AKADEMIA TARNOWSKA
Wydział Politechniczny

Kierunek: ***Informatyka***

Specjalność: Informatyka Stosowana
Specjalizacja: Inżynieria Oprogramowania

2023/2024

Michał Baka

PRACA INŻYNIERSKA

Projekt i implementacja aplikacji webowej umożliwiającej publikowanie i przeglądanie ogłoszeń z lokalami do wynajęcia

Promotor pracy:
mgr inż. Tomasz Gądek

Tarnów, 2024

Spis treści

1. Wstęp

Przedmiotem niniejszej pracy dyplomowej była aplikacja internetowa, będąca serwisem ogłoszeniowym z lokalami do wynajęcia. W kolejnych rozdziałach przybliżone zostały wymagania, które powinien spełniać system, technologie wykorzystane przy jego budowie, wybrane elementy implementacyjne oraz uzyskane efekty.

1.1. Motywacja

Przeciętna osoba żyjąca w XXI wieku nie ma problemu, by przemieszczać się po względnie niewielkich obszarach jak na przykład obręb danego województwa. Wyruszenie za granicę również nie stanowi większego problemu [5]. Rozwój różnego rodzaju transportu publicznego, indywidualnego oraz Internetu, sprawił, że świat stał się mniejszy. Czy to w celach rekreacyjnych, biznesowych, czy po prostu w poszukiwaniu nowego miejsca do zamieszkania przemieszczanie stało się istotną częścią życia. Posiadanie komfortowego lokum jest kluczowym aspektem planowania podróży i relokacji. Dodatkowo, wiele osób nie wyobraża sobie wakacji bez podróży. Ponownie, takie wyjazdy wiążą się z koniecznością zapewnienia sobie odpowiedniego miejsca do noclegu. Hotel jest popularną i wygodną opcją. Jednak warto zaznaczyć, że hotele zazwyczaj są projektowane tak, aby podobać się jak największej liczbie gości. Z tego powodu niektóre osoby mogą nie poczuć się jak w domu przebywając w takim miejscu. Jednak istnieje sposób by temu zaradzić: wynajem prywatnego mieszkania. Jest to rozwiązanie, które nabrało popularności w ciągu ostatnich kilkunastu lat. W ten prosty sposób ludzie, nawet gdy podróżują, mogą poczuć się jak u siebie w domu. Dzięki gwałtownemu rozwojowi tego rynku pojawiło się zapotrzebowanie na usługi, które w łatwy i szybki sposób umożliwią wynajęcie lokalu. Potrzeba również zaoferować funkcjonalne narzędzia osobom chcącym wystawić swój lokal by inni mogli go wynająć.

1.2. Cel pracy

Celem niniejszej pracy było zaprojektowanie i zaimplementowanie aplikacji, która umożliwi wynajmowanie lokali, ich ocenę, ale też wystawianie ogłoszeń. Najbardziej podstawowa funkcjonalność, czyli przeglądanie ogłoszeń powinna być dostępna dla wszystkich użytkowników tj. niezalogowanych i tych zalogowanych. System po zalogowaniu udostępnia kolejne opcje: wynajmu oraz umieszczenia nowego ogłoszenia. Osoba, która wynajęła lokal powinna mieć możliwość zmiany daty pobytu, anulowania go oraz recenzji danego lokalu.

Jeśli zostanie wybrana druga opcja, użytkownik może dodać ogłoszenie, edytować je lub usunąć. Dodatkowo, powinien być dostępny mechanizm umożliwiający zarządzanie zaplanowanymi pobytami, co oznacza możliwość edycji daty ich rozpoczęcia i zakończenia lub całkowitego anulowania wraz z podaniem przyczyny.

1.3. Zakres pracy

W ramach projektu inżynierskiego przygotowany został system składający się z:

- aplikacji serwerowej,
- bazy danych,
- aplikacji internetowej.

2. Analiza biznesowa

W niniejszym rozdziale przybliżone zostaną wymagania funkcjonalne i niefunkcjonalne, jakie powinna spełniać aplikacja.

2.1. Wymagania funkcjonalne

Podstawowe funkcjonalność systemu to możliwość wstawiania, przeglądania i odpowiadania na ogłoszenia.

2.1.1. Umieszczanie ogłoszeń

Aplikacja powinna umożliwiać umieszczenie ogłoszenia z lokalem do wynajęcia każdemu zalogowanemu użytkownikowi. Ogłoszenie powinno posiadać atrybuty, które udostępnią użytkownikowi dokładne informacje na temat interesującego go lokalu. Niektóre z tych atrybutów to:

- nazwa lokalu,
- cena wynajmu za noc,
- adres,
- ilość pokoi,
- zdjęcia lokalu.

Użytkownik powinien mieć również kontrolę nad swoimi ogłoszeniami tj. powinien móc je edytować lub całkowicie usunąć.

2.1.2. Przeglądanie ogłoszeń

Kolejna funkcjonalność to przeglądanie ogłoszeń znajdujących się w serwisie wraz z możliwością filtrowania wyników, na przykład według ceny za noc oraz kraju. Powiązane z przeglądaniem ogłoszeń jest ich proponowanie użytkownikom. Do tego celu może zostać wykorzystany technika znana jako *Collaborative filtering* [14]. Rekomendacje są nieaktywne dla użytkowników niezalogowanych.

2.1.3. Wynajmowanie lokalu

Zalogowany użytkownik powinien mieć możliwość odpowiedzi na ogłoszenie w postaci wynajęcia danego lokalu. Wynajem powinien być na określony okres czasu i dla określonej liczby osób z możliwością jego późniejszej zmiany. Szczególna uwaga powinna zostać poświęcona by okresy wynajmu nie nachodziły na siebie. Użytkownik w razie potrzeby ma możliwość aktualizacji wynajmu, jeśli tylko nie został on anulowany lub zakończony. Inną powiązaną akcją jest anulowanie wynajmu.

2.1.4. Recenzja lokalu

Zalogowany użytkownik może napisać recenzję danego lokalu pod warunkiem, że wcześniej go wynajął. Inni użytkownicy mogą ocenić, czy dana recenzja była pomocna lub zgodna z prawdą.

2.1.5. Rejestracja oraz logowanie

Obsługa rejestracji oraz logowania użytkownika umożliwia działanie większości wymienionych funkcjonalności. W ten sposób użytkownikowi zostanie zapewnione większe bezpieczeństwo. Ponadto po zalogowaniu ma się dostęp do pełnej wersji serwisu, więc jest to dodatkowa zachęta dla użytkownika, aby takie konto założyć.

2.2. Wymagania niefunkcjonalne

W niniejszym podrozdziale omówione zostaną wymagania niefunkcjonalne. Wymagania niefunkcjonalne służą zapewnieniu wysokiej jakości doświadczenie użytkownikowi korzystającemu z aplikacji [11].

Zaczynając od obsługowości. System nie jest skierowany do ściśle określonej grupy osób. Mogą z niego korzystać osoby w różnym wieku i o różnym stopniu zaawansowania w posługiwaniu się takimi urządzeniami jak komputer, czy smartfon. Z tego powodu aplikacja powinna mieć prosty i czytelny interfejs. Każda akcja powinna być czytelnie oznaczona, a samo jej wykonanie możliwie najprostsze.

Niezawodność to kolejna rzecz nad którą należy się pochylić. System powinien działać stabilnie i wydajnie, aby doświadczenie użytkowników było jak najlepsze. Ponadto mnóstwo osób, które będą udostępniać swoje lokale, może traktować tę działalność jako ważną część swojego dochodu. Jakikolwiek przestój w działaniu lub inne nieprawidłowe funkcjonowanie sprawi, że użytkownicy przestaną korzystać z aplikacji, a to spowoduje straty dla wspomnianej grupy osób.

Ostatnią rzeczą, na którą należy zwrócić uwagę to bezpieczeństwo. Za bezpieczeństwo aplikacji powinien odpowiadać system autoryzacji, uwierzytelnienie i rejestracji. Rejestracja użytkownika odbywa poprzez podanie następujących danych:

- imię i nazwisko,
- nazwa użytkownika,
- hasło,
- email.

Po udanej rejestracji wysłana zostanie wiadomość email z linkiem aktywującym konto. W tym momencie proces rejestracji kończy się i można przejść do uwierzytelnienia. Proces ten służy weryfikacji tożsamości użytkownika i odblokowuje przed nim kolejne funkcje systemu.

3. Opis wykorzystanych technologii

Niniejszy rozdział posłuży przedstawieniu technologii, które wchodzą w skład systemu będącego przedmiotem pracy.

3.1. Java

Java to to silnie typowany i obiektowy język programowania, którego pierwsza wersja trafiła na rynek w 1996 roku. Powstały dzięki pracy Mike'a Sheridana, Patrick Naughtona i Jamesa Goslinga z firmy Sun Microsystems. Język miał być odpowiedzią na potrzeby programistów, którzy szukali narzędzia przyjaznego użytkownikowi, o prostej składni i bogatej bibliotece. Mając to na uwadze twórcy opracowali listę 11 kluczowych elementów, które powinna zawierać Java. Oto kilka z nich [1]:

- prostota,
- obiektowość,
- niezawodność,
- przenośność
- wielowątkowość.

3.2. Spring

Spring to platforma programistyczna dla języka Java (może wspierać też inne języki działające w oparciu o JVM [12]). Sam w sobie Spring to kontener IoC (ang. *Inversion Of Control* – odwrócenie sterowania). Zarządza on komponentami na które składa się aplikacja (ang. *beans*). Elementy te są ze sobą wiązane dzięki *wstrzykiwaniu zależności*. Dzięki temu komponenty nie muszą zarządzać cyklem życia innych komponentów od których zależą. Zajmuje się tym *kontekst aplikacji Spring*, czyli wspomniany wcześniej kontener [2]. Podejście to bardzo ułatwia testowanie aplikacji oraz poprawia jej modularność.
W samej aplikacji nie został wykorzystany "czysty" Spring, a jego rozwinięcie, czyli Spring Boot. Spring Boot oferuje przede wszystkim autokonfigurację, która działa w oparciu m.in. o to, co znajduje się w ścieżce klasy oraz zmienne środowiskowe. Programista oszczędza dzięki temu mnóstwo czasu, który spędziłby na konfigurację przy użyciu plików XML [2].

Podrozdziały od 3.2.1 do 3.2.3 przybliżą najważniejsze tzw. *startery Spring Boot* użyte w aplikacji serwerowej.

3.2.1. Spring Data Neo4j

Spring Data Neo4j (SDN) to moduł Spring Data służący do komunikacji z bazą danych Neo4j. Ponadto oferuje funkcję mapowania klas na graf wykorzystywany w bazie danych.

SDN oferuje trzy poziomy abstrakcji:

- Neo4j Client,
- Neo4j Template,
- Neo4j Repository.

Neo4j Client, w przeciwieństwie do dwóch kolejnych poziomów, jest przeznaczony dla użytkowników potrzebujących niskopoziomowego dostępu do bazy danych [6].

Kluczowe cechy SDN to:

- wsparcie dla paradygmatu imperatywnego oraz reaktywnego,
- możliwość pisania zapytań przy użyciu *@Query*,
- pełne wsparcie dla stałych encji.

3.2.2. Spring WebFlux

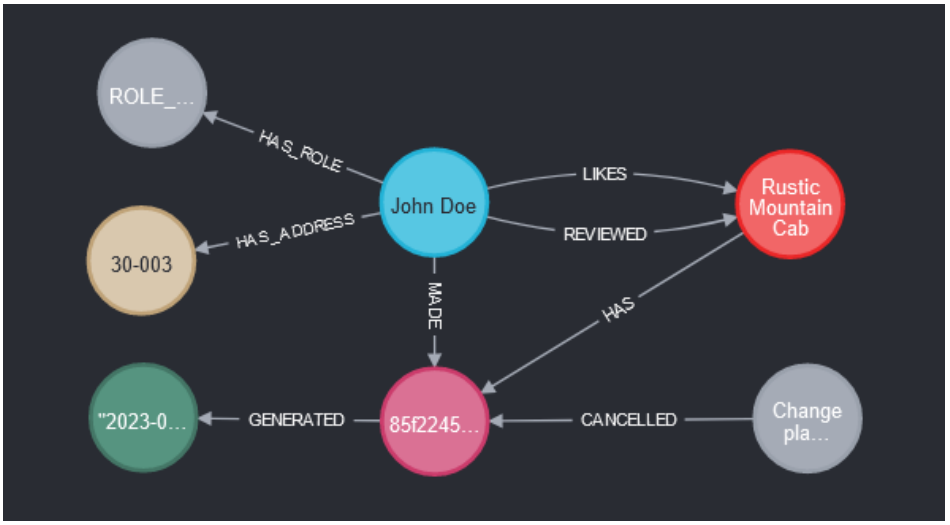
Spring WebFlux to framework do budowy aplikacji internetowych podobnie jak Spring Web MVC. Istotną różnicą jest to, że Spring WebFlux jest nieblokujący, czyli pozwala obsługiwać wiele żądań równocześnie, bez blokowania wątków. Do pracy z tym rozwiązaniem wykorzystywany jest reaktywny paradygmat programowania. Spring WebFlux działa w oparciu o Project Reactor, jednak programista może użyć innej implementacji Reactive Streams [2][9].

3.2.3. Spring Security

Spring Security to narzędzie, które służy do zabezpieczania aplikacji poprzez autoryzację i uwierzytelnienie użytkowników. Oprócz tego, dostarcza mechanizmy obrony przed atakami takimi jak Cross-Site Request Forgery (CSRF), Cross-Site Scripting (XSS) i wiele innych. Jest to standardowy sposób ochrony aplikacji opartych na frameworku Spring, zarówno tych działających w oparciu o serwlety, jak i te o charakterze reaktywnym. Spring Security umożliwia także definiowanie niestandardowych reguł zabezpieczeń i dostosowywanie mechanizmów uwierzytelniania do potrzeb konkretnego projektu [2][10].

3.3. Neo4j

Neo4j to system zarządzania bazą danych typu grafowego. Napisany w języku Java, Neo4j wykorzystuje graf jako sposób przechowywania danych, w odróżnieniu od niektórych innych rozwiązań, które mogą zapisywać graf w postaci na przykład relacyjnej bazy danych. Dane są przechowywane w postaci dynamicznych węzłów z nadaną etykietą. Połączenia między danymi odbywa się za pomocą relacji o określonej nazwie [3][7].



Rys. 3.1. Przykładowy graf w bazie Neo4j [opracowanie własne]

3.3.1. Cypher

Cypher to popularny język zapytań bazujący na openCypher. Jest wykorzystywany w wielu implementacjach grafowych baz danych oraz samym Neo4j. Język ten cechuje intuicyjność i prostota [7].

3.3.2. neo4j-faker

neo4j-faker to narzędzie rozszerzające Cypher o nowe funkcje i procedury [17]. Pozwalają one na generowanie dużej ilości testowych danych przydatnych w początkowych fazach rozwoju aplikacji. Wykorzystując neo4j-faker programista oszczędza mnóstwo czasu. Wystarczy jeden skrypt, by wypełnić bazę setkami węzłów i relacji między nimi.

Biblioteka umożliwia wygenerowanie danych takich jak:

- osoba z podanym imieniem i nazwiskiem,
- adresy email, IP, URL,
- numer telefonu,
- data i czas.

3.4. JUnit

JUnit to narzędzie służące do pisania testów jednostkowych w języku Java. Ułatwia programistom pisanie, uruchamianie i zarządzanie testami, co pozwala na sprawdzanie poprawności działania kodu źródłowego. Framework ten zapewnia metody i adnotacje do definiowania testów, ułatwiając proces testowania oprogramowania [15].

3.5. Reactor Test

Reactor Test to framework do testowania aplikacji napisanych z wykorzystaniem Project Reactor. Umożliwia testowanie reaktywnych aplikacji i operacji asynchronicznych dzięki dostarczonym narzędziom. Podstawowy blok budujący test to interfejs StepVerifier, który w czytelny sposób pozwala pisać testy sprawdzające zachowanie aplikacji w warunkach asynchronicznych i wielowątkowych [14].

3.6. TypeScript

TypeScript to otwartoźródłowy i ściśle typowany język programowania, który narodził się z inicjatywy firmy Microsoft. Pierwsza wersja tego języka pojawiła się w roku 2012 [4].

Język ten powstał w odpowiedzi na problemy, które JavaScript wprowadzał, zwłaszcza w przypadku większych projektów. Dynamiczna natura języka JavaScript powodowała pojawienie się trudnych do wykrycia błędów w trakcie działania programu. TypeScript, dzięki swojemu systemowi typowania, pozwala uniknąć wielu błędów już na etapie transpilacji. W przypadku wykrycia błędu, takiego jak niepoprawny typ danych, proces zostaje przerwany.

TypeScript jest nadzbiorem języka JavaScript, co oznacza, że każdy poprawnie działający program napisany w JavaScript będzie również poprawnym programem w języku TypeScript. Jednakże, TypeScript nie działa bezpośrednio w tym samym środowisku, co JavaScript. Aby uruchomić TypeScript, konieczne jest jego wcześniejsze transpilowanie do JavaScript.

3.7. Angular

Angular to utrzymywany przez Google reaktywny framework do budowy aplikacji internetowych. Angular powstał z wykorzystaniem TypeScript jako następca AngularJS [4].

Główne założenia Angulara to umożliwienie programistom budowania skalowalnych aplikacji internetowych poprzez wykorzystanie struktury komponentowej, dwukierunkowego wiązania danych, oraz obszernych narzędzi do testowania.

Framework ten wspiera budowanie aplikacji jednostronicowych, umożliwiając rozwój aplikacji w sposób modułowy i efektywny. Angular stawia również nacisk na separację odpowiedzialności.

4. Dokumentacja techniczna

Niniejszy rozdział zawiera informacje na temat architektury, bezpieczeństwa aplikacji oraz REST API.

4.1. Aplikacja internetowa

Wykorzystanie aplikacji internetowej w porównaniu do aplikacji mobilnej ma istotną przewagę, polegającą na dostępności na praktycznie każdym urządzeniu wyposażonym w przeglądarkę internetową oraz połączenie z Internetem. Ta uniwersalność oznacza, że użytkownicy nie muszą instalować żadnych dodatkowych programów, co sprawia, że korzystanie z aplikacji jest wygodne i możliwe na różnych platformach. Dzięki temu podejściu użytkownicy mogą korzystać z aplikacji bez względu na to, czy pracują na komputerze, tablecie, czy smartfonie, o ile mają dostęp do przeglądarki internetowej i połączenia z Internetem. To rozwiązanie sprawia, że aplikacja staje się dostępna dla szerokiej grupy użytkowników.

W projekcie wykorzystano framework Angular w wersji 15.2 oraz bibliotekę Angular Material, która jest implementacją wzornictwa Material Design od firmy Google. To połączenie umożliwia budowanie interfejsów o nowoczesnym wyglądzie i funkcjonalności. Każda element widoczny w interfejsie użytkownika to osobny komponent. Na każdy z nich składają się trzy pliki:

- plik z rozszerzeniem .ts, w którym znajduje się klasa z adnotacją "@Component" obsługująca logikę danego komponentu,
- plik z rozszerzeniem .html, który jest szablonem definiujący wygląd komponentu. Może komunikować się z plikiem .ts,
- plik z rozszerzeniem .css, w którym znajdują się style zdefiniowane przez programistę i przeznaczone do użycia w danym komponencie.

Dodatkowo, framework Angular oferuje moduł Router, który umożliwia nawigację wewnątrz aplikacji. Wykorzystany został również Tailwind, który oferuje wiele użytecznych klas CSS przydatnych w budowaniu interfejsu użytkownika.

Aplikacja kliencka komunikuje się z serwerem przy użyciu dostarczanego z Angularem narzędzia o nazwie "HttpClient". Do tego zadania zostały wykorzystane klasy określane jako *serwisy*. Są one wstrzykiwane do komponentów lub innych serwisów. Dane są dostarczane przez protokół HTTP.

4.2. Aplikacja serwerowa

Aplikacja serwerowa powstała w oparciu o Spring Boot. Jej głównym zadaniem jest komunikacja z bazą danych i dostarczanie do klienta żądanych danych. W aplikacji znajdują klasy z adnotacją "@Node". Są one obiektową reprezentacją węzłów znajdujących się w bazie danych, w których przechowywane są wszystkie informacje. Relacje między węzłami są dostępne w postaci generycznej kolekcji "Set". Informacje o typie relacji między węzłami przechowuje adnotacja "@Relationship" umieszczona tuż nad kolekcjami.

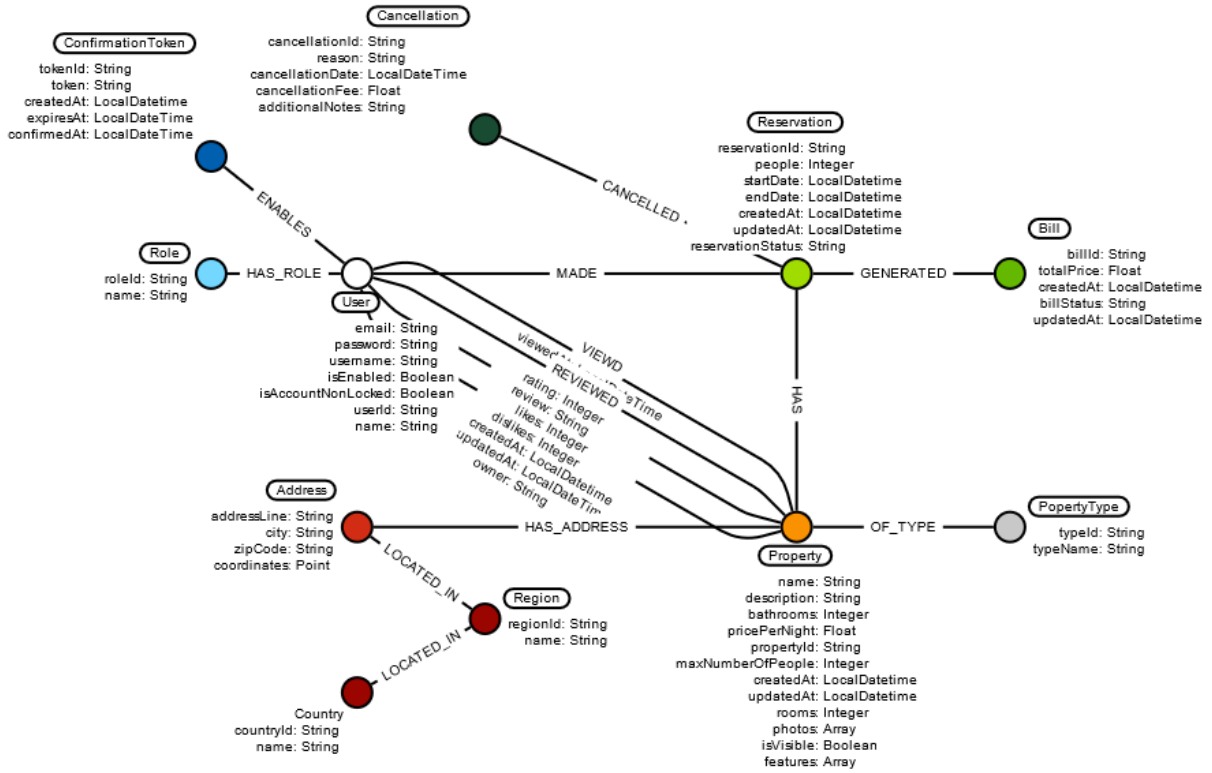
Komunikacja z bazą danych przy pomocy repozytoriów rozszerzających "ReactiveNeo4jRepository". Jest to generyczny interfejs przyjmujący dwa parametry: typ węzła za jaki ma odpowiadać oraz typ identyfikatora danego węzła. Mniej złożone zapytania mogą być generowane przez Spring Boot przy zastosowaniu odpowiedniego nazewnictwa dla metod. Do budowy bardziej złożonych zapytań można wykorzystać następujące rozwiązania:

- adnotację "@Query",
- interfejs "Neo4jClient" lub jego jego reaktywny odpowiednik,
- rozszerzenie interfejsu repozytorium przez Cypher DSL.

Aplikacja oferuje również zabezpieczenia. Mają one chronić zasoby przed użytkownikami, którzy próbują uzyskać dostęp do danych, które do nich nie należą. Szczegóły przedstawione zostały w rozdziale 4.4.

Serwer udostępnia dane klientom poprzez wykorzystanie REST API (ang. *Representational State Transfer Application Programming Interface*). Dostępne węzły zostały szerzej omówione w rozdziale 4.3.

4.3. Baza danych



Rys. 4.1. Schemat bazy danych [opracowanie własne]

Na Rysunku 4.1 znajduje się schemat bazy danych, z którą współpracuje aplikacja serwerowa. Ta baza danych przechowuje niezbędne informacje, które są kluczowe dla funkcjonowania całego systemu.

W grafie znajduje się węzeł o etykiecie "Property", który przechowuje szczegółowe informacje na temat konkretnych lokali, stanowiących przedmiot ogłoszeń. W tym węźle gromadzone są podstawowe informacje, takie jak cena za noc, liczba pokoi oraz zdjęcia lokalu. Szczególnie istotne jest pole "isVisible", które określa, czy dane ogłoszenie jest widoczne dla zwykłych użytkowników. Ogłoszenia, w których to pole jest ustawione na "false", są dostępne jedynie dla osób zarządzających aplikacją oraz właściciela lokalu. Pozostałe informacje związane z lokalami są przechowywane w osobnych węzłach. Rodzaj danego lokum tj. czy jest to willa lub chatka w lesie, znajduje się w węźle "PropertyType". Dzięki wydzieleniu tej informacji z "Property" unikamy niepotrzebnej redundancji, ponieważ możemy ten sam typ przypisać do wielu innych lokali. Do połączenia z "PropertyType" wykorzystana została relacja "OF_TYPE".

Dane dotyczące położenia lokalu znajdują się w "Address". Warto zaznaczyć, że węzeł "Address" nie przechowuje wszystkich informacji dotyczących adresu. Łączy się on z węzłami "Country" i "Region", aby otrzymać pełny adres, który jednoznacznie określa położenie lokalu. "Address" łączy się ze wspomnianymi węzłami za pomocą relacji "LOCATED_IN", natomiast z "Property" przez "HAS_ADDRESS".

Trzy spośród wymienionych węzłów, mianowicie "PropertyType", "Country" oraz "Region", zawierają stałe dane. Te rodzaje informacji rzadko ulegają zmianom lub w ogóle nie zmieniają się przez cały okres funkcjonowania systemu. W wyniku tego nie ma możliwości ich modyfikacji z poziomu aplikacji serwerowej lub klienta. Jedyną opcją jest bezpośrednie zapytanie do bazy danych.

Kolejny bardzo istotny węzeł to "Reservation". Przechowuje on najbardziej podstawowe informacje na temat historycznych, jak i obecnych wynajmów. Wyróżniającą się właściwością tego węzła jest "reservationStatus", ponieważ może przyjmować tylko trzy wartości:

- ACTIVE,
- CANCELED,
- COMPLETED.

Podobnie jak we wcześniej opisanym przypadku także i tutaj niektóre dane zostały wydzielone. Informacje na temat płatności znajdują się w "Bill". Podobnie jak "Reservation" rachunek za wynajem również może być w trzech różnych stanach:

- ACTIVE,
- CANCELED,
- PAID.

Rachunek łączy się z wynajmem przez relację "GENERATED". System umożliwia użytkownikowi anulowanie wynajmu. W takim przypadku wszelkie informacje związane z odwołaniem wynajmu są przechowywane w węźle "Cancellation". Po anulowaniu wynajmu, status zarówno wynajmu, jak i rachunku zostaje zmieniony na "CANCELED".

Użytkownik aplikacji jest reprezentowany przez węzeł "User". Informacje zawarte w tym węźle służą identyfikacji użytkownika oraz sprawdzeniu, czy ma dostęp do aplikacji ("isEnabled"). Na Rysunku 4.1 można zauważyć, że omawiany węzeł jest ściśle związany z węzłem "Property" za pomocą czterech różnych relacji. Dwie z tych relacji można określić jako "proste".

Pierwsza z nich to relacja "LIKES", która informuje, czy użytkownik polubił dany lokal. Druga relacja "POSTED" wskazuje, które ogłoszenia należą do danego użytkownika. Pozostałe dwie relacje, mianowicie "VIEWED" oraz "REVIEWED", są bardziej złożone, ponieważ przechowują dodatkowe informacje. Relacja "VIEWED" śledzi pięć ostatnich lokali, które użytkownik ostatnio przeglądał, a informacje te są zapisywane razem z datą i godziną przeglądania. Z kolei relacja "REVIEWED" pozwala użytkownikowi wystawić recenzję po zakończeniu pobytu w danym lokalu, a przydatność recenzji jest określana na podstawie właściwości "likes" oraz "dislikes".

"ConfirmationToken" to węzeł przechowujący specjalny kod służący do aktywacji nowego konta. Jest on generowany w trakcie rejestracji użytkownika i przesyłany na adres e-mail podany w procesie rejestracji. Użytkownika po kliknięciu w link z kodem aktywuje konto, a informacja o tym jest zapisywana w "ConfirmationToken" jako "confirmedAt".

4.4. Bezpieczeństwo

Istotną częścią aplikacji serwerowej jest to w jaki sposób zapewnia bezpieczeństwo. Pewne funkcjonalności powinny być dostępne jedynie dla zalogowanych użytkowników. Rola, jaka została przydzielona użytkownikowi, również ma istotny wpływ na to co użytkownik może zrobić oraz do jakich danych ma dostęp.

Niniejsza aplikacja korzysta z kilku rozwiązań, które służą temu by wymienione wymagania zostały spełnione. Są one następujące:

- autoryzacja żądań przy pomocy tokena w standardzie JWT,
- adnotacja "@Valid",
- role oraz sprawdzenie, czy użytkownik ma dostęp do danego zasobu,
- niewykorzystywanie ręcznego "składania" zapytań.

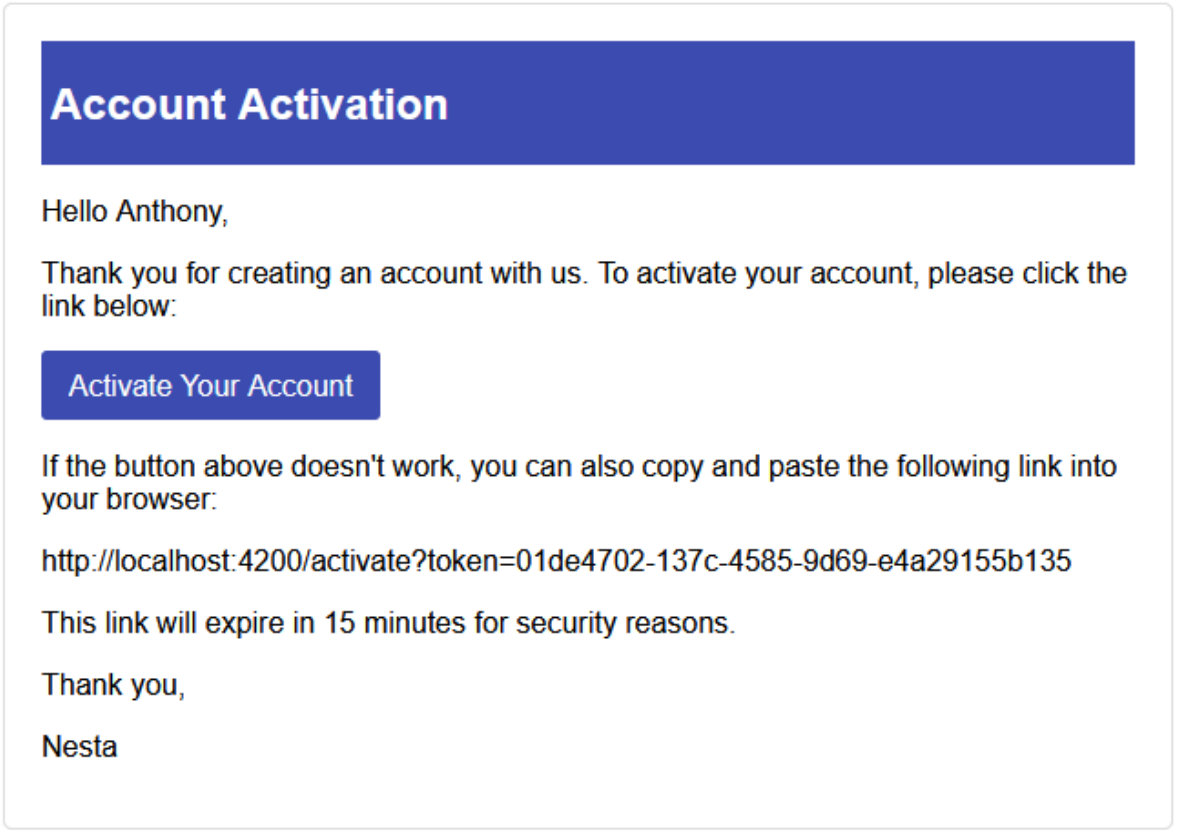
JWT (JSON Web Token) to sposób na zabezpieczenie informacji przesyłanych pomiędzy stronami przy użyciu formatu danych o nazwie JSON [16]. W przypadku niniejszego systemu JWT przechowuje następujące informacje:

- data powstania,
- data wygaśnięcia,
- id użytkownika,
- flagę mówiącą czy dany użytkownik jest administratorem.

Aplikacja serwerowa korzysta z tego rozwiązania w trakcie procesu logowania. Jeżeli proces uwierzytelnienia się powiedzie, serwer przesyła w odpowiedzi token.

Jest on ważny przez określony czas. Gdy wygaśnie, nie może być dłużej używany i użytkownik musi się zalogować ponownie. JWT jest przechowywany w przeglądarce w tzw. "Pamięci lokalnej", z której jest usuwany po wylogowaniu.

Istotną kwestią jest również zadbanie by rejestracja nowego użytkownika nie była zbyt prosta. Ktoś mógłby założyć wiele kont w serwisie i wykorzystać je do umieszczania w nim treści, które nie powinny się w nim znaleźć. By temu zaradzić po założeniu konta użytkownik musi udać się na swoją skrzynkę pocztową. Tam będzie czekała na niego wiadomość email z linkiem aktywującym konto. Dopiero po wejściu w podany link jest możliwość zalogowania się na konto.



Rys. 4.2. E-mail z linkiem aktywującym konto [Opracowanie własne]

Adnotacja "@Vaild" to część zewnętrznego pakietu służącego do weryfikacji poprawności przesyłanych danych. Działa ona wraz z innymi adnotacjami, którymi oznacza się informacje, które powinny być sprawdzone pod kątem poprawności. Dzięki takiemu rozwiązaniu w aplikacji udało się ograniczyć powtarzalny kod, który by do tego służył. Ograniczono też ryzyko przesłania danych, które po zapisaniu mogłyby wpłynąć na stabilność systemu oraz jego bezpieczeństwo.

```java
@Operation(summary = "Deletes property from the database.")
@DeleteMapping("/{propertyId}/by/{userId}")
Mono<ResponseEntity<Void>> deleteProperty(@PathVariable String propertyId, @PathVariable String userId,
                                          @AuthenticationPrincipal UserDetails userDetails) {

    log.info("Deleting property");
    User user = (User) userDetails;
    if (!user.getUserId().equals(userId) && !user.isAdmin()) {
        log.error("User {} tried to delete other user's property", user.getUserId());
        return Mono.just(ResponseEntity.status(HttpStatus.FORBIDDEN).build());
    }

    Mono<Void> result = propertyService.deleteProperty(new DeletePropertyDto(userId, propertyId));
    return mapToMonoResponseEntity(result);
}
```

Rys. 4.3. Przykładowa zabezpieczona metoda [opracowanie własne]

Role wprowadzają podział na zwykłych użytkowników oraz administratorów w aplikacji. Druga grupa ma za zadanie utrzymywać porządek w serwisie. Z tego powodu potrzebują większych uprawnień, by na przykład ukryć ogłoszenie danego użytkownika. Rysunek 4.3 przedstawia taki przypadek. Metoda służy do usunięcia ogłoszenia z lokalem. Dostęp do niej powinien mieć tylko użytkownik, do którego należy to ogłoszenie oraz administrator. Dzięki temu możliwe jest utrzymanie porządku w aplikacji przez osoby, które się tym zajmują.

Ostatni środek bezpieczeństwa dotyczy warstwy dostępu do danych. Spring Data Neo4j umożliwia wysłanie zapytania do bazy danych na kilka sposobów. W aplikacji w głównej mierze wykorzystywane są zapytania generowane automatycznie na podstawie nazwy metody. Użyta została także adnotacja "@Query" pozwalająca ręcznie zbudować zapytanie. Zrezygnowanie z budowy zapytań, na przykład poprzez łączenie łańcuchów znaków, znacznie ograniczyło możliwość wystąpienia ataku *Cypher Injection*. Atak ten polega na wstrzyknięciu własnego kodu Cypher do zapytania.

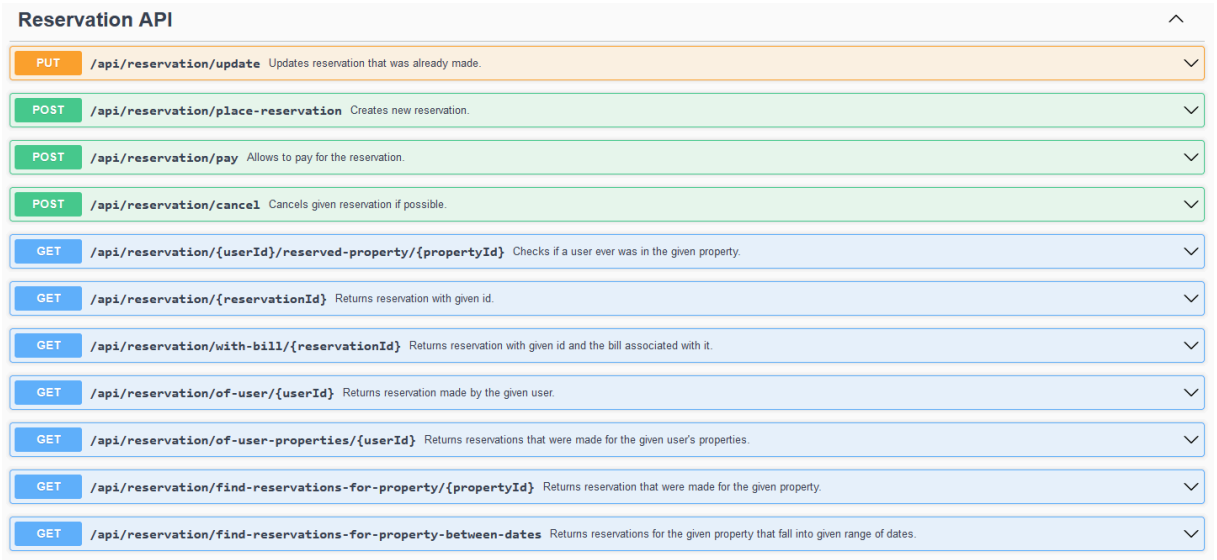
4.5. API aplikacji serwerowej

Aplikacja korzysta z REST (ang. *Representational State Transfer Application Programming Interface*) do komunikacji z aplikacją kliencką. Jest to jeden z najpopularniejszych stylów architektonicznych do komunikacji komputer-komputer oraz komputer-człowiek. W Spring Boot jego wykorzystanie opiera się o klasy-kontrolery. Wykorzystując odpowiednie adnotacje dla klasy oraz metod definiowany jest węzeł z konkretną metodą. Sama komunikacja odbywa się przy użyciu protokołu HTTP. W aplikacji wykorzystano następujące metody:

- GET,
- POST
- PUT,
- PATCH,
- DELETE.

Pośrednio wykorzystywana jest również metoda "OPTIONS". Aplikacja kliencka używa jej przy tak zwanym *preflight request*. Żądanie z tą metodą jest wysyłane przed tym właściwym, aby sprawdzić, czy można je bezpiecznie wysłać.

4.5.1. Kontroler wynajmu



Rys. 4.4. Kontroler wynajmu [opracowanie własne]

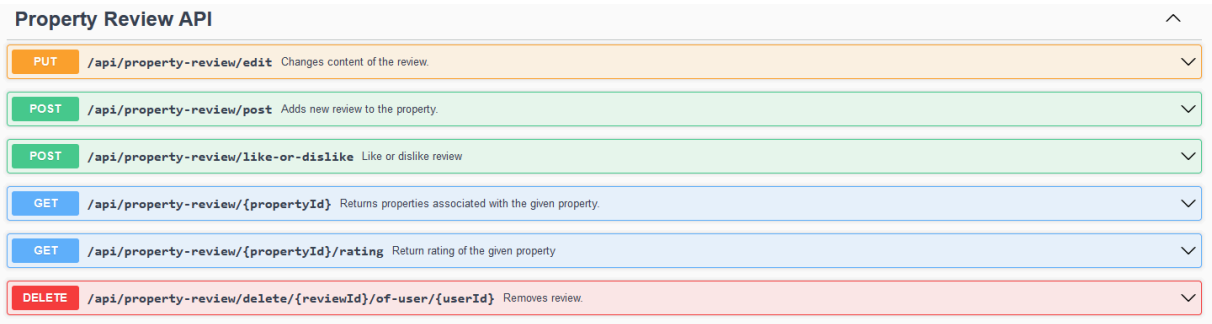
Kontroler odpowiadający za dokonywanie wynajmu oraz zarządzanie nimi. Dostęp do węzłów przedstawionych na rysunku 4.4 ma tylko zalogowany użytkownik.

4.5.2. Kontroler ogłoszeń lokali

Rys. 4.5. Kontroler ogłoszeń lokali [opracowanie własne]

Kontroler umożliwiający zarządzanie dostępnymi lokalami. Tylko węzły z metodą "GET" (oprócz "/api/property/hide-all-of/{userId}") nie wymagają aby użytkownik był zalogowany.

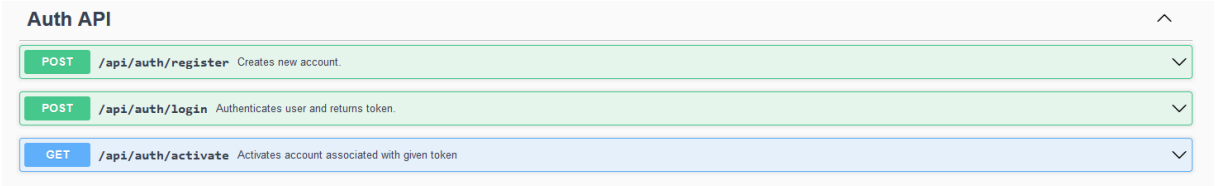
4.5.3. Kontroler recenzji lokali



Rys. 4.6. Kontroler recenzji lokali [opracowanie własne]

Kontroler służący zarządzaniu recenzjami napisanymi przez użytkownika oraz pobieraniu ich dla danego lokalu. Tylko węzły z metodą "GET" nie wymagają aby użytkownik był zalogowany.

4.5.4. Kontroler rejestracji i uwierzytelnienia użytkownika



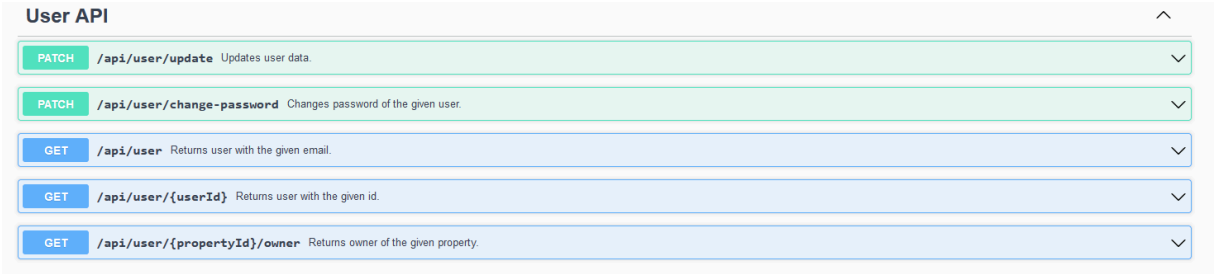
Rys. 4.7. Kontroler rejestracji i uwierzytelnienia użytkownika [opracowanie własne]

Węzły przedstawione na rysunku 4.7 umożliwiają kolejno:

- założenie nowego konta w serwisie,
- zalogowanie się na istniejące konto,
- aktywację założonego konta.

Węzły nie wymagają uwierzytelnienia.

4.5.5. Kontroler użytkownika



Rys. 4.8. Kontroler użytkownika [opracowanie własne]

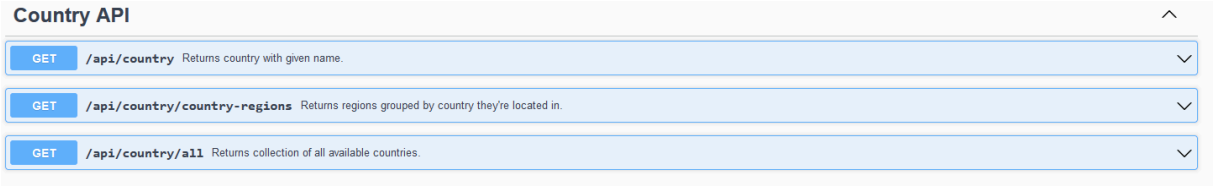
Kontroler odpowiedzialny za zarządzanie użytkownikiem oraz zwracanie informacji o nim. Tylko węzły z metodą "PATCH" wymagają uwierzytelnienia.

4.5.6. Kontroler rekomendacji lokali

Rys. 4.9. Kontroler rekomendacji lokali [opracowanie własne]

Kontroler zwracający proponowane lokale zalogowanemu użytkownikowi.

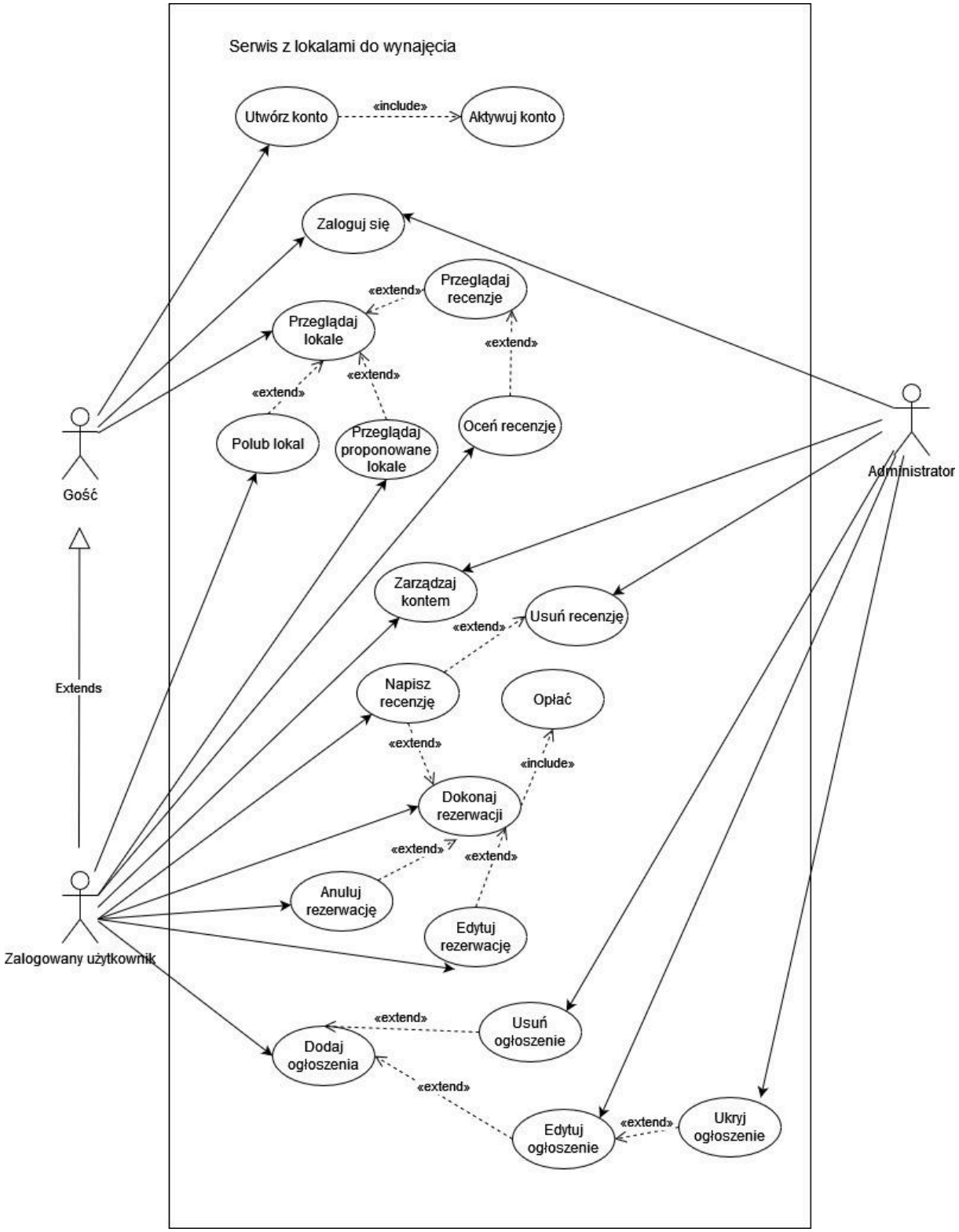
4.5.7. Kontroler kraju



Rys. 4.10. Kontroler kraju [opracowanie własne]

Kontroler zwracający wybrany kraj lub wszystkie dostępne w bazie.

5. Diagram przypadków użycia



Rys. 5.1. Diagram przypadków użycia [opracowanie własne]

Rysunek 5.1 przedstawia diagram przypadków użycia ilustrujący funkcjonalności dostępne w systemie. Przewidziano trzech aktorów:

- gość,
- zalogowany użytkownik,
- administrator.

Gość, to użytkownik, który nie jest zalogowany lub nie założył swojego konta w serwisie. Z tego powodu funkcje, do których ma dostęp są bardzo ograniczone. Gość posiada możliwość przeglądania ogłoszeń, recenzji oraz założenia konta. Po zakończonym procesie rejestracji i aktywowaniu konta może się na nie zalogować.

Zalogowany użytkownik ma dostęp do pełnej wersji serwisu. Poza podstawową opcją przeglądania ogłoszeń zyskuje możliwość otrzymywania rekomendacji. Użytkownik może również dodawać ogłoszenia ze swoimi lokalami oraz zarządzać nimi. Istnieje również możliwość wynajmu lokalu z ogłoszenia innego użytkownika. Po zakończonym pobycie użytkownik może ocenić lokal poprzez napisanie recenzji wraz z ocenę.

Ostatni aktor to administrator. Jego rola polega na zarządzaniu serwisem. Ma on dostęp do wszystkich ogłoszeń, recenzji, wynajmów oraz kont użytkowników i może nimi zarządzać.

6. Wybrane rozwiązania implementacyjne

Kolejne podrozdziały zawierają opisy wybranych rozwiązań implementacyjnych, które znalazły się w systemie. Przybliżone zostały między innymi różne sposoby budowy zapytań, globalne przechwytywanie wyjątków w aplikacji serwerowej oraz obsługa wynajmu lokali.

6.1. System rekomendacji

```
@Query("""
    MATCH (user:User {userId: $userId})-[:LIKES|VIEWED]->(property:Property)<-[:LIKES|VIEWED]-(otherUser:User)
    WHERE user <> otherUser
    WITH user, otherUser, COUNT(DISTINCT property) AS propertyCount
    ORDER BY propertyCount DESC

    MATCH (otherUser)-[:LIKES|VIEWED]->(recommendedProperty:Property)-[r_address:HAS_ADDRESS]-(a:Address)
    WHERE NOT EXISTS((user)-[:LIKES|VIEWED]->(recommendedProperty))
    OPTIONAL MATCH (recommendedProperty)-[r_type:OF_TYPE]-(type)
    OPTIONAL MATCH (a)-[r_located_r:LOCATED_IN]-(region:Region)-[r_located_c:LOCATED_IN]-(c:Country)
    RETURN DISTINCT recommendedProperty, collect(r_address), collect(a), collect(r_type), collect(type),
    collect(r_located_r), collect(region), collect(r_located_c), collect(c)
    LIMIT 5
    """)
Flux<Property> recommendPropertiesBasedOnLikesAndViewed(String userId);

1 usage
@Query("""
    MATCH (user:User {userId: $userId})-[:MADE]->(:Reservation)<-[:HAS]-(reservedProperty:Property)
    WITH user, COLLECT(DISTINCT reservedProperty) AS userReservedProperties

    MATCH (otherUser:User)-[:MADE]->(:Reservation)<-[:HAS]-(otherReservedProperty:Property)-[r_address:HAS_ADDRESS]-(a:Address)
    WHERE user <> otherUser AND NOT otherReservedProperty IN userReservedProperties
    OPTIONAL MATCH (otherReservedProperty)-[r_type:OF_TYPE]-(type)
    OPTIONAL MATCH (a)-[r_located_r:LOCATED_IN]-(region:Region)-[r_located_c:LOCATED_IN]-(c:Country)
    RETURN DISTINCT otherReservedProperty, collect(r_address), collect(a), collect(r_type), collect(type),
    collect(r_located_r), collect(region), collect(r_located_c), collect(c)
    LIMIT 5
    """)
Flux<Property> recommendPropertiesBasedOnPreviousReservations(String userId);
```

Rys. 6.1. Zapytania zwracające proponowane ogłoszenia lokali [Opracowanie własne]

Rysunek 6.1 przedstawia dwa zapytania w języku Cypher składające się na system rekomendacji wykorzystywany w aplikacji.

"recommendPropertiesBasedOnLikesAndViewed" to pierwsza z metod składająca się na wspomniany system. Przyjmuje ona jako argument unikalny identyfikator użytkownika na podstawie którego wyszukiwani są inni użytkownicy. Brani pod uwagę są ci, którzy polubili lub oglądali te same ogłoszenia co użytkownik z identyfikatorem "userId". Druga część zapytania ma za zadanie zwrócić lokale, które zostaną przedstawione użytkownikowi jako te, które mogą go zainteresować.

"recommendPropertiesBasedOnPreviousReservations" zwraca ogłoszenia lokali na podstawie historii wynajmów użytkownika oraz wynajmów innych osób. Na podstawie tego zwraca powiązane z nimi lokale.

Obie przedstawione metody wykorzystują tak zwany Collaborative Filtering. Jest to technika polegająca dopasowywaniu użytkowników o podobnych gustach. Na podstawie tego zwracane są rekomendowane ogłoszenia [14].

6.2. Uwierzytelnienie

```
@Override
public Mono<SecurityContext> load(ServerWebExchange exchange) {
    return Mono.justOrEmpty(exchange.getRequest() ServerHttpRequest
            .getHeaders() HttpHeaders
            .getFirst(HttpHeaders.AUTHORIZATION))
        .filter(authHeader -> authHeader.startsWith("Bearer "))
        .switchIfEmpty(Mono.error(new AuthException("This is not a bearer token or no token provided")))
        .flatMap(authHeader -> {
            String token = authHeader.substring( beginIndex: 7);
            return authenticate(token);
        }).doOnError(throwable -> log.error(throwable.getMessage()));
}

1 usage
private Mono<SecurityContext> authenticate(String token) {
    return Mono.just(jwtService.extractUserId(token)) Mono<String>
        .flatMap(userId -> {
            Mono<UserDetails> userDetails = userRepository.findById(userId).map(user -> user);
            return Mono.zip(userDetails, Mono.just(token));
        }).onErrorResume(throwable -> Mono.error(new AuthException(throwable.getMessage()))) Mono<Tuple2<
        .flatMap(tuple -> updateSecurityContext(tuple.getT1(), tuple.getT2()));
}

1 usage
private Mono<SecurityContext> updateSecurityContext(UserDetails userDetails, String token) {
    if (!userDetails.isEnabled()) {
        return Mono.error(new AuthException("Account disabled"));
    }

    var passwordAuthenticationToken = new UsernamePasswordAuthenticationToken(userDetails, token);
    return authenticationManager.authenticate(passwordAuthenticationToken)
        .map(SecurityContextImpl::new);
}
```

Rys. 6.2. Fragment kodu odpowiadającego za uwierzytelnienie [Opracowanie własne]

Kod przedstawiony na rysunku 6.2. to fragment funkcjonalności odpowiadającej za uwierzytelnienie użytkownika. Proces ten zaczyna się od metody "load", która jest uruchamiana przy każdym żądaniu, które wymaga uwierzytelnienia. Taki sposób działania jest konieczny ponieważ wykorzystywany jest JSON Web Token (rozdział 4.4). Metoda pobiera z otrzymanego żądania nagłówek "Authorization", w którym powinien znaleźć się token z przedrostkiem "Bearer". Jeśli nic nie zostało znalezione lub brakuje wspomnianego przedrostka proces się kończy i użytkownik nie uzyskuje dostępu do żądanego zasobu.

Dalsza część kodu odpowiada za sprawdzenie, czy przesłany token jest ważny oraz odnalezienie w bazie użytkownik żądającego dostępu. Na koniec aktualizowany jest kontekst bezpieczeństwa aplikacji. Odbywa się to przez wywołanie "authenticationManager.authenticate(passwordAuthenticationToken)".

6.3. Filtrowanie i sortowanie ogłoszeń

```
propertiesResultStatement = Cypher.match(List.of(propertyAndAddress, addressAndRegion, countryAndRegion))
    .where(condition) OngoingReadingWithWhere
    .optionalMatch(typeOfFoundProperty) OngoingReadingWithoutWhere
    .returning(
        property.getRequiredSymbolicName(),
        Functions.collect(propertyAndAddress),
        address.getRequiredSymbolicName(),
        Functions.collect(addressAndRegion),
        region.getRequiredSymbolicName(),
        Functions.collect(countryAndRegion),
        country.getRequiredSymbolicName(),
        Functions.collect(typeOfFoundProperty),
        propertyType.getRequiredSymbolicName()) OngoingReadingAndReturn
    .orderBy(orderBy == null ? List.of() : List.of(orderBy)) OngoingMatchAndReturnWithOrder
    .skip(pageRequest.getOffset()) TerminalExposesLimit
    .limit(oneMore) BuildableStatement<ResultStatement>
    .build();
```

Rys. 6.3. Fragment kodu odpowiadającego za filtrowanie i sortowanie ogłoszeń [Opracowanie własne]

Ważną funkcjonalnością aplikacji jest możliwość filtrowania i sortowania ogłoszeń lokali przez użytkownika. Rysunek 6.3. przedstawia fragment kodu odpowiadającego za budowę zapytania implementującego opisaną funkcjonalność przy wykorzystaniu Cypher DSL. W zależności od tego na podstawie jakich parametrów ma odbywać się filtrowanie zmienna "condition" będzie zawierała inny warunek. Na przykład w nazwie ma znajdować się fraza "lake" oraz cena pobytu ma być mniejsza niż 120.

Parametry sortowania znajdują się w zmiennej "orderBy" i również są dynamicznie zmieniane na podstawie przesłanych do metody parametrów. Domyślnie sortowanie odbywa się po dacie umieszczenia ogłoszenia.

Wyniki zwrócone z metody są podzielone na strony z wykorzystaniem interfejsu "Pageable". Dzięki temu można je w łatwy i przejrzysty sposób przedstawić na stronie internetowej. Jest to również forma optymalizacji, ponieważ nie ma potrzeby pobierania wszystkich lokali z bazy danych i w takiej formie przesyłania ich dalej. Skraca to czas potrzebny na przesłanie danych oraz zmniejsza zużycie pamięci.

6.4. Wynajmowanie lokalu

```java
@Override
public Mono<ReservationWithBillDto> placeReservation(PlaceReservationDto reservationDto) {
    if (null == reservationDto) {
        log.error("'reservation' was null");
        return Mono.error(new IllegalArgumentException("'reservation' was null"));
    }

    if (ChronoUnit.DAYS.between(reservationDto.startDate(), reservationDto.endDate()) < 1L) {
        return Mono.error(new ReservationException("Reservation period is too short"));
    }

    return doesReservationIntersectWithOther(reservationDto) Mono<Boolean>
        .flatMap(canReservationBePlaced ->
            !canReservationBePlaced
                ? Mono.error(new ReservationException("User already has reservations in given period or place is already reserved for given period"))
                : Mono.just(reservationDto)) Mono<PlaceReservationDto>
        .flatMap(this::assembleReservation) Mono<Reservation>
        .doOnError(throwable -> log.error("Error while assembling the reservation", throwable))
        .flatMap(reservationRepository::save)
        .flatMap(savedReservation -> addBillToReservation(savedReservation, reservationDto))
        .flatMap(reservationWithBill -> connectPropertyWithReservation(reservationDto.propertyId(), reservationWithBill))
        .flatMap(reservation -> connectUserWithReservation(reservationDto.userId(), reservation))
        .doOnError(throwable -> log.error("Error while placing the reservation", throwable))
        .map(Reservation::toDtoWithBill);
}
```

Rys. 6.4. Główna metoda odpowiadająca za wynajem lokalu [Opracowanie własne]

Metoda "placeReservation" to punkt, od którego rozpoczyna się proces wynajmu. Na początku sprawdzana jest możliwość wynajmu. Pierwsza instrukcja warunkowa sprawdza, czy okres na jaki użytkownik chce dokonać wynajmu nie jest zbyt krótki. W dalszej część pojawia się metoda "doesReservationIntersectWithOther". Sprawdza ona, czy wynajem nie koliduje z innymi już dokonanymi przez użytkownika oraz innych osób, które również wybrały ten sam lokal. Istotne jest również połączenie nowego węzła "Reservation" z "Bill", aby móc przechowywać w nim informację na dotyczące płatności za wynajem lokalu. Pozostała część metody odpowiada za połączenie z węzłami "Property" i "User".

6.5. Rejestracja użytkownika

```
return userRepository.findByUsername(registerDto.login())
    .defaultIfEmpty(
        new User(
             userId: null,
            registerDto.name(),
            registerDto.login(),
            registerDto.email(),
            passwordEncoder.encode(registerDto.password()),
             isEnabled: false,
             isAccountNonLocked: true)).flatMap(user -> {
        if (!Strings.isBlank(user.getUserId())) {
            log.error("Username '{}' already taken", registerDto.login());
            return Mono.error(new IllegalStateException("Username '" + registerDto.login() + "' already taken"));
        }

        return userRepository.findByEmail(registerDto.email());
    }).defaultIfEmpty(
        new User(
             userId: null,
            registerDto.name(),
            registerDto.login(),
            registerDto.email(),
            passwordEncoder.encode(registerDto.password()),
             isEnabled: false,
             isAccountNonLocked: true)).flatMap(user -> {
        if (!Strings.isBlank(user.getUserId())) {
            log.error("Email '{}' already taken", registerDto.email());
            return Mono.error(new IllegalStateException("Email '" + registerDto.email() + "' already taken"));
        }

        return roleRepository.findByName("ROLE_USER")
            .flatMap(role -> {
                user.addRole(role);
                return userRepository.save(user);
            }).flatMap(u -> sendMail(u, registerDto));
    });
```

Rys. 6.5. Rejestracja użytkownika [Opracowanie własne]

Rejestracja w systemie nowego użytkownika to jedna z najbardziej podstawowych funkcjonalności. Rysunek 6.5 przedstawia implementację, która zaczyna się od próby znalezienia w bazie danych użytkownika z taką samą nazwą. Jeśli kogoś takiego nie ma sprawdzany jest adres e-mail. W przypadku, gdy zapytanie nic nie zwróciło można przystąpić do rejestracji. Klasa "User" reprezentuje użytkownika i to tutaj przechowywane są dane przekazane z formularza rejestracyjnego. Następnie przypisywana jest rola (domyślnie jest to zwykły użytkownik) oraz zostaje wysłana wiadomość e-mail. Zawiera ona w sobie informacje na temat procesu aktywacji założonego konta (rozdział 4.4).

6.6. Przechwytywanie wyjątków

```
@ExceptionHandler(AuthException.class)
public Mono<ResponseEntity<Map<String, String>>> handleUnauthorized(AuthException e) {
    return Mono.just(new ResponseEntity<>(getErrorsMap(e.getMessage()), HttpStatus.UNAUTHORIZED));
}

4 usages
private <T> Map<String, T> getErrorsMap(T error) { return Map.of( k1: "errors", error); }
```

Rys. 6.6. Przykładowa metoda przechwytująca wyjątek [Opracowanie własne]

W aplikacji serwerowej wyjątki są przechwytywane przez klasę "RestExceptionHandler" mającą adnotację "@RestControllerAdvice". Pozwala to na scentralizowany sposób radzenia sobie z wyjątkami, które nie zostały przechwycone w inny sposób. Istotną zaletą tego rozwiązania jest to, że dla każdego wyjątku można przypisać status. Dla przykładu rysunek 6.6. przedstawia metodę przechwytującą wyjątki typu "AuthExcepetion". W odpowiedzi zwrócona zostanie informacja o tym, jaki problem wystąpił oraz status. W tym przypadku o kodzie 401.

6.7. Pobieranie recenzji dla danego lokalu

```
@Override
public Flux<Reviewed> findReviewsForGivenProperty(String propertyId) {
    if (Strings.isBlank(propertyId)) {
        log.error("'propertyId' was null or empty");
        return Flux.error(new IllegalArgumentException("'propertyId' was null or empty"));
    }

    return neo4jClient.query( cypher: """
            MATCH (User)-[r:REVIEWED]-(property:Property {propertyId: $propertyId})
            WITH r AS review, id(r) AS r_id, property
            RETURN review{.*, r_id}, property;
            """).bind(propertyId).to( name: "propertyId") RunnableSpec
        .fetchAs(Reviewed.class) MappingSpec<Reviewed>
        .mappedBy((typeSystem, record) -> mapRecordToReviewed(record)) RecordFetchSpec<Revie
        .all();
}
```

Rys. 6.7. Pobieranie recenzji z bazy danych [Opracowanie własne]

Przedstawiona na rysunku 6.7. metoda zwraca recenzja dla lokalu o podanym identyfikatorze. Wykorzystana została klasa "Neo4jClient", która jest jednym z trzech sposobów użytych do budowy zapytań w aplikacji serwerowej (rozdział 4.2).

Jest to niskopoziomowy sposób komunikacji z bazą danych ponieważ programista musi sam zapewnić odpowiednią metodę przetwarzającą wynik na odpowiednią klasę.

6.8. Filtr CORS

```
@Bean
public CorsWebFilter corsFilter() {
    UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
    CorsConfiguration config = new CorsConfiguration();
    config.setAllowCredentials(true);
    config.setAllowedOrigins(Collections.singletonList(allowedOrigin));
    config.setAllowedHeaders(
        Arrays.asList(
            ORIGIN,
            CONTENT_TYPE,
            ACCEPT,
            AUTHORIZATION
        ));

    config.setAllowedMethods(
        Arrays.asList(
            GET.name(),
            POST.name(),
            DELETE.name(),
            PUT.name(),
            PATCH.name()
        ));

    source.registerCorsConfiguration( path: "/**", config);

    return new CorsWebFilter(source);
}
```

Rys. 6.8. Konfiguracja filtra CORS [Opracowanie własne]

Komunikacją aplikacji serwerowej z aplikacją kliencką wymaga specjalnej konfiguracji. Zajmuje się tym metoda na rysunku 6.8. Ma ona za zadanie umożliwić żądaniom z podanego adresu na dostęp do serwera, wraz z dozwolonymi metodami oraz nagłówkami.

6.9. Zakończenie wynajmu

```
@Scheduled(fixedDelayString = "${scheduled.completeReservations.delay}", timeUnit = TimeUnit.SECONDS)
public Publisher<Void> scheduleReservationStatusUpdate() {
    return Mono.defer(reservationService::completeReservations)
        .doOnError(throwable -> log.error("Unexpected error occurred", throwable))
        .then();
}
```

Rys. 6.9. Metoda zamykająca wynajem [Opracowanie własne]

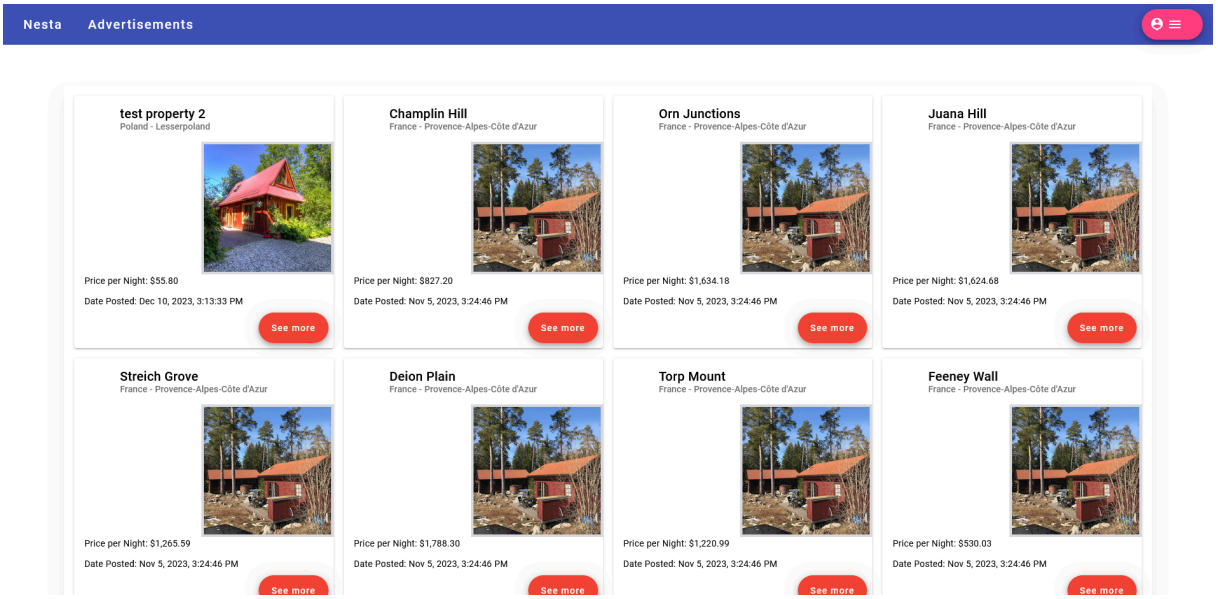
Wynajem posiada status, który opisuje jego obecny stan. Może być aktywny, zakończony lub anulowany. Stany pierwszy i trzeci są obsługiwane w kodzie poprzez, odpowiednio przejście przez proces wynajmu oraz jego anulowanie. Ostatni opisuje wynajem, który dobiegł końca. W tym samym czasie może być wiele aktywnych wynajmów, które mogą się w każdej chwili zakończyć. Stąd też operacje zakończenia trzeba wykonywać cyklicznie. Rysunek 6.9. prezentuje rozwiązanie przedstawionego problemu.

Adnotacja "@Scheduled" to kluczowa część przedstawionego kodu. Umożliwia ona zaplanowanie wykonania danej metody. W tym przypadku metoda ma się wykonać z zadeklarowanym opóźnieniem w sekundach po ostatnim wywołaniu. W dalszej części znajduje się właściwy kod aktualizujący status. Istotne z punktu widzenia integralności bazy danych jest aby metoda znajdowała się w osobnej klasie. Dzięki temu można użyć adnotacji "@Transactional", która zapewnia obsługę transakcji dla wykonywanego zapytania.

7. Interfejs użytkownika

Niniejszy rozdział to opis graficznego interfejsu użytkownika. Przedstawione zostaną najważniejsze ekrany, na które natknie się użytkownik w trakcie używania serwisu.

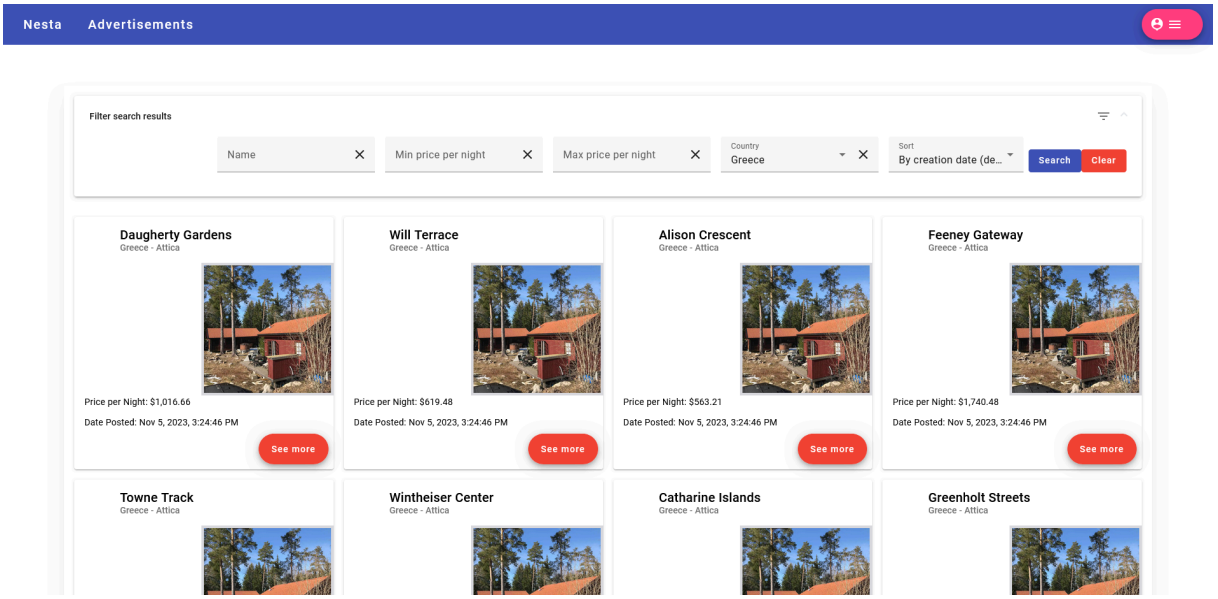
7.1. Strona główna



Rys. 7.1. Strona główna [opracowanie własne]

Rysunek 7.1 przedstawia stronę główna aplikacji. Znajdują się na niej najnowsze ogłoszenia, które mogą zainteresować użytkownika. Przycisk "See more" przenosi użytkownika do szczegółów danego ogłoszenia. Serduszko obok pozwala zalogowanemu użytkownikowi zapisać dane ogłoszenie do ulubionych. Poniżej ogłoszeń został umieszczony przycisk, który przekierowuje użytkownika do podstrony ze wszystkimi lokalami.

7.2. Strona z ogłoszeniami



Rys. 7.2. Strona z listą ogłoszeń [opracowanie własne]

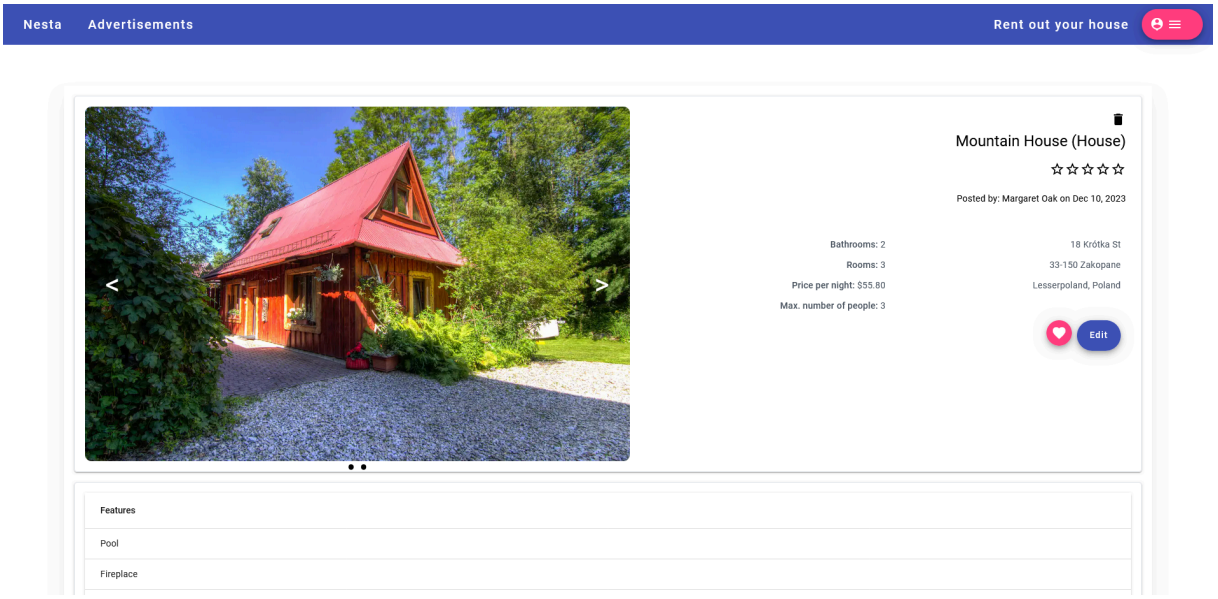
Po naciśnięciu przycisku "Find your place" na stronie głównej lub "Advertisements" na pasku u góry strony, użytkownik zostaje przeniesiony do widoku wszystkich aktualnie dostępnych ogłoszeń. Z tego miejsca istnieje możliwość filtrowania ogłoszeń na podstawie następujących kryteriów:

- nazwa,
- minimalna oraz maksymalna cena za pobyt przez jedną noc,
- kraj, w którym znajduje się lokal.

Ponadto użytkownik ma możliwość sortowanie wyników po cenie oraz dacie dodania. Sam widok ogłoszeń jest podzielony na strony. Nawigację umożliwiają przyciski na dole wraz z możliwością wyboru liczby lokali wyświetlanych jednocześnie.

Zalogowany użytkownik dodatkowo ma dostęp do rekomendacji. Rekomendowane ogłoszenia pojawiają się powyżej "zwykłych".

7.3. Ogłoszenie



Rys. 7.3. Strona z ogłoszeniem [opracowanie własne]

Wybrawszy interesujące ogłoszenie, użytkownik zostaje przeniesiony na stronę ze szczegółami. Tutaj przedstawione są wszystkie informacje na temat danego lokalu na przykład: cena za pobyt, liczba pokoi, lista udogodnień. Z poziomu tego widoku jest dostępnych kilka akcji:

- dodanie lokalu do ulubionych,
- wynajem,
- zapoznanie się z recenzjami lub napisanie własnej,
- przejście do profilu właściciela ogłoszenia,
- edycja ogłoszenia.

Pierwsze dwie akcje są dostępne jedynie dla zalogowanego użytkownika. Podobnie jest z pisaniem recenzji z tym, że dodatkowo nieruchomość musiała być w przeszłości wynajęta. Edycji ogłoszenia może dokonać jedynie właściciel lub administrator.

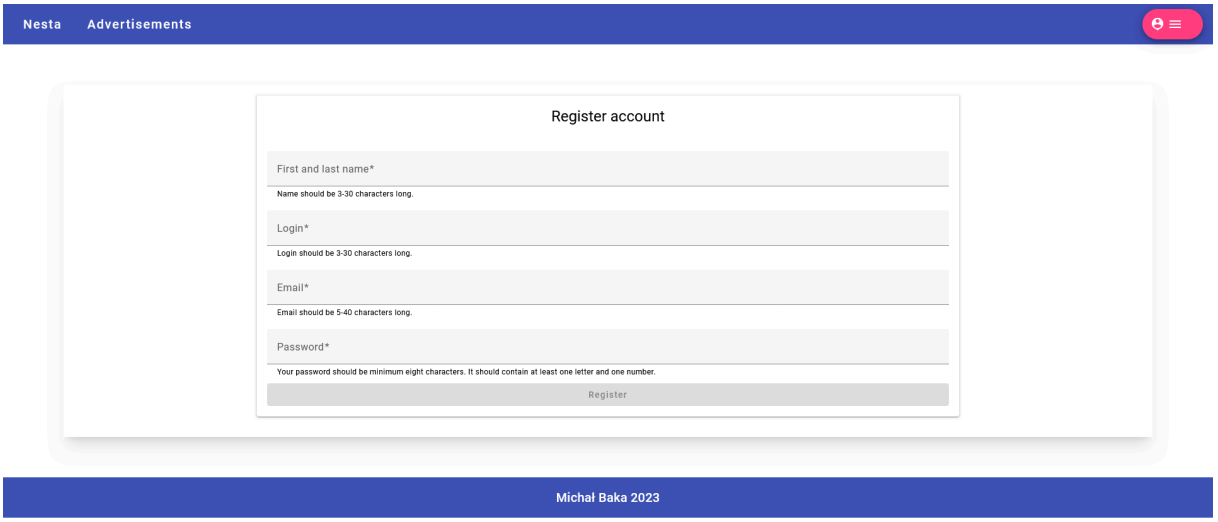
Nie każde ogłoszenie jest dostępne do obejrzenia. Jeśli jest ono ustawione jako niewidoczne, to tylko właściciel oraz administrator może je zobaczyć. Ogłoszenie takie jest oznaczone symbolem przekreślonego oka.

7.4. Recenzje lokalu z ogłoszenia

Rys. 7.4. Widok recenzji danego lokalu [opracowanie własne]

Lista recenzji danego lokalu jest dostępna z poziomu szczegółów ogłoszenia (podrozdział 7.3). Jest ona wyświetlana w oknie modalnym ponad główną zawartością strony. Użytkownik ma możliwość zapoznania się recenzjami innych osób, które spędziły czas w danym lokalu oraz ocenić czy recenzja była wartościowa. Służą do tego przyciski w lewym dolnym rogu widoczne na rysunku 7.4. Po najechaniu na któryś z przycisków, wyświetlona zostaje informacja o liczbie osób, które w dany przycisk kliknęło. Istnieje również możliwość napisania własnej recenzji. Można to zrobić naciskają pasek na dole okna. Pojawia się wtedy odpowiedni formularz. Dla użytkownika, który nie jest zalogowany lub nigdy nie był w danym lokalu, formularz nie będzie dostępny. Po napisaniu recenzji istnieje możliwość jej usunięcia przez osobę, która ją napisała lub administratora.

7.5. Strona z formularzem rejestracji użytkownika



Rys. 7.5. Formularz rejestracji użytkownika [opracowanie własne]

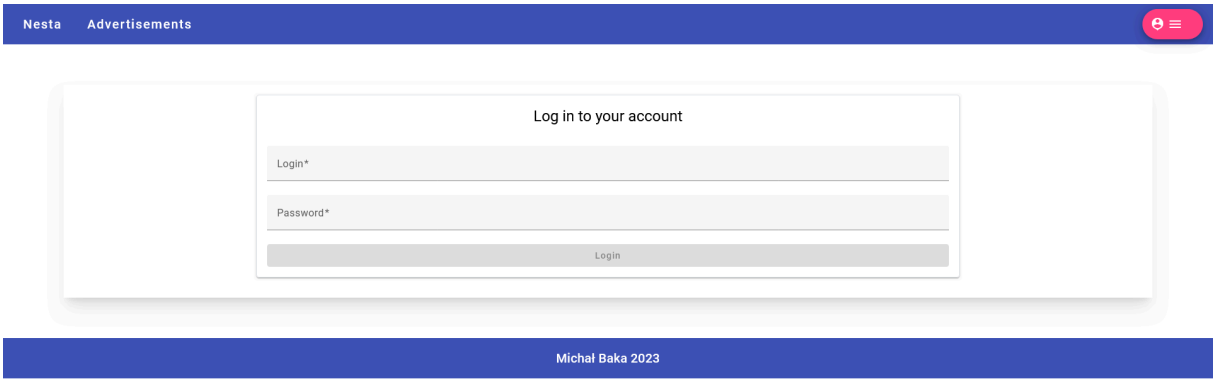
Formularz rejestracji użytkownika umożliwia założenie konta w aplikacji i jest dostępny z poziomu menu znajdującego się w prawym górnym rogu. Konto to jest konieczne do korzystania z pełnej wersji serwisu. Pola zawarte w formularzu to:

- pierwsze i drugie imię,
- nazwa użytkownika,
- adres e-mail,
- hasło.

Wszystkie wymienione pola są wymagane. Ponadto poprawność wprowadzanych danych jest sprawdzana. Dla przykładu pole typu "Email"przyjmie tylko ciąg znaków, który jest poprawnym adresem e-mail. Jeśli tak nie jest to przycisk "Register" jest zablokowany. Podobne mechanizmy zostały zaimplementowane dla pozostałych pól. Jeśli dane przekazane w formularzu będą poprawne, ale pojawią się inne problemy już po stronie serwera informacja o tym zostanie wyświetlona nad formularzem. Przykład takiej sytuacji to zajęta nazwa użytkownika.

Po udanej rejestracji użytkownik jest przenoszony na stronę główną. Na dole ekranu pojawia się również informacja o wysłaniu e-maila z linkiem umożliwiającym aktywację konta. Po wejściu w link aktywacyjny konto jest aktywne i można się na nie zalogować.

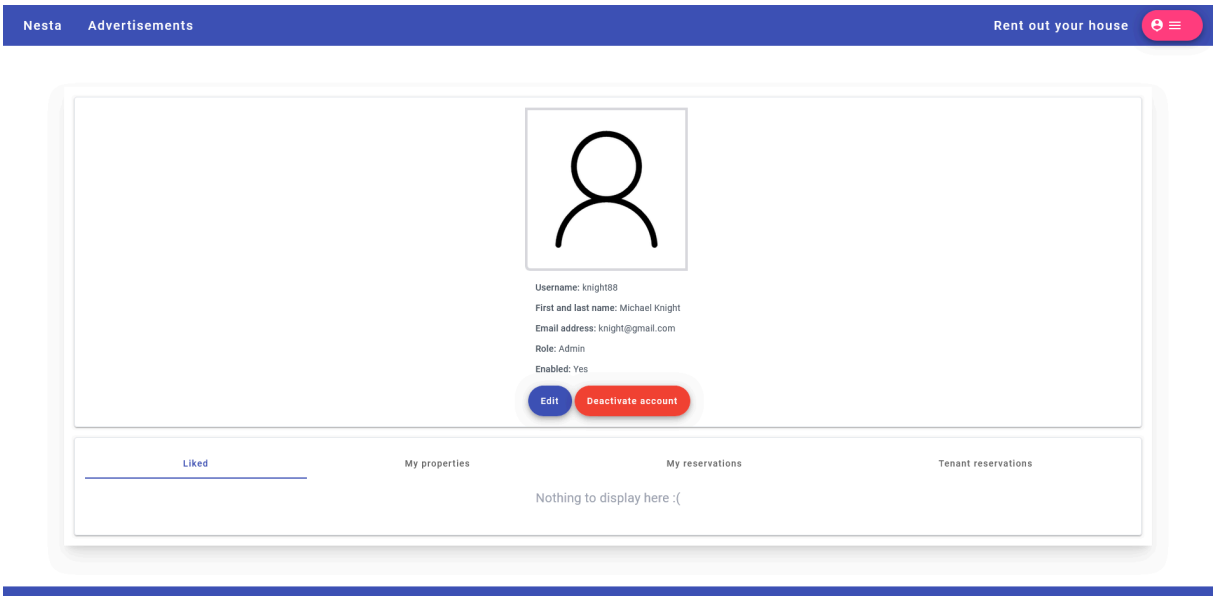
7.6. Strona z formularzem logowania



Rys. 7.6. Formularz logowania użytkownika [opracowanie własne]

Rysunek 7.6 przedstawia formularz logowania. Umożliwia on użytkownikowi dostęp do pełnej wersji serwisu. Oba pola formularza są wymagane do odblokowania przycisku "Login". Jeśli użytkownik poda niepoprawne dane logowania, nad formularzem zostanie wyświetlony komunikat z informacją o błędzie. Formularz jest dostępny z poziomu menu znajdującego się w prawym górnym rogu.

7.7. Strona użytkownika



Rys 7.7. Strona użytkownika [opracowanie własne]

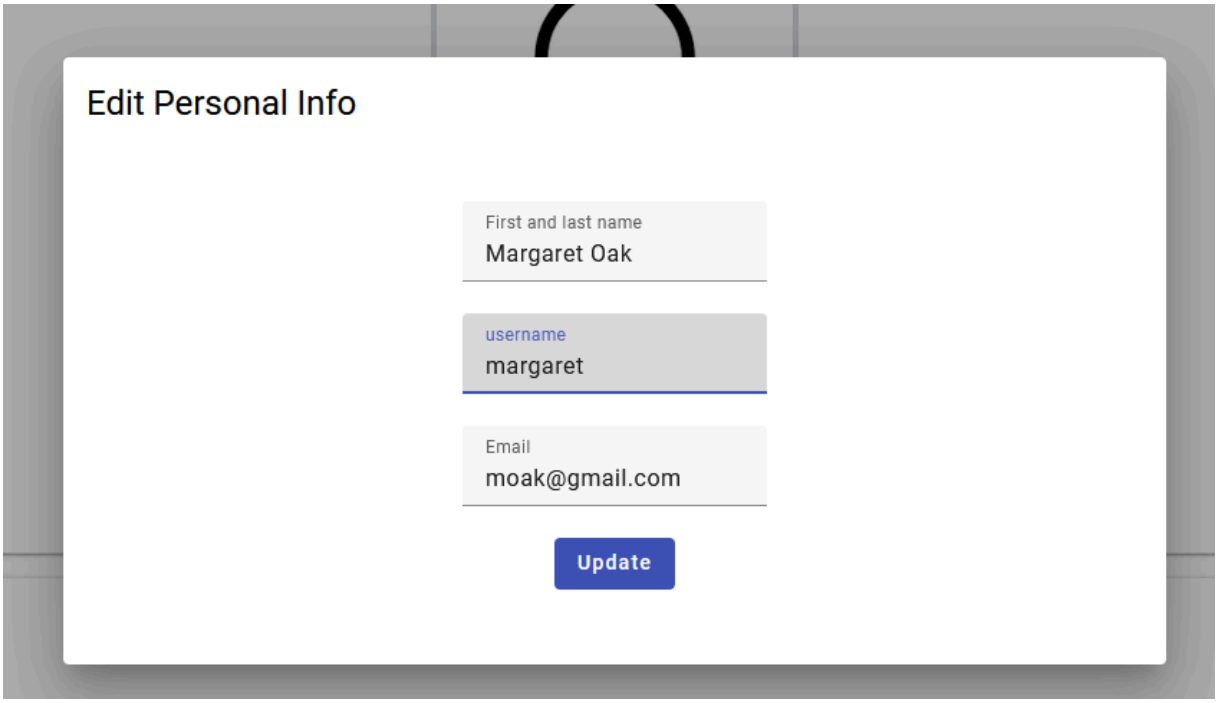
Strona domowa użytkownika to miejsce, gdzie dostępne są dane wybranej osoby. Można tutaj uzyskać dostęp do ulubionych ogłoszeń oraz lokali wystawionych w postaci ogłoszeń przez danego użytkownika.

Jeśli obecnie przeglądany użytkownik jest tym, który jest obecnie zalogowany, to uzyskuje on dostęp do następujących funkcji:

- edycja danych użytkownika,
- wyłączenie konta,
- dostęp do listy wynajmów użytkownika,
- dostęp do listy wynajmów, jakie zrobili inni użytkownicy.

Dostęp do wyżej wymienionych funkcjonalności posiada również administrator.

7.8. Formularz edycji danych użytkownika



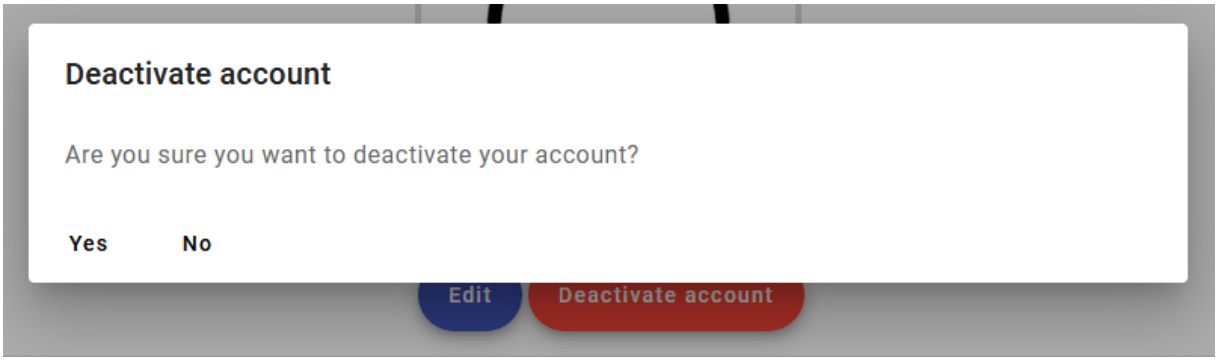
Rys. 7.8. Formularz edycji danych użytkownika [opracowanie własne]

Dialog ukazany na rysunku 7.8 dostępny jest z poziomu strony użytkownika (podrozdział 7.7). Otwierany jest on poprzez naciśnięcie przycisku "Edit". Z poziomu tego okna użytkownik może zmienić swoje dane, które podał w trakcie rejestracji. W przeciwieństwie do formularzy opisanych w podrozdziałach 7.6 i 7.7, pola nie są wymagane. Przesłanie pola z pustą wartością sprawia, że jest ono pomijane przy aktualizacji.

Podobnie jak formularze opisane w podrozdziałach 7.5. i 7.6. także i ten wypisuje ewentualne błędy powyżej pól.

Aby zamknąć okno z formularzem, należy nacisnąć klawisz "Esc" na klawiaturze lub kliknąć ciemny obszar dookoła.

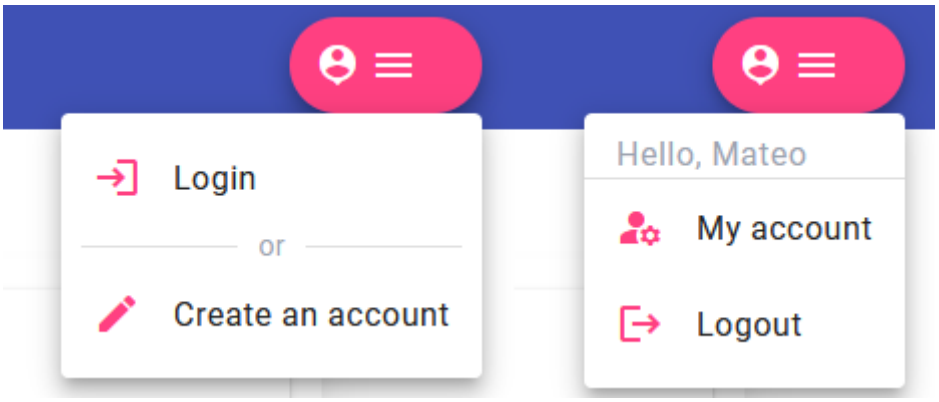
7.9. Dialog dezaktywacji konta



Rys. 7.9. Dialog dezaktywacji konta [opracowanie własne]

Rysunek 7.9 przedstawia okno dialogowe służące do dezaktywacji konta. Opcja ta jest dostępna na stronie użytkownika. Wybranie odpowiedzi twierdzącej blokuje dostęp do konta.

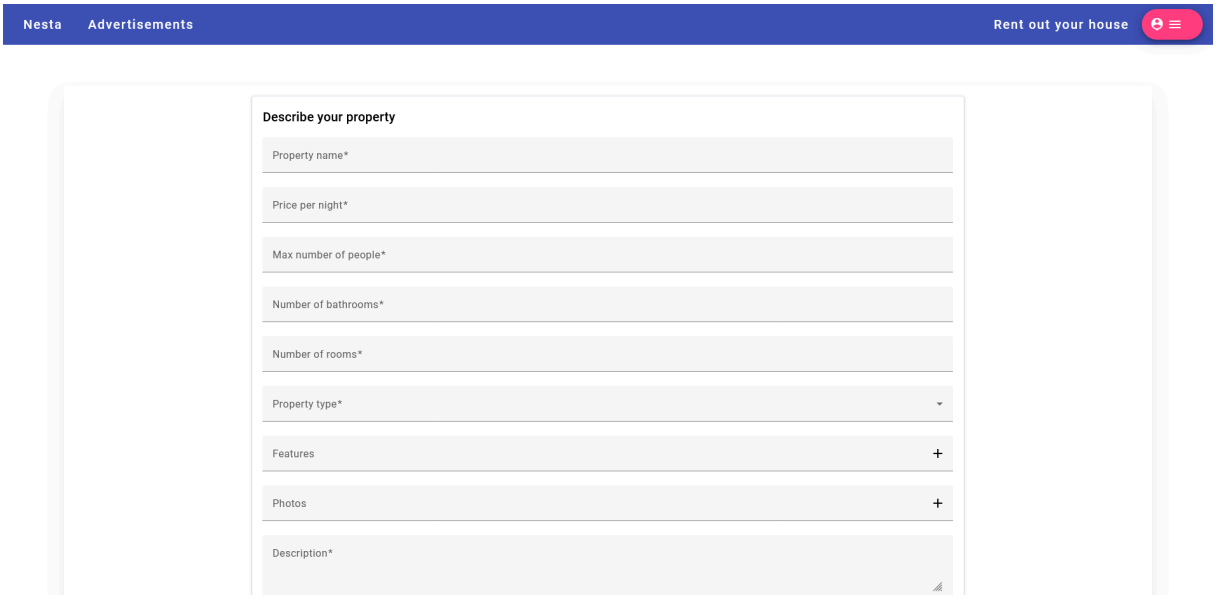
7.10. Menu użytkownika



Rys. 7.10. Menu użytkownika. Po lewej dla niezalogowanego użytkownika, po prawej dla zalogowanego [opracowanie własne]

Menu z rysunku 7.10 umieszczone jest na pasku u góry. Posiada w sobie kilka opcji, które różnią się w zależności od tego, jaki użytkownik z niego korzysta. Niezalogowany użytkownik może założyć konto lub zalogować się na istniejące. Po zalogowaniu wspomniane opcje zastępowane są przez nowe: "My account" i "Logout". "My account" to odnośnik do strony aktualnie zalogowanego użytkownika opisanej w podrozdziale 7.7. Druga z opcji pozwala na wylogowanie się. Po udanym zakończeniu procesu w dolnej części ekranu wyświetlana jest informacja.

7.11. Formularz dodawania nowego ogłoszenia

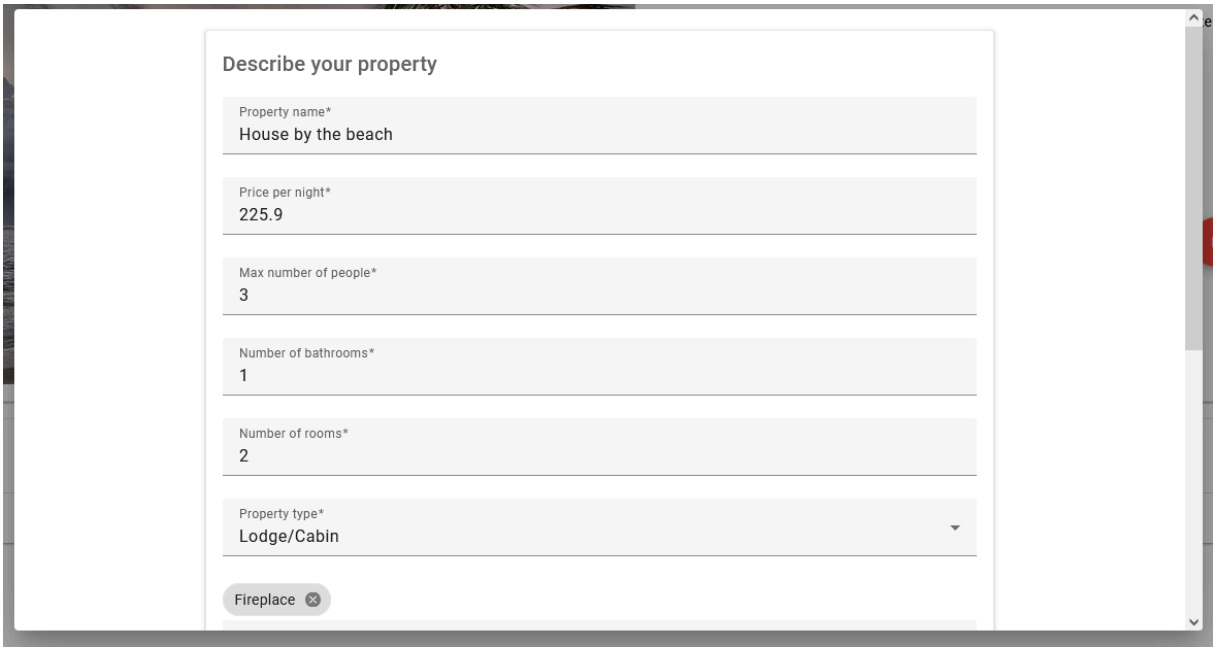


Rys. 7.11. Formularz dodawania nowego ogłoszenia o wynajem lokalu [Opracowanie własne]

Dodawanie nowego ogłoszenia odbywa się z użyciem formularzu przedstawionego na rysunku 7.11. Zawiera on w sobie wszystkie niezbędne informacje na temat nowego lokalu. Formularz zawiera w sobie weryfikację poprawności wprowadzanych informacji. Dodatkowo, gdyby pojawiły się problemy po wysłaniu danych, odpowiedni komunikat zostanie wyświetlony ponad formularzem.

Aby dodać nowe ogłoszenie należy, na pasku na górze strony internetowej, nacisnąć "Rent out your house".

7.12. Edycja ogłoszenia

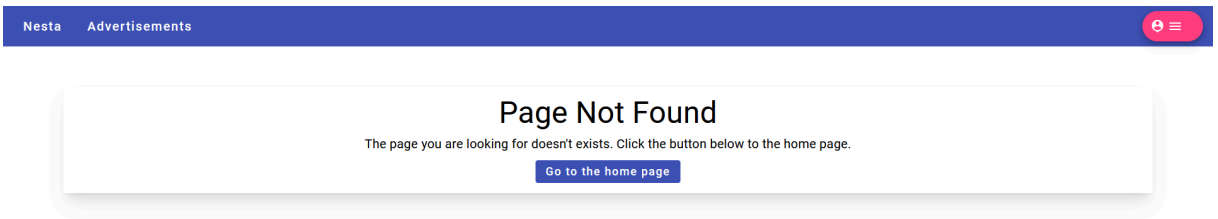


Rys. 7.12. Formularz edycji ogłoszenia [Opracowanie własne]

Edycja ogłoszenia odbywa się przy użyciu tego samego formularza co dodawanie nowego. Istotną różnicą jest brak możliwości zmiany adresu oraz nowe pole. Umożliwia ono ustawienie widoczności ogłoszenia.

Opisywana funkcjonalność dostępna jest z poziomu strony ze szczegółami ogłoszenia. Właściciel oraz administratorzy mają dostęp do przycisku "Edit", który uruchamia okno przedstawione na rysunku 7.12. Wyjście możliwe jest po zatwierdzeniu formularza lub po kliknięciu w ciemny obszar dookoła okna.

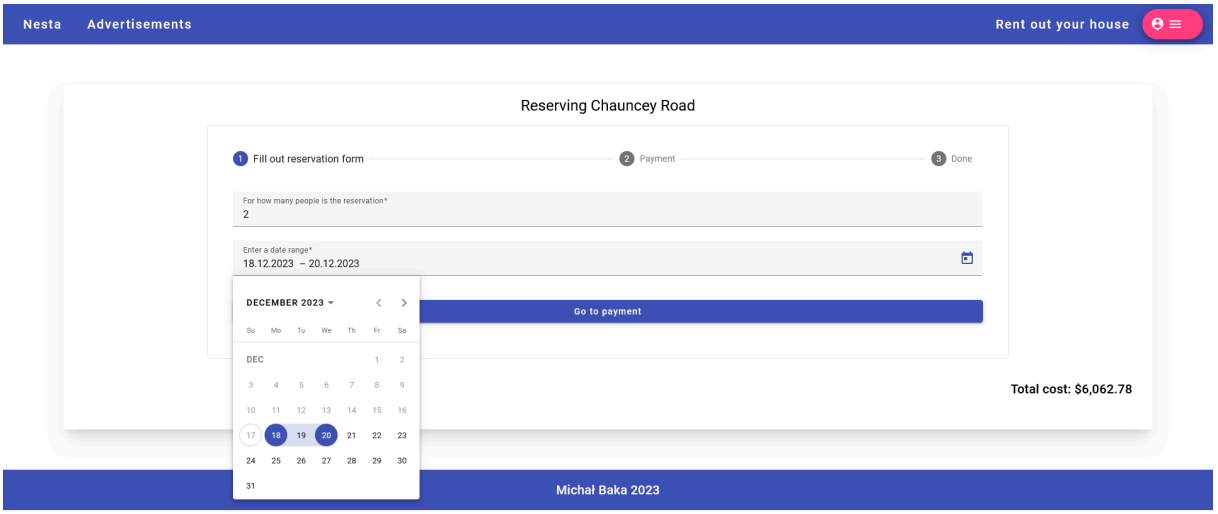
7.13. Nieistniejąca podstrona



Rys. 7.13. Informacja o nieistniejącej podstronie [Opracowanie własne]

Rysunek 7.13. przedstawia prosty komunikat pojawiający się, jeśli użytkownik trafi na nieistniejącą podstronę. Z poziomu tego ekranu można jedynie udać się na stronę główną klikając przycisk "Go to the home page".

7.14. Formularz wynajmu

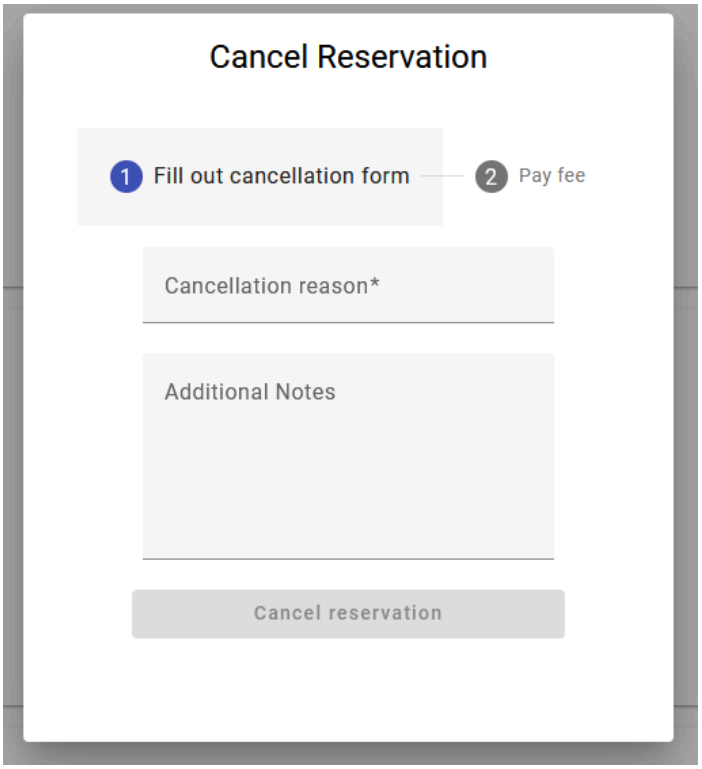


Rys. 7.14. Formularz wynajmu lokalu [Opracowanie własne]

Kluczową funkcjonalnością systemu jest możliwość wynajmu lokalu. Formularz z rysunku 7.14 to implementacja wspomnianej funkcjonalności. Przewidziane zostały dwa pola: liczba osób, dla których ma być wynajem oraz czas w jakim ma się ona odbyć. Formularz sprawdza poprawność wprowadzanych danych. Na przykład nie można wynająć lokalu dla pięciu osób jeśli przewidziano miejsce tylko dla trzech. Po zatwierdzeniu, wykonywane są kolejne testy możliwości dokonania wynajmu. Jednym z nich jest upewnienie się, czy w danym okresie czasu nieruchomość nie jest już wynajęta. W przypadku braku możliwości dokonania wynajmu zostanie wyświetlony stosowny komunikat. W przypadku udanego przejścia przez proces użytkownik przenoszony jest na ekran płatności. W ostatnim kroku wyświetlane jest krótkie podsumowanie.

Formularz jest dostępny na podstronie szczegółów ogłoszenia. Uruchamia go przycisk "Make reservation".

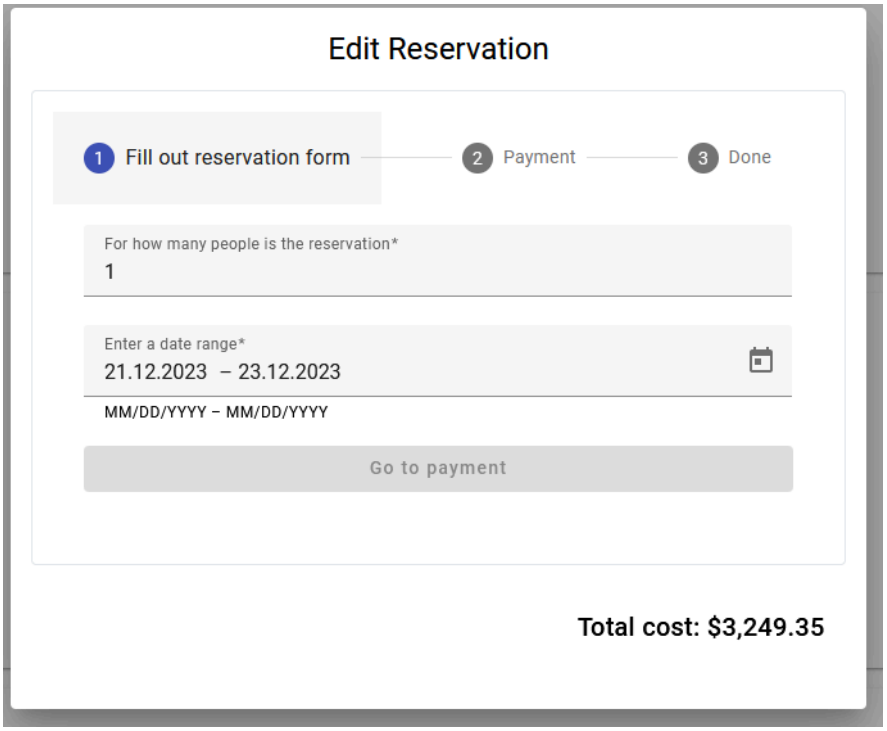
7.15. Formularz anulowania wynajmu



Rys. 7.15. Formularz anulowania wynajmu [Opracowanie własne]

Formularz z rysunku 7.15 służy do anulowania wynajmu. Pierwsze z pól jest wymagane. Należy w nim umieścić powód anulowania wynajmu. Po poprawnym wypełnieniu formularza przycisk na dole zostaje odblokowany. Nie można anulować wynajmu w czasie jego trwania. Okno z przedstawionym formularzem jest dostępne z poziomu zakładki "My reservations" na stronie użytkownika.

7.16. Formularz edycji wynajmu



Rys. 7.16. Formularz edycji wynajmu [Opracowanie własne]

Edycja wynajmu odbywa się przy użyciu formularza opisanego w podrozdziale 7.14. Formularz jest dostępny z poziomu zakładki "My reservations" na stronie użytkownika.

8. Testy

Niniejszy rozdział skupia się na przedstawieniu kilku testów jednostkowych, które znalazły w aplikacji serwerowej. Przykłady te mają za zadanie pokazać jak wygląda testowanie kodu imperatywnego oraz reaktywnego napisanego z wykorzystaniem Spring WebFlux. Do pierwszego wykorzystano JUnit, natomiast do drugiego Reactor Test.

8.1. Test sprawdzający poprawność wygenerowanego tokenu

```
@Test
void generateToken_shouldGenerateValidToken() {
    // given
    when(mockUserDetails.getUsername()).thenReturn( t: "test");
    Map<String, Object> claims = new LinkedHashMap<>();
    claims.put("isAdmin", false);
    claims.put("userId", 5);

    // when
    String generatedToken = jwtService.generateToken(claims, mockUserDetails);

    // then
    assertEquals(TOKEN, generatedToken);
}
```

Rys. 8.1. Test sprawdzający poprawność wygenerowanego tokenu [Opracowanie własne]

Uwierzytelnienie użytkownika odbywa się przy użyciu specjalnego tokenu generowanego przez serwer. Kod z rysunku 8.1. sprawdza, czy wygenerowany token jest identyczny z tokenem przechowywanym w stałej "TOKEN".

Test został podzielony na trzy części. Dane potrzebne do przeprowadzenia testu znajdują się poniżej komentarza o treści "given" i są to "userId" i "isAdmin". Ponadto konfigurowany jest tak zwany *mock*. Jest to specjalny obiekt, który w trakcie testowania pozwala na zredukowanie zależności od innego kodu. W przypadku omawianego testu "mockUserDetail.getUsername()" ma zwrócić ciąg znaków "test". W części "when" wywoływana jest testowana metoda. Ostatnia sekcja zajmuje się weryfikacją, czy wynik otrzymany po wywołaniu metody jest zgodny z oczekiwaniami. Odbywa się to przy użyciu statycznej metody "assertEquals" zapewnionej przez framework JUnit.

8.2. Test sprawdzający czy podany identyfikator był pusty lub miał wartość "null"

```
@ParameterizedTest
@NullAndEmptySource
void findPropertyById_nullOrEmptyId_shouldReturnError(String propertyById) {
    StepVerifier.create(propertyService.findPropertyById(propertyById)) FirstS
        .expectError(IllegalArgumentException.class) StepVerifier
        .verify();
}
```

Rys. 8.2. Test sprawdzający czy podany identyfikator był pusty lub miał wartość "null" [Opracowanie własne]

Aplikacja serwerowa działa w oparciu o Spring WebFlux. Rysunek 8.2. to przykład jednego z najprostszych testów możliwych do wykonania testując kod powstały z użyciem tego frameworku. Podstawą każdego testu jest "StepVerifier", który umożliwia testowanie kodu w krokach. Metoda "create" przygotowuje nowy "StepVerifier" gotowy do testowania. W następnym kroku programista deklaruje to czego oczekuje. W przypadku omawianego testu ma zostać zwrócony obiekt typu "Mono" z błędem o typie "IllegalArgumentException". Na końcu uruchamiana jest weryfikacja zadeklarowanych oczekiwań.

8.3. Test sprawdzający możliwość anulowania wynajmu

```
@ParameterizedTest
@EnumSource(value = ReservationStatus.class, names = {"CANCELED", "COMPLETED"})
void cancelReservation_reservationIsNotActive_shouldReturnError(ReservationStatus status) {
    // given
    var cancelReservationDto = new CancelReservationDto( userId: "321",  reservationId: "123",  cancellationReason: "Reasons",  additionalNotes: null);
    Reservation reservation = getReservation();
    reservation.setReservationStatus(status);

    User user = getUser();
    user.getRoles().clear();
    user.getRoles().add(new Role( roleId: "1",  name: "ROLE_ADMIN"));

    when(mockUserRepository.findById(anyString())).thenReturn(Mono.just(user));
    when(mockReservationRepository.findById(anyString())).thenReturn(Mono.just(reservation));

    // then
    StepVerifier.create(reservationService.cancelReservation(cancelReservationDto)) FirstStep<CancellationDto>
        .expectErrorMatches(throwable -> throwable instanceof ReservationException
            && throwable.getMessage().equals("Reservation cannot be cancelled")) StepVerifier
        .verify();
}
```

Rys. 8.3. Test sprawdzający możliwość anulowania wynajmu [Opracowanie własne]

Rysunek 8.3. przedstawia bardziej złożony przykład testowania reaktywnego kodu. Test ma za zadanie sprawdzić, czy dla wynajmu ze stanem innym niż aktywny możliwe jest jego anulowanie.

Omawiany test to test parametryzowany na co wskazuje adnotacja "@ParametrizedTest". Dzięki temu można sprawdzić kilka przypadków pisząc tylko jedną metodę testującą.

Podobnie jak test opisany w podrozdziale 8.1 tak i tutaj użyto podziału na sekcje. Z racji tego, że wywołanie testowanej metody nie odbywa się w osobnej linii, sekcja “when” nie została uwzględniona. Rezultatem wykonania metody "cancelReservation" powinien być obiekt typu "Mono" z błędem. Oczekiwany wynik został zdefiniowany w metodzie "expectErrorMatches". Dzięki niej możliwe jest sprecyzowanie jaki powinien być oczekiwany błąd otrzymany w wyniku wywołania testowanej metody.

9. Podsumowanie i wnioski

Przedmiotem niniejszej pracy dyplomowej było zaprojektowanie i implementacja aplikacji umożliwiającej publikowanie i przeglądanie ogłoszeń dotyczących lokali do wynajęcia. Realizacja projektu obejmowała trzy główne elementy:

- aplikację serwerową,
- aplikację kliencką (interfejs użytkownika),
- bazę danych.

Wymagania funkcjonalne opisane w rozdziałach od 2.1.1 do 2.1.5 zostały z powodzeniem spełnione. System umożliwia przeglądanie ofert lokali z opcją filtrowania wyników. Ponadto zalogowany użytkownik otrzymuje rekomendację lokali w oparciu o ogłoszenia, które przeglądał oraz przeszłe wynajmy. Aplikacja umożliwia wynajmowanie lokali dodanych przez inne osoby. Wynajem jest na określony okres czasu oraz dla określonej liczby osób, przy czym istnieje opcja edycji lub całkowitego anulowania wynajmu. Po zakończeniu wynajmu użytkownik ma możliwość wyrażenia swojej opinii na temat pobytu w danym lokalu poprzez dostępny formularz. Dodatkowo, w ramach projektu, zaimplementowano system logowania i rejestracji, wraz z aktywacją konta poprzez e-mail.

Wymagania niefunkcjonalne również znalazły swoje miejsce w opisywanej aplikacji. Posiada ona prosty i spójny interfejs użytkownika, działa stabilnie oraz posiada systemy zapewniające bezpieczeństwo użytkowników oraz ich zasobów.

Przedstawiona aplikacja jest w pełni funkcjonalna i spełnia wymagania określone na początku projektu. Są jednak dalsze możliwości jej rozwoju. Pierwsza z nich to system pozwalający na zgłoszenie nieodpowiednich ogłoszeń. Dzięki temu administracja mogłaby w szybszy i bardziej efektywny sposób zwalczać akty wandalizmu w serwisie. Kolejną funkcjonalnością jest komunikator wewnątrz aplikacji. Z pomocą takiego narzędzia zainteresowana osoba mogłaby w łatwy sposób skomunikować się z właścicielem lokalu, aby na przykład dopytać o szczegóły nieujęte w ogłoszeniu. Ostatnią rzeczą jest system powiadomień. Informowałby on o zdarzeniach w aplikacji bez użycia zewnętrznej formy komunikacji, jak na przykład e-mail. Powiadomienia mogłyby dotyczyć chociażby kończącego się wynajmu lub tego, że ktoś go anulował.

Implementacja opisanego systemu pozwoliła na praktyczne wykorzystanie umiejętności zdobytych w trakcie studiów. Umożliwił on także poznanie nowych technologii oraz wykorzystanie ich w praktyce. Rozwiązywanie problemów, które pojawiły się w trakcie prac nad aplikacją było cennym doświadczeniem.

10. Bibliografia

Źródła książkowe

[1] Cay S. Horstmann. *Java. Podstawy.* Helion wydanie X 2016

[2] Craig Walls. *Spring in Action. Sixth Edition* Manning

[3] Ian Robinson, Jim Webber, Emil Eifrem *Graph Databases. New Opportunities for Connected Data. 2nd Edition.* O'Reilly 2015

[4] Yakov Fain, Anton Moiseev. *Angular. Programowanie z użyciem języka TypeScript* Helion wydanie II 2020

Źródła internetowe

[5] *Podróże po UE* [Online] https://european-union.europa.eu/live-work-study/travelling-eu_pl

[6] *Spring Data Neo4j - dokumentacja techniczna* [Online] https://docs.spring.io/spring-data/neo4j/docs/current/reference/html/

[7] *Neo4j - dokumentacja techniczna* [Online] https://neo4j.com/docs/

[8] *JWT – dokumentacja techniczna.* [Online] https://jwt.io/introduction/

[9] *Spring WebFlux - dokumentacja techniczna* [Online] https://docs.spring.io/spring-framework/reference/web/webflux.html

[10] *Spring Security - dokumentacja techniczna* [Online] https://docs.spring.io/spring-security/reference/index.html

[11] *Wymagania funkcjonalne vs. wymagania niefunkcjonalne: Które wymagania są ważniejsze?* [Online] https://www.commint.pl/baza/wymagania-funkcjonalne-vs-wymagania-niefunkcjonalne-ktore-wymagania-sa-wazniejsze

[12] *Spring Framework - dokumentacja techniczna* [Online] https://docs.spring.io/spring-framework/reference/overview.html

[13] *Przykładowy system rekomendacji stworzony przy użyciu bazy danych Neo4j* [Online] https://neo4j.com/developer/cypher/guide-build-a-recommendation-engine/

[14] *Reactor Test - dokumentacja techniczna* [Online] https://projectreactor.io/docs/core/release/reference/index.html

[15] *JUnit 5 - dokumentacja techniczna* [Online] https://junit.org/junit5/docs/current/user-guide/

[16] *JWT - dokumentacja techniczna* [Online] https://jwt.io/

[17] *neo4j-faker - dokumentacja techniczna* [Online]
https://github.com/neo4j-contrib/neo4j-faker

11. Spis rysunków