędy i niedociągnięcia w aplikacji, zanim zostanie taka aplikacja upubliczniona, zapewni jakość oprogramowania poprzez sprawdzenie, czy spełnia ono wymagania i oczekiwania użytkowników.

4.1 Testy Jednostkowe

Do przeprowadzenia testów wykorzystano framework Mockito, który pozwala na tworzenie obiektów mockowych w izolacji od ich zależności. Pierwszą funkcjonalnością, która została przetestowana, jest proces uwierzytelnienia i autoryzacji. Testy jednostkowe zapewniają, że funkcje zabezpieczeń działają zgodnie z oczekiwaniami, uniemożliwiając nieautoryzowany dostęp do zasobów.

Poniższa klasa testowa (Listing 4.1) sprawdza działanie metody loadUserByUsername zdefiniowanej w klasie CustomUserDetailsService, która ma za zadanie odnaleźć użytkownika na podstawie adresu e-mail i zwrócić UserDetails (szczegóły użytkownika) lub zgłosić wyjątek, jeśli użytkownik nie zostanie znaleziony.

```
@ExtendWith(MockitoExtension.class)
class CustomUserDetailsServiceTest {
    @Mock
    private UserRepository userRepository;
    @InjectMocks
    private CustomUserDetailsService customUserDetailsService;
    @Test
    void shouldToReturnUserDetails() {
        UserEntity userEntity = new UserEntity();
        userEntity.setId(1L);
        userEntity.setEmail("test@example.com");
        userEntity.setPassword("password123123");
        userEntity.setRoleEntity(new RoleEntity("ROLE_REFUGEE", 1L));

when(userRepository.findByEmail("test@example.com")).thenReturn(userEntity);

        UserDetails userDetails =
             customUserDetailsService.loadUserByUsername("test@example.com");

        assertNotNull(userDetails);
        assertEquals("test@example.com", userDetails.getUsername());
        assertEquals("password123123", userDetails.getPassword());
        assertTrue(userDetails.getAuthorities().stream()
                .anyMatch(a -> a.getAuthority().equals("ROLE_REFUGEE")));
    }
    @Test
    void shouldToThrowUsernameNotFoundException() {
        when(userRepository.findByEmail(anyString())).thenReturn(null);

        assertThrows(UsernameNotFoundException.class,
() ->
customUserDetailsService.loadUserByUsername("nonexistent@example.com"));
    }}
```

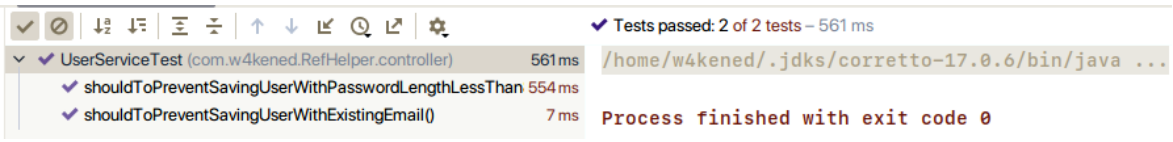
Listing 4.1 Klasa testująca uwierzytelnianie użytkownika

Rys. 4.1.1 Wyniki testów klasy CustomUserDetailsServiceTest

W kolejnej klasie testowej (Listing 4.2) został przetestowany proces rejestracji użytkownika, który powinien być odporny na próbę zapisania konta z istniejącym adresem e-mail lub na zapisanie konta, gdy długość hasła jest mniejsza 8 znaków.

```
@ExtendWith(MockitoExtension.class)
public class UserServiceTest {
    @Mock
    private UserRepository userRepository;
    @Mock
    RoleRepository roleRepository;
    @Mock
    PasswordEncoder passwordEncoder;
    @Mock
    CityRepository cityRepository;
    @InjectMocks
    private UserServiceImpl userServiceImpl;
    @Test
    void shouldToPreventSavingUserWithExistingEmail() {
        when(userRepository.findByEmail("existingEmail@gmail.com"))
                .thenReturn(new UserEntity("existingEmail@gmail.com",
"password123"));

        UserDto newUser = UserDto.builder()
                        .firstName("Jan")
                        .lastName("Kowalski")
                        .email("existingEmail@gmail.com")
                        .password("123123")
                        .cityName("Warszawa")
                        .build();
        boolean isSaved = userServiceImpl.saveUser(newUser);

        assertFalse(isSaved);
    }
    @Test
    void shouldToPreventSavingUserWithPasswordLengthLessThan8() {
        UserRepository userRepository = mock(UserRepository.class);
        UserService userService = new UserServiceImpl(userRepository,
                        roleRepository, passwordEncoder, cityRepository);

        UserDto userWithShortPassword = new UserDto();
        userWithShortPassword.setPassword("1234567");
        boolean isSaved = userService.saveUser(userWithShortPassword);

        assertFalse(isSaved);
    }
}
```

Listing 4.2 Klasa testująca rejestracje użytkownika

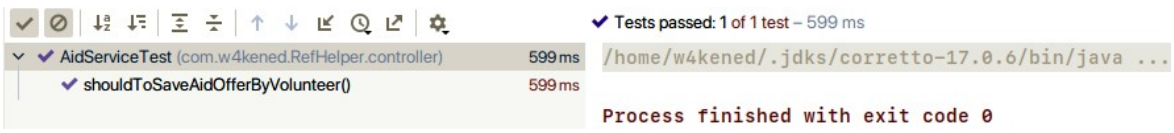


Rys. 4.1.2 Wyniki testów klasy UserServiceTest

Następna klasa zawiera metody shouldToSaveAidOfferByVolunteer (Listing 4.3), shouldToPreventSavingAidOfferByRefugee (Listing 4.4) sprawdzające zapis oferty przez użytkowników które posiadają różne role

```
@BeforeEach
void setUp() {
    MockitoAnnotations.openMocks(this);
    Authentication authentication = mock(Authentication.class);
    when(authentication.getName()).thenReturn("test@example.com");
    SecurityContext securityContext = mock(SecurityContext.class);
    when(securityContext.getAuthentication()).thenReturn(authentication
    SecurityContextHolder.setContext(securityContext);
    UserEntity userEntity = new UserEntity();
    userEntity.setId(1L);
    userEntity.setEmail("test@example.com");
    userEntity.setRoleEntity(new RoleEntity("ROLE_VOLUNTEER", 2L));
 when(userService.findByEmail("test@example.com")).thenReturn(userEntity);
}
@Test
void shouldToSaveAidOfferByVolunteer() throws NotFoundException {
    AidDto aidDto = new AidDto();
    aidDto.setId(5L);
    aidDto.setDescription("Test Description");
    aidDto.setAddress("Test Address");
    aidDto.setLatitude(123.45);
    aidDto.setLongitude(67.89);
    aidDto.setSelectedCategoryAid(1L);

    boolean result = aidService.saveAid(aidDto);

    assertTrue(result);
}
```

Listing 4.3 Metoda testowa shouldToSaveAidOfferByVolunteer



Rys. 4.1.3 Wyniki testów metody shouldToSaveAidOfferByVolunteer

```java
@BeforeEach
void setUp() {
    MockitoAnnotations.openMocks(this);
    Authentication authentication = mock(Authentication.class);
    when(authentication.getName()).thenReturn("test@example.com");
    SecurityContext securityContext = mock(SecurityContext.class);
    when(securityContext.getAuthentication()).thenReturn(authentication
    SecurityContextHolder.setContext(securityContext);
    UserEntity userEntity = new UserEntity();
    userEntity.setId(1L);
    userEntity.setEmail("test@example.com");
    userEntity.setRoleEntity(new RoleEntity("ROLE_REFUGEE", 2L));
 when(userService.findByEmail("test@example.com")).thenReturn(userEntity);
}
@Test
void shouldToPreventSavingAidOfferByRefugee() throws NotFoundException {
    AidDto aidDto = new AidDto();
    aidDto.setId(5L);
    aidDto.setDescription("Test Description");
    aidDto.setAddress("Test Address");
    aidDto.setLatitude(123.45);
    aidDto.setLongitude(67.89);
    aidDto.setSelectedCategoryAid(1L);

    boolean result = aidService.saveAid(aidDto);

    assertFalse(result);
}
```

Listing 4.4 Metoda testowa shouldToPreventSavingAidOfferByRefugee

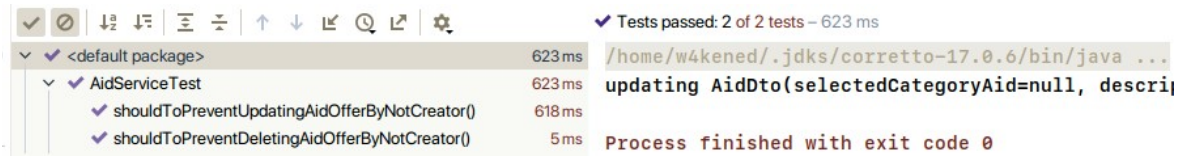
Rys. 4.1.4 Wyniki testów metody shouldToPreventSavingAidOfferByRefugee

Metody testowe shouldToPreventUpdatingAidOfferByNotCreator , shouldToPreventDeletingAidOfferByNotCreator (Listing 4.5) demonstrują że uprawnienia do usuwania lub aktualizacji oferty są dostępne wyłącznie dla twórcy tworzonej oferty

```java
@BeforeEach
void setUp() {
    MockitoAnnotations.openMocks(this);
    Authentication authentication = mock(Authentication.class);
    when(authentication.getName()).thenReturn("test@example.com");
    SecurityContext securityContext = mock(SecurityContext.class);
    when(securityContext.getAuthentication()).thenReturn(authentication);
    SecurityContextHolder.setContext(securityContext);

    UserEntity userEntity = new UserEntity();
    userEntity.setId(1L);
    userEntity.setEmail("test@example.com");
    userEntity.setRoleEntity(new RoleEntity("ROLE_VOLUNTEER", 2L));
    when(userService.findByEmail("test@example.com")).thenReturn(userEntity);
```

```
    AidDto aidDto = new AidDto();
    aidDto.setId(5L);
    aidDto.setDescription("Test Description");
    aidDto.setAddress("Test Address");
    aidDto.setLatitude(123.45);
    aidDto.setLongitude(67.89);
    aidDto.setSelectedCategoryAid(1L);

    AidEntity mockedAidEntity = new AidEntity();
    mockedAidEntity.setId(5L);

when(aidRepository.findById(5L)).thenReturn(Optional.of(mockedAidEntity));

    aidService.saveAid(aidDto);
}
@Test
void shouldToPreventUpdatingAidOfferByNotCreator() throws NotFoundException {
    UserEntity anotherUserEntity = new UserEntity();
    anotherUserEntity.setId(2L);
    anotherUserEntity.setEmail("test2@example.com");
    anotherUserEntity.setRoleEntity(new RoleEntity("ROLE_VOLUNTEER",2L));

    AidDto existingAidDto = new AidDto();
    existingAidDto.setId(5L);
    existingAidDto.setDescription("Changed Description");
    existingAidDto.setAddress("Changed Address");
    boolean isUpdated = aidService.updateAid(existingAidDto, 5L);

    assertFalse(isUpdated);
}
@Test
void shouldToPreventDeletingAidOfferByNotCreator() throws NotFoundException {
    UserEntity anotherUserEntity = new UserEntity();
    anotherUserEntity.setId(2L);
    anotherUserEntity.setEmail("test2@example.com");
    anotherUserEntity.setRoleEntity(new RoleEntity("ROLE_VOLUNTEER",2L));

    AidDto existingAidDto = new AidDto();
    existingAidDto.setId(5L);
    existingAidDto.setDescription("Changed Description");
    existingAidDto.setAddress("Changed Address");
    boolean isUpdated = aidService.deleteAidById(5L);

    assertFalse(isUpdated);
}
```

Listing 4.5 Metody testujące działanie uprawnień dla obcych użytkowników



Rys. 4.1.5 Wyniki metod testujących uprawnienia

4.2 Testy integracyjne

Testy integracyjne są ważnym elementem kompleksowej strategii testowej, zapewniając większą pewność co do funkcjonowania systemu w realnym środowisku przez sprawdzenie pracy między warstwami aplikacji.
Klasa HomeControllerTest przedstawia testy obciążeniowe które sprawdzają zachowanie serwera podczas przetwarzania tysięcy zapytań bazodanowych.

Metody testowe shouldToEnterAllUsersHomePageWithWrongCredentials oraz shouldToEnterAllUsersHomePageWithMatchingCredentials (Listing 4.6) dokonują wyświetlanie strony głównej przez 200 użytkowników

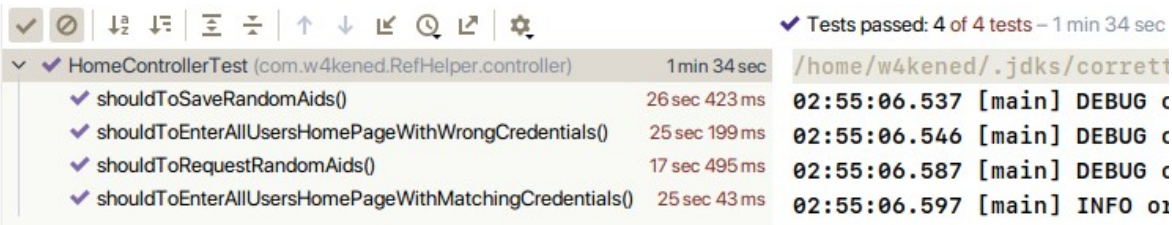
```
@SpringBootTest
@AutoConfigureMockMvc
public class HomeControllerTest {
...
    @BeforeEach
    public void setUp() {
        this.listOfUsers = generateMultipleUserDTOs();
        for (UserDto userDto : listOfUsers) {
            userService.saveUser(userDto);}
    }
    @Test
    public void shouldToEnterAllUsersHomePageWithWrongCredentials() throws
Exception {
        int dmlCounter = 0;
        List<UserDto> listOfUsers = generateMultipleUserDTOs();
        Map<Boolean, UserDto> mapOfSelects = new HashMap<>();
        for (UserDto userDto : listOfUsers) {
            mockMvc.perform(post("/login")
                            .param("email", userDto.getEmail())
                            .param("password", userDto.getPassword()+1))
                    .andExpect(status().is3xxRedirection())
                    .andExpect(redirectedUrl("/login?error=true"));
            dmlCounter++;
        }
        assertThat(dmlCounter).isEqualTo(200);
    }
    @Test
    public void shouldToEnterAllUsersHomePageWithMatchingCredentials() throws
Exception {
        int dmlCounter = 0;
        for (UserDto userDto : listOfUsers) {
            mockMvc.perform(post("/login")
                            .param("email", userDto.getEmail())
                            .param("password", userDto.getPassword()))
                    .andExpect(status().is3xxRedirection())
                    .andExpect(redirectedUrl("/home"));
            dmlCounter++;
        }
        assertThat(dmlCounter).isEqualTo(200);
    } ...
}
```

Listing 4.6 Metody testujące wyświetlanie strony głównej przez 200 użytkowników z poprawnymi oraz nieprawidłowymi hasłami.

Metody testowe (Listing 4.6) dokonujące zapisu dużej ilości ofert oraz zapisu zgłoszeń ofert

```
@Test
    public void shouldToSaveRandomAids() throws Exception {
        List<AidDto> aidDtos = generateMultipleAidDTOs();
        int dmlCounter = 0;
        for (AidDto aidDto : aidDtos) {
            for (UserDto userDto : listOfUsers) {
                if (userDto.getSelectedRole() == 2) {
                    mockMvc.perform(post("/addAid")
                                    .with(csrf())
                                    .flashAttr("aidDto", aidDto)
                                    .with(user(userDto.getEmail())))
                            .andExpect(status().is3xxRedirection())
                            .andExpect(redirectedUrl("/home"));
                    dmlCounter++;
                }
            }
        }
        assertThat(dmlCounter).isEqualTo(1600);
    }
    @Test
    public void shouldToRequestRandomAids() throws Exception {
        List<AidEntity> aidEntities = aidService.findAll();
        for (AidEntity aidEntity : aidEntities) {
            String getEndpoint = "/requestAid/" + aidEntity.getId();
            UserDto randomuser = listOfUsers.get((int)
                                        (Math.random() * listOfUsers.size()));
            if (randomuser.getSelectedRole() == 3) {
                mockMvc.perform(get(getEndpoint)
                                .with(csrf())
                                .with(user(randomuser.getEmail())))
                        .andExpect(status().is3xxRedirection())
                        .andExpect(redirectedUrl(
                                "/home?AidRequestedSuccessfull"));
            }
        }
    }
```

Listing 4.6 Metody testujące zapis ofert pomocy oraz zapis żadania oferty pomocy z dużo ilością wywołań DLL zapytań.



Rys 4.2.6 Wyniki testów obciążeniowych

Jak widać na wynikach (Rys 4.2.6), przeprowadzone testy obciążeniowe pozwoliły zaobserwować prawidłowe współdziałanie komponentów aplikacji między sobą oraz rzeczywiste zachowanie systemu pod względem jego wydajności oraz odporności na duże obciążenie.

5. Podsumuwanie

Celem pracy było opracowanie aplikacji webowej w postaci „Portalu Informacyjnego Wspomagającego Uchodźców”, który skupiał się na udostępnianiu informacji dotyczących pomocy dla uchodźców oraz umożliwiał wolontariuszom efektywne dostarczanie informacji pomocy poprzez aplikacje.

Wykonana aplikacja umożliwiała uchodżcom na efektywne wyszukiwanie informacji dotyczącej pomocy w postaci ofert oraz pozwolała na klarowne odnalezienie takich oferty za pomocą mapy. Warstwa pośrednia aplikacji zapewniała wolontariuszom tworzenie oraz zarządzanie ofertami pomocy zgodnie z założeniami aplikacji.
Wykorzystywana darmowa biblioteka Leaflet okazała się bardzo użyteczną ze względu na ilość zaprezentowanych modułów pozwalających na geowyszukiwanie oraz optymalizacje wyświetlania znaczników na mapie interaktywnej.
Podczas implementacji projektu były dokonane próby integracji systemów rozkładu jazdy komunikacji publicznej do wyświetlania rozkladu jazdy autobusów według adresów przypisanych do ofert pomocy. Jednakże, mimo podjętych prób, integracja okazała się niemożliwa ze względu na decentralizowaną bazę danych rozkładów jazdy oraz zamknięty dostęp do API.

Wówczas wykonywana aplikacja zapewniała użytkowników statystykami w postaci wykresów informujących o ilości zgłoszonych żądań pomocy w aplikacji, oraz wykresów pozwalających na monitorowanie regionów dotkniętych napływem użytkowników-uchodźców.

Wykorzystywany Spring Framework pozwolił w łatwy sposób organizować oraz zabiezpieczać komponenty aplikacji z możliwościa skalowania oraz przyszłego rozwoju aplikacji. Rozwinięcie nowych funkcjonalności aplikacji stanowiło kluczowy etap w procesie rozwoju projektu. Kolejną taką funkcjonalnością, którą mogłaby się pojawić, jest wprowadzenie czatu pomiędzy uchodźcami ubiegającymi się o ofertę pomocy a wolontaiuszem który utworzył taką ofertę.

Podsumowując, aplikacja RefHelper może stanowić bardzo wygodną i przydatną alternatywę dla osób ubiegających się o pomoc, zapewniając im prosty i szybki dostęp do niezbędnych informacji oraz ułatwiając interakcje z wolontariuszami czy organizacjami oferującymi wsparcie.

Literatura

[1] https://data2.unhcr.org/en/situations/ukraine/location?secret=unhcrrestricted [*dostęp 2022-03-19]*

[2] https://stat.gov.pl/download/gfx/portalinformacyjny/pl/defaultaktualnosci/5491/3/1/1/wolontariat_w_2022.pdf

[3] https://refaid.com/

Spis rysunków