Państwowa Wyższa Szkoła Zawodowa w Tarnowie

Wydział Politechniczny

Kierunek: Informatyka

Specjalność: Informatyka stosowana

Specjalizacja: Inżynieria systemów informatycznych

2021/2022

Gabriela Jasnosz

PRACA INŻYNIERSKA

Elektroniczny second hand

Promotor pracy:

mgr inż. Tomasz Gądek

Tarnów, 2022

Spis treści

1. Wstęp

Przedmiotem niniejszej pracy dyplomowej jest aplikacja internetowa, będąca serwisem ogłoszeniowym z odzieżą. W kolejnych rozdziałach przedstawione zostaną technologie wykorzystane przy jej tworzeniu, wybrane elementy implementacyjne oraz uzyskane efekty.

1.1. Motywacja

Fast fashion, czyli tak zwana „szybka moda" to model biznesowy, który stosuje wiele marek odzieżowych. Założeniem tego modelu jest masowa produkcja ubrań, będących tańszymi odpowiednikami produktów z wyższej półki. Trendy odzieżowe zmieniają się bardzo szybko, co przekłada się na krótki czas życia produktu. Ubrania są wyrzucane, a w sklepach pojawiają się nowe, odpowiadające panującej modzie. W obecnych czasach szybki rozwój fast fashion prowadzi do wielu zagrożeń, nie tylko dla środowiska, lecz także dla człowieka.

Przemysł odzieżowy jest obecnie drugą co do wielkości gałęzią przemysłu odpowiedzialną za zanieczyszczenie świata. Szacuje się, że jest odpowiedzialny za ok. 8% gazów cieplarnianych oraz 20% całkowitego zużycia wody na świecie. Masowe wyrzucanie ubrań powoduje powstawanie wysypisk śmieci. Jedno z największych wysypisk na świecie znajduje się w Chile, na pustyni Atakama. Szacuje się, że rocznie trafia tam ok. 39 tysięcy ton tekstyliów [11]. Problem dotyczy też ich biodegradacji, ponieważ często tymi tekstyliami są materiały syntetyczne, które potrzebują ok. 200 lat, aby jej ulec. Warto też wspomnieć o fakcie, że postępująca globalizacja doprowadziła do możliwości przeniesienia produkcji odzieży do krajów, gdzie koszty wytwarzania są niskie. Najczęściej prowadzi to do łamania praw człowieka – praca przez wiele godzin, niskie płace, wyzysk dzieci.

Wszystkie opisane przeze mnie fakty doprowadziły do powstania pomysłu promowania slow fashion. Jednym z założeń tego modelu jest moda na niewyrzucanie ubrań.

1.2. Cel pracy

Celem pracy było zaprojektowanie oraz zaimplementowanie systemu pełniącego funkcję odzieżowego serwisu ogłoszeniowego. Utworzona aplikacja promuje dbanie o ekologię poprzez umożliwienie użytkownikom wystawianie na sprzedaż niepotrzebnych ubrań oraz nadawanie im drugiego życia.

Na rynku istnieje wiele serwisów ogłoszeniowych, natomiast niewiele z nich jest ściśle ukierunkowanych na zamieszczanie ubrań. Niesie to ze sobą pewne problemy, gdyż to zagadnienie wymaga szczególnej uwagi np. jeśli chodzi o wybór właściwości wystawianych na sprzedaż przedmiotów np. docelowej płci, rozmiaru oraz kategorii. Im precyzyjniej będzie się

dało opisać wystawiany przedmiot, tym większe prawdopodobieństwo, że nasz przedmiot zostanie zauważony przez zainteresowanego oraz sprzedany. Dlatego jednym z głównych celów aplikacji było to, aby umożliwiała jak najbardziej precyzyjne opisanie dodawanego przedmiotu. Szczegółowe wymagania dotyczące funkcji systemu zostaną opisane w kolejnym rozdziale.

1.3. Zakres pracy

W ramach projektu inżynierskiego zaimplementowany został system, który składa się z następujących modułów:

- aplikacja serwerowa,
- aplikacja internetowa (webowa),
- baza danych.

Aplikacja internetowa łączy się z aplikacją serwerową. Pośrednio tworzą jedną całość dzięki komunikacji przy pomocy żądań HTTP (ang. *Hypertext Transfer Protoco*l) opisanych szczegółowo w utworzonej dokumentacji API (ang. *Application Programming Interface*) oraz protokołowi WebSocket. Najważniejsze funkcjonalności aplikacji zostały przetestowane za pomocą testów jednostkowych oraz integracyjnych.

Wszystkie z wymienionych części aplikacji zostaną szczegółowo przedstawione w dalszej części pracy.

2. Analiza biznesowa

W niniejszym rozdziale zaprezentowane zostaną założenia dotyczące funkcjonalności systemu, będącego przedmiotem tej pracy.

2.1. Wymagania funkcjonalne

Główną funkcją, bez której utworzony system nie mógłby nazywać się serwisem ogłoszeniowym, jest możliwość wystawiania na sprzedaż własnych przedmiotów. Aplikacja powinna oferować możliwość dodania nowego przedmiotu oraz nadania mu atrybutów takich jak np. docelowa płeć, kategoria i rozmiar. Aplikacja powinna także umożliwić przesyłanie zdjęć przedmiotów, które użytkownicy chcą sprzedać oraz dać możliwość wyboru zdjęcia głównego. Użytkownicy aplikacji powinni być w stanie przeglądać dodane ogłoszenia, a także filtrować je według swoich potrzeb – dlatego nadanie atrybutów przy tworzeniu ogłoszenia jest bardzo istotne.

Z punktu widzenia użytkownika aplikacji ważne jest, aby umożliwiała ona kontakt z innymi. Jeżeli daną osobę zainteresuje zamieszczone ogłoszenie, powinna ona być w stanie napisać wiadomość do jego właściciela, aby nie musieć tego robić np. telefonicznie. Wszystkie rozmowy powinny być zapisywane po to, aby użytkownik mógł w każdym momencie wrócić do swoich konwersacji.

Kolejną ważną funkcją, którą system powinien zapewniać, jest utworzenie własnego konta. Użytkownicy nieposiadający konta powinni być w stanie oglądać przedmioty dodawane przez użytkowników, nie powinni jednak być w stanie dodawać własnych oraz kontaktować się z innymi osobami. Z jednej strony wpłynie to pozytywnie na bezpieczeństwo aplikacji, a z drugiej zachęci użytkowników do założenia własnego konta, aby zyskać więcej możliwości.

Użytkownicy powinni móc zarządzać własnym profilem – dodawać dane teleadresowe, czy też zmieniać zdjęcie profilowe. Jeżeli posiadają utworzone ogłoszenia, powinni być w stanie nimi zarządzać, czyli np. usuwać lub zmieniać ich właściwości.

Następną ważną funkcją systemu jest możliwość przeglądania profili innych osób. Wchodząc na profil innego członka aplikacji, powinna zostać udostępniona możliwość dodania oceny wraz z komentarzem. Funkcja ta wpłynie pozytywnie na możliwość weryfikacji uczciwości danej osoby.

Inną akcją, którą zalogowani użytkownicy powinni być w stanie wykonać, jest dodanie innych osób do listy obserwowanych. Dzięki temu będą w stanie potem wrócić do profili tych osób, bez potrzeby zapamiętywania ich samemu. Przedmioty tych użytkowników będzie można w łatwy sposób wyświetlić, filtrując wyniki wyszukiwania przedmiotów.

Ostatnią funkcją systemu, którą powinien zapewniać, jest możliwość logowania się do panelu moderatora uprawnionych do tego osób. Osoby te powinny mieć możliwość zarządzania przedmiotami dodanymi w aplikacji, czyli powinny mieć opcję ich usuwania, jeżeli naruszają standardy systemu.

2.2. Wymagania niefunkcjonalne

2.2.1. Bezpieczeństwo

Podstawowym wymaganiem aplikacji jest jej bezpieczeństwo, stąd potrzebna jest implementacja systemu autoryzacji i uwierzytelnienia. Uwierzytelnienie będzie miało za zadanie weryfikację tożsamości użytkownika chcącego zalogować się na swoje konto, natomiast proces autoryzacji będzie weryfikował czy zalogowany użytkownik posiada prawo dostępu do określonych zasobów aplikacji.

Istotną kwestią jest również proces tworzenia nowego konta – każdy użytkownik tworzący nowe konto musi powiązać je ze swoim adresem e-mail. Konto nie powinno być aktywne, dopóki użytkownik nie potwierdzi, że podany e-mail należy do niego.

2.2.2. Wydajność

Ważnym aspektem jest również wydajność aplikacji, czyli prędkość jej działania. Aplikacja będzie przechowywała wiele zdjęć przesyłanych przez klientów, stąd ważną sprawą jest zadbanie o optymalizację ich przechowywania oraz udostępniania.

2.2.3. Przejrzysty interfejs aplikacji

Interfejs aplikacji powinien być przejrzysty oraz jasny dla każdego użytkownika. Poszczególne jego elementy powinny być projektowane w taki sposób, aby mogły być użyte ponownie. Dzięki temu zapewniona zostanie spójność poszczególnych widoków.

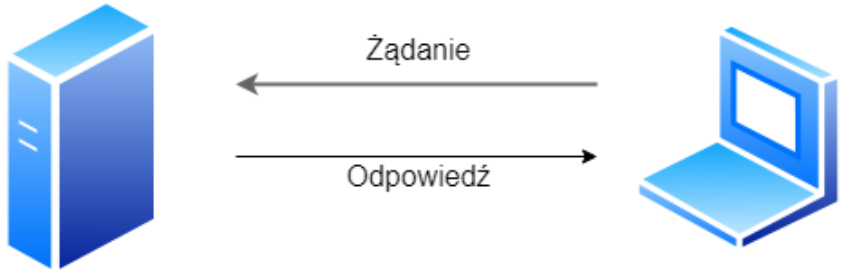
3. Architektura systemu

Aplikacja została napisana w oparciu o architekturę klient – serwer. Polega ona na utworzeniu dwóch modułów: aplikacji serwera oraz aplikacji klienta. Klient wysyła żądanie do serwera, natomiast serwer udostępnia zasoby, pobierając je z bazy danych. Połączenie między modułami jest możliwe dzięki protokołom komunikacyjnym. W aplikacji wykorzystane zostały dwa protokoły – HTTP (ang. *Hypertext Transfer Protocol*) oraz WebSocket. Oba protokoły działają w czwartej warstwie (warstwie aplikacji) modelu TCP/IP.

3.1. Protokół HTTP

HTTP, czyli protokół przesyłania danych hipertekstowych odpowiada za transmisję dokumentów hipermedialnych takich jak np. HTML. Klient rozpoczyna połączenie z serwerem, następnie wysyła do niego żądanie, czeka na odpowiedź od serwera, a gdy ją otrzyma, połączenie jest zamykane. Jedną z ważniejszych cech protokołu jest bezstanowość. Cecha ta oznacza, że pomiędzy żądaniami serwer nie przechowuje żadnych danych [2].

Rysunek 3.1 przedstawia schemat działania protokołu HTTP.

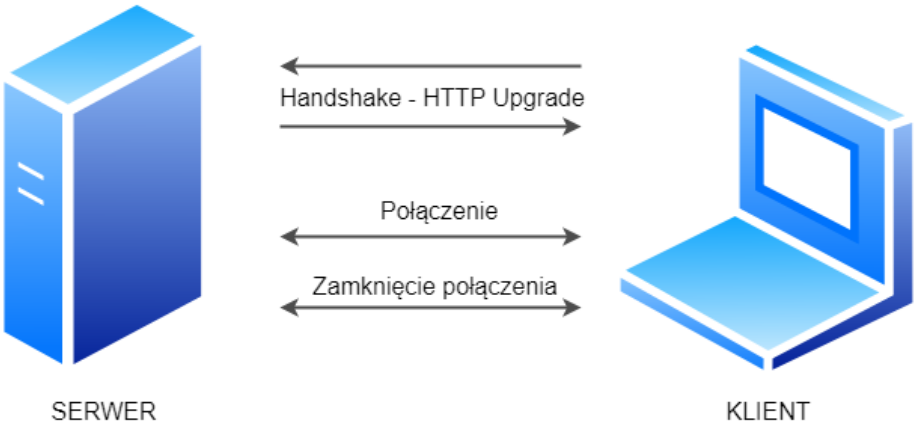


Rys. 3.1. Schemat protokołu HTTP [opracowanie własne]

3.2. Protokół WebSocket

WebSocket to protokół komunikacyjny, który zapewnia dwukierunkową wymianę informacji poprzez jedno połączenie. Porównując do HTTP, wymiana danych między aplikacją klienta, a serwerem odbywa się przy małym obciążeniu [22]. W aplikacji protokół WebSocket został wykorzystany do implementacji funkcji wysyłania wiadomości między użytkownikami. Dzięki jego zastosowaniu umożliwiona została asynchroniczna komunikacja między nimi.

Rysunek 3.2 przedstawia schemat działania protokołu WebSocket.



Rys. 3.2. Schemat WebSocket [opracowanie własne]

Połączenie rozpoczyna się, kiedy klient wysyła żądanie handshake (uścisk dłoni). Jest żądaniem HTTP po to, aby zapewnić kompatybilność z oprogramowaniem serwera oraz pośrednikami. Dzięki temu pojedynczy port może być używany jednocześnie przez klientów HTTP oraz WebSocket. Po ustanowieniu połączenia, serwer i klient są w stanie nadawać oraz odbierać wiadomości w trybie pełnego dupleksu co oznacza, że wiadomości mogą być wysyłane jednocześnie bez utraty transferu [22]. Trwa to do momentu, gdy jedna ze stron postanowi zamknąć połączenie. W utworzonej aplikacji kontakt jest przerywany, kiedy użytkownik wyloguje się.

4. Analiza technologiczna

W rozdziale zostaną opisane najważniejsze technologie wykorzystane do utworzenia aplikacji, będącej przedmiotem tej pracy.

4.1. Aplikacja serwerowa

4.1.1. Java

Aplikacja serwerowa została w całości napisana w języku Java. Jest to język programowania, który powstał w 1991 roku w firmie Sun Microsystems. Składnia Javy została zaczerpnięta z języka C, natomiast wiele elementów dotyczących obiektowości z języka C++ [5].

Podstawowe założenia języka, to:

- obiektowość — oznacza to, że w Javie wszystko jest obiektem. Cecha ta wpływa na łatwość modyfikacji oraz możliwości rozbudowy oprogramowania,
- przenośność — programy napisane w tym języku, mogą być uruchamiane w różnych systemach,
- otwartość — standard języka jest otwarty, a SDK (ang. *Software Development Kit* — zestaw narzędzi niezbędny do korzystania z danej technologii) dla rozwiązań niekomercyjnych jest darmowy w każdej wersji [14].

Początkowo język miał być wykorzystany do tworzenia oprogramowania urządzeń domowego użytku, natomiast wraz z rozwojem komputerów oraz Internetu uznano, że bardziej przydatny będzie w obszarze aplikacji desktopowych oraz internetowych [5]. Od samej chwili powstania do dzisiaj, Java zajmuje wysokie miejsca w rankingu popularności. Według indeksu społeczności programistów TIOBE Java znajduje się obecnie na trzecim miejscu na liście [3]. Na przestrzeni lat Java ewoluowała. W aplikacji, będącej przedmiotem tej pracy została wykorzystana Java w wersji 15.

4.1.2. Maven

Maven jest narzędziem wykorzystywanym do zarządzania projektem oprogramowania. Ułatwia nam dodawanie zależności, ponieważ wszystkie definiowane są w jednym pliku – pom.xml. Oprócz tego Maven zarządza również raportowaniem, budowaniem oraz dokumentacją aplikacji [10].

4.1.3. Spring

Spring jest szkieletem tworzenia aplikacji w języku Java. Podobnie jak Java, jest technologią open source (otwartą, darmową). Jego najważniejszą funkcją jest ułatwienie programowania w Javie, na co wpływa m.in. możliwość korzystania ze wstrzykiwania zależności [19]. Wstrzykiwanie zależności jest implementacją paradygmatu IoC (ang. *Inversion Of Control* – odwrócenie sterowania). Paradygmat ten zakłada odwrócenie sterowania wykonywaniem aplikacji do używanego frameworku [4]. W przypadku wstrzykiwania zależności, odpowiedzialność kontroli cyklu życia obiektów jest przekazana do kontenera Springa. Kontener ten zarządza tworzeniem obiektów, tak zwanych beanów, a także tworzy powiązania pomiędzy poszczególnymi komponentami. Dzięki temu każdy obiekt otrzymuje swoje zależności w momencie jego tworzenia od kontenera. Obiekty nie muszą same tworzyć zależności [19].

W aplikacji swoje wykorzystanie znalazł Spring Boot, czyli rozszerzenie frameworka Spring. Spring Boot zapewnia automatyczną konfigurację oraz ułatwia sam proces tworzenia aplikacji dzięki istnieniu starterów Spring Boot.

4.1.4. JSON Web Token

JWT, czyli JSON Web Token jest otwartym standardem przesyłania informacji, opartym na JSON (ang. *JavaScript Object Notation*) [9]. JSON z kolei jest uniwersalnym formatem tekstowym bazującym na podzbiorze języka JavaScript [7].

Informacje przesyłane przez JWT są podpisywane cyfrowo za pomocą algorytmu HMAC oraz tajnego klucza, stąd też są bezpieczne oraz zweryfikowane [9].

Token składa się z trzech części:

- header (nagłówek) – zawiera informacje o typie tokena oraz algorytmie, który był użyty do jego podpisania,
- payload (dane właściwe) – zawiera tak zwane claims (deklaracje, prawa), które można zdefiniować dowolnie, natomiast zazwyczaj przechowuje się tam informacje m.in. o dacie ważności tokena oraz roli użytkownika, dla którego token był wystawiony,
- signature (podpis) – podpis serwera tworzony na podstawie tajnego klucza oraz danych umieszczonych w dwóch pierwszych częściach tokena [9].

4.1.5. Java Persistence API

Java Persistence API to narzędzie umożliwiające mapowanie obiektowo – relacyjne, utworzone do pracy w języku Java [21]. Polega ono na przekształcaniu obiektów klas

napisanych w języku Java na encje bazy danych. Jest ono możliwe dzięki wykorzystaniu udostępnionych adnotacji. Przykładowe adnotacje:

- @Entity – klasa nią oznaczona oznacza, że jest ona reprezentacją obiektu bazodanowego,
- @OneToOne – reprezentuje powiązanie między obiektami typu jeden do jednego,
- @Id – oznacza się nią pole, będące identyfikatorem danej encji.

4.1.6. Spring Data JPA

Spring Data JPA to jeden z modułów Spring Data. Oparty jest o standard JPA. Wykorzystywany jest przy pracy w warstwie dostępu do bazy danych. Jego głównym celem jest ograniczenie powielania kodu oraz utrzymanie jego zwięzłości, oraz jasności. Ułatwia wykonywanie operacji na bazie danych z poziomu aplikacji poprzez udostępnienie specjalnych interfejsów, posiadających wbudowane metody oraz funkcje. Jednymi z cenniejszych ich funkcji są przekształcanie nazwy funkcji na zapytanie SQL, a także metody umożliwiające paginację oraz sortowanie [18].

4.1.7. JUnit

JUnit to biblioteka służąca do tworzenia testów w Javie. Narzędzie udostępnia szereg adnotacji pozwalających na szybką ich konfigurację. Daje możliwość testowania zarówno pojedynczych klas oraz funkcji, jak i też kilku z nich naraz. Testy napisane w ramach omawianej aplikacji zostały napisane z wykorzystaniem biblioteki JUnit 5.

4.1.8. Swagger

Swagger jest zbiorem narzędzi pomocnych w projektowaniu, budowaniu oraz dokumentowaniu interfejsów REST API. Specyfikacja Open API, według której zbudowany został Swagger, pozwala na opisanie takich cech interfejsu API jak dostępne metody, ich parametry wejściowe oraz wyjściowe, a także metody uwierzytelnienia [20].

4.2. Aplikacja klienta

4.2.1. JavaScript

JavaScript jest skryptowym językiem programowania, wykorzystywanym m.in. przy tworzeniu stron internetowych oraz implementacji interakcji pomiędzy jej elementami. Został opublikowany w 1995 roku przez firmę Netscape. Początkowo miał on znaleźć swoje zastosowanie jako technologia pozwalająca na dodawanie programów do stron internetowych w przeglądarce, będącej produktem firmy – Netscape Navigator, lecz z czasem został

wprowadzony w pozostałych przeglądarkach internetowych. W 1996 roku ustandaryzowano specyfikację języka JavaScript i wydano standard ECMAScript [23]. Od tego czasu wydano wiele wersji języka. W 2012 roku standard ECMAScript w wersji 5.1 stał się obsługiwany przez wszystkie przeglądarki [6].

Na podstawie JavaScript powstało wiele frameworków i bibliotek, wspomagających tworzenie i rozwój aplikacji internetowych. Do najpopularniejszych należą Angular, Vue.js, a także React.js, który został wykorzystany do implementacji aplikacji klienta w omawianym projekcie.

4.2.2. HTML + CSS

HTML (ang. *HyperText Markup Language*) to język znaczników, używany przy tworzeniu dokumentów hipertekstowych, natomiast CSS (ang. *Cascading Style Sheets*) to język arkuszy stylów. Obie technologie są odpowiedzialne za interfejs użytkownika w aplikacjach. HTML odpowiada za strukturę, natomiast CSS za wygląd poszczególnych elementów. Kody obu języków są interpretowane przez przeglądarki WWW i tak powstaje interfejs, który widzimy na stronach internetowych [1].

Zamiast czystego CSS w projekcie swoje wykorzystanie znalazła technologia SASS (ang. *Syntactically Awesome Stylesheets*), czyli preprocesor języka CSS. SASS posiada składnię w pełni zgodną z CSS, dostarczając jednocześnie wielu nowych użytecznych funkcji jak np. możliwość deklaracji zmiennych czy też zagnieżdżania się reguł [17].

4.2.3. React.js

Do implementacji aplikacji klienta użyto biblioteki React.js. Jest to biblioteka JavaScript, która służy do budowania interfejsów użytkownika. Została wydana w 2013 roku przez firmę Facebook. Struktura aplikacji napisanych za jej pomocą opiera się na tworzeniu odizolowanych komponentów, które zarządzają własnym stanem. React korzysta ze składni JSX, czyli rozszerzenia składni JavaScript, możliwością używania znaczników HTML. JSX reprezentuje ideę biblioteki React, według której nie powinno oddzielać się struktury od logiki aplikacji [16].

Główną zaletą Reacta jest jego wydajność, a wpływa na to wykorzystanie tak zwanego wirtualnego modelu DOM (ang. *Document Object Model* – obiektowy model dokumentu np. HTML). Wirtualny DOM to koncepcja, według której wirtualny odpowiednik interfejsu użytkownika jest przechowywany w pamięci i synchronizowany z realnym modelem DOM. Dzięki niej, kiedy tworzony jest nowy komponent, nie jest on umieszczany bezpośrednio w docelowym DOM, lecz w wirtualnym. Wpływa to na szybkość procesu, ponieważ kiedy

następuje synchronizacja modelu wirtualnego z rzeczywistym, ma miejsce też proces uzgadniania, dzięki któremu wykonywana jest minimalna liczba akcji, aby rzeczywisty DOM był aktualny [15].

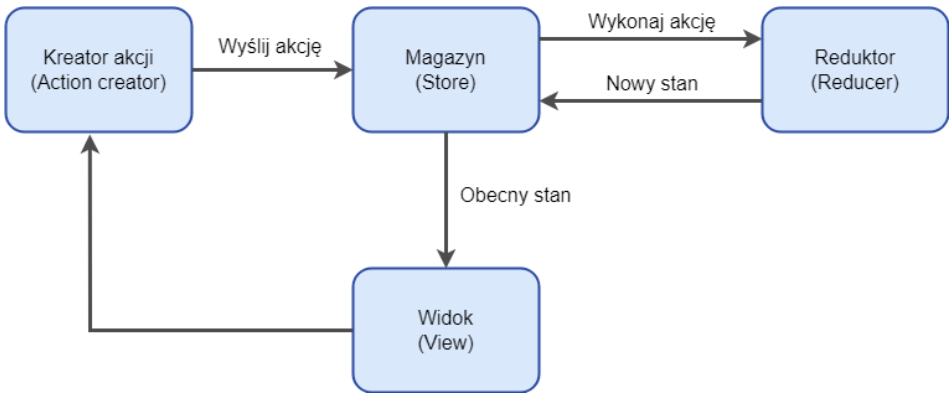
4.2.4. Redux

Redux to biblioteka służąca do zarządzania stanem aplikacji. Jest implementacją architektury Flux, której koncepcja opiera się przede wszystkim na jednokierunkowym przepływie informacji. Według tego wzorca cały stan aplikacji przechowywany jest w tzw. magazynie (store). Przechowywany w nim stan można z łatwością odczytać z wielu miejsc aplikacji. Ciężej jest natomiast go zmienić. Można to zrobić tylko wtedy, gdy wywołamy akcję. Jest to zaletą, ponieważ dzięki temu ograniczamy dostęp do zmiennych. Stan aplikacji jest też w tej sytuacji bardziej chroniony.

Ważne w Reduxie jest też utworzenie tak zwanego reduktora, czyli pośrednika między stanem aplikacji, a jej logiką. Reduktor określa, w jaki sposób zdefiniowany jest stan oraz co ma się stać po wykonaniu konkretnej akcji [15]. Zastosowanie biblioteki Redux ma wiele zalet m.in.:

- ułatwienie zarządzania zależnościami aplikacji,
- uniknięcie sytuacji, w której po zmianie stanu, wiele komponentów zostaje niepotrzebnie zaktualizowanych, dzięki temu, że stan nie musi być już przekazywany z komponentu do komponentu,
- uproszczenie struktury interfejsu użytkownika [15].

Rysunek 4.1. przedstawia uproszczony schemat przepływu danych w bibliotece Redux.



Rys. 4.1 Przepływ danych w bibliotece Redux [opracowanie własne]

4.3. Baza danych

4.3.1. PostgreSQL

W projekcie inżynierskim została zastosowana baza danych PostgreSQL. Jest to relacyjna oraz obiektowa baza danych, która powstała w 1995 roku na Uniwersytecie Kalifornijskim w Berkeley. Cała jej struktura składa się z pojedynczych obiektów reprezentowanych przez tabele, a każda tabela składa się z kolumn, które reprezentują kolejne atrybuty obiektu. Pomiędzy tabelami mogą istnieć powiązania — na tym polega relacyjność. Związki te są tworzone poprzez sformułowanie tak zwanych kluczy obcych. Klucz obcy jest kolumną, która odwołuje się do unikatowej kolumny w innej tabeli, którą najczęściej jest jej klucz główny [13].

Do największych zalet tej bazy danych należą:

- elastyczność – daje możliwość definiowania własnych typów oraz budowy niestandardowych funkcji,
- skalowalność – zarówno pod względem liczby danych, które może przechowywać, jak i też liczby użytkowników, którzy mogą z niej jednocześnie korzystać [12].

5. Implementacja systemu

5.1. Zakres funkcji

Utworzony system posiada trzech podstawowych aktorów:

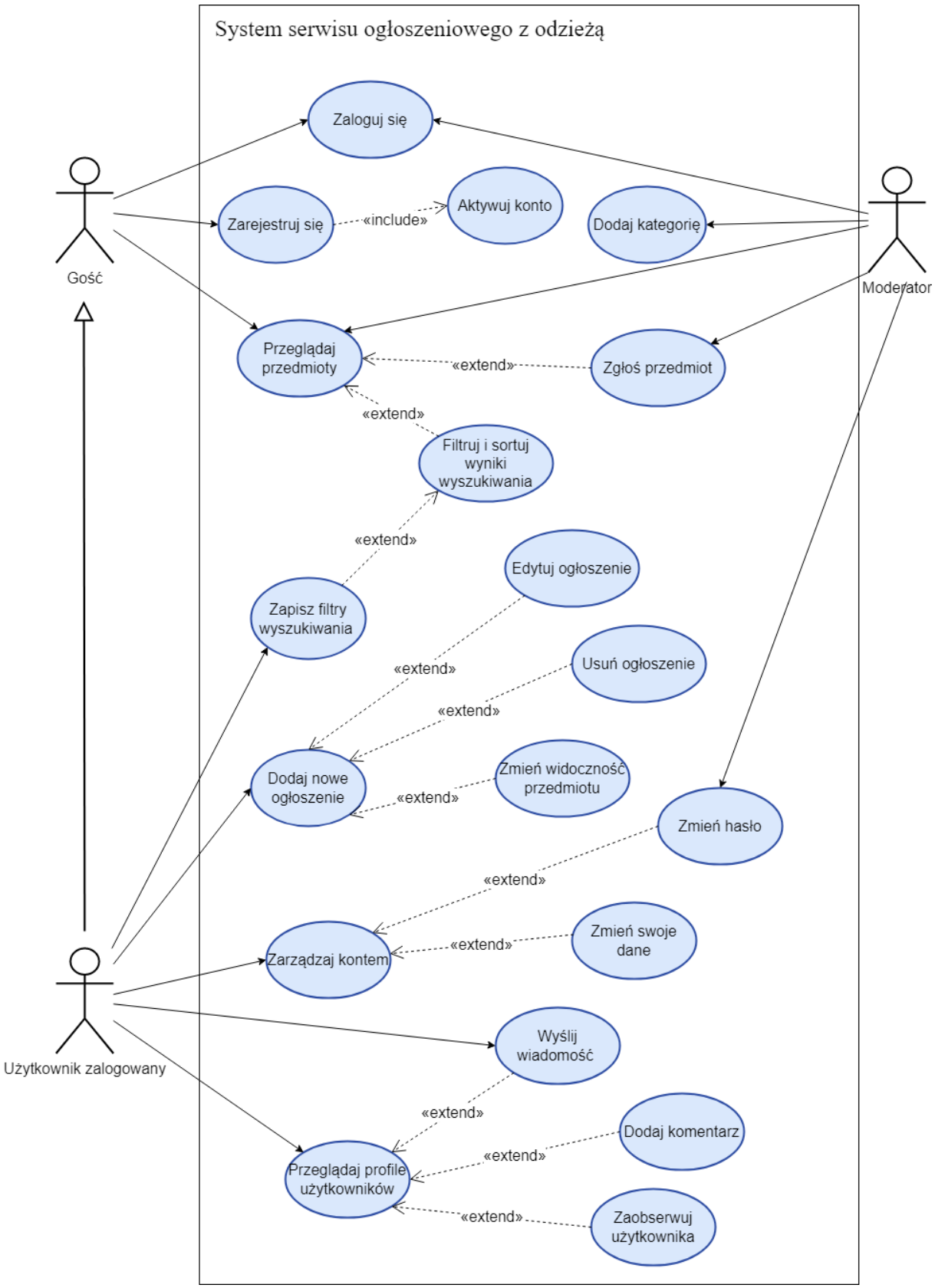
- gość (użytkownik niezalogowany),
- użytkownik zalogowany,
- moderator.

Użytkownicy niezalogowani to użytkownicy z najniższym poziomem uprawnień. Mają dostęp do listy przedmiotów, więc mogą je przeglądać oraz filtrować wyniki. Posiadają także możliwość utworzenia nowego konta oraz jego aktywacji. Po pomyślnym aktywowaniu konta zyskują możliwość logowania.

Kiedy użytkownik loguje się na swoje konto, otrzymuje nowe możliwości. Poza funkcją przeglądania oraz filtrowania przedmiotów ma uprawnienia do zapisywania wybranych filtrów wyszukiwania, oraz dodawania własnych ogłoszeń. Gdy doda ogłoszenie, jest w stanie nim zarządzać, czyli m.in. edytować właściwości wystawionego przedmiotu. Posiada też funkcję przeglądania profili pozostałych użytkowników oraz kontaktowania się z nimi.

Ostatnim aktorem jest moderator, czyli osoba zarządzająca zasobami systemu. Moderator ma uprawnienia do zarządzania dodanymi ogłoszeniami – jest w stanie usuwać ogłoszenia, jeżeli naruszają standardy systemu. Może także dodawać nowe kategorie.

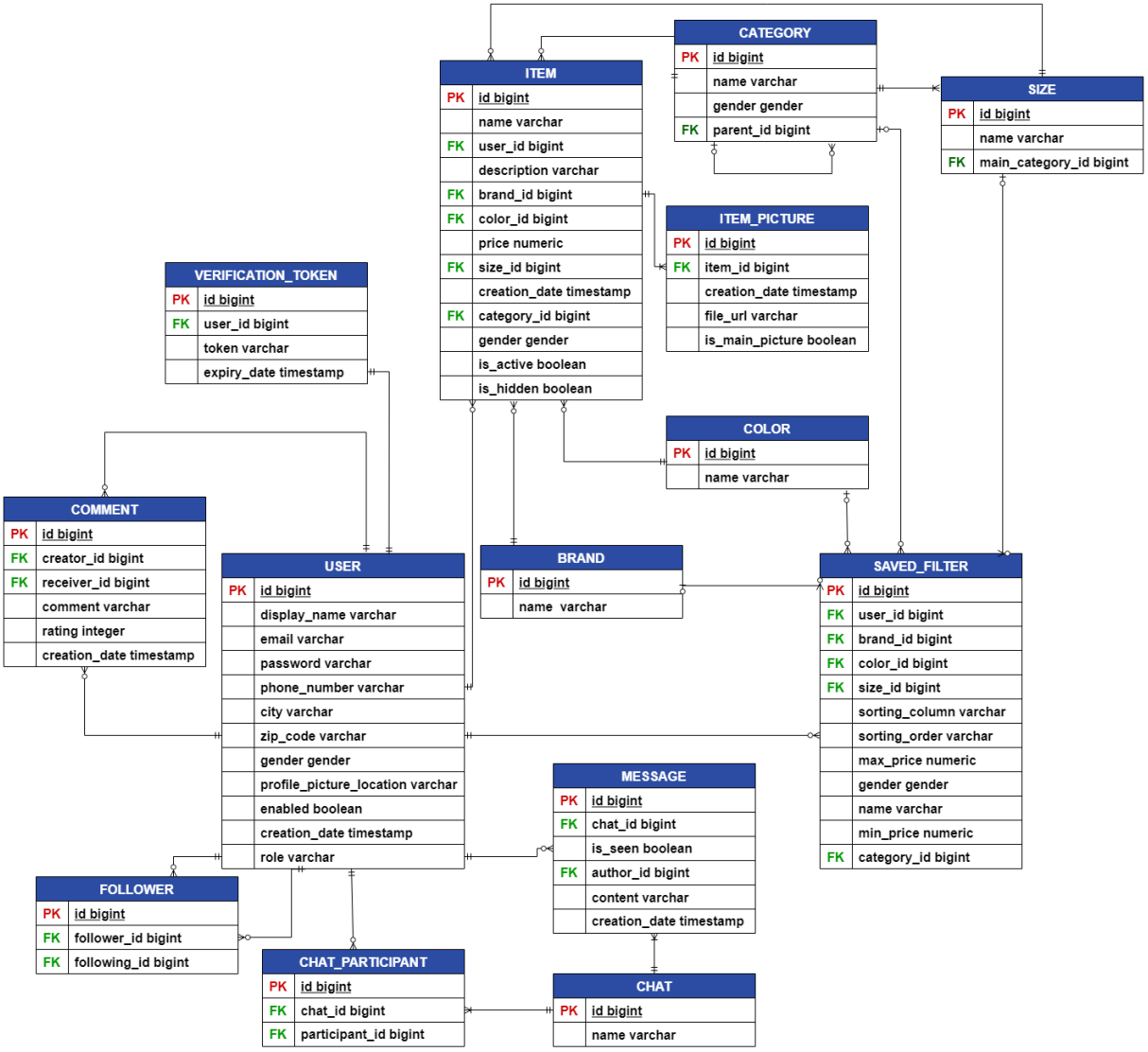
Cała funkcjonalność systemu została przedstawiona za pomocą diagramu przypadków użycia zamieszczonego na rysunku 5.1.



Rys. 5.1. Diagram przypadków użycia [opracowanie własne]

5.2. Diagram ERD

ERD (ang. *Entity Relationship Diagram*), czyli diagram związków encji, to graficzny sposób przedstawienia powiązań pomiędzy encjami w modelu danych systemu.



Rys. 5.2. Diagram ERD [opracowanie własne]

Diagram ERD przedstawiony na rysunku 5.2. obrazuje strukturę bazy danych utworzonej na potrzebę przechowywania danych niezbędnych do funkcjonowania aplikacji.

Jedną z ważniejszych części opisywanej bazy danych jest tablica przedmiotów (item). Przechowuje ona wszystkie atrybuty przedmiotów wystawianych na sprzedaż w serwisie. Rozmiary, kategorie, kolory oraz marki znajdują się w oddzielnych tabelach – dzięki temu unikamy powielania informacji w tabeli przedmiotów. Każda z wymienionych tabel posiada ograniczenie niepowtarzalności na kolumnie przechowującej nazwę danego atrybutu, które zabezpiecza przed wprowadzeniem do niej np. dwóch takich samych rozmiarów. Wszystkie z nich posiadają powiązanie z tabelą przedmiotów typu jeden do wielu. Oznacza to, że

przedmiot może posiadać przypisany tylko jeden rekord z każdej z nich, natomiast ich pojedynczy rekord może być powiązany z wieloma rekordami tabeli przedmiotów

Bardzo ważne w aplikacji było również ograniczenie możliwości wprowadzania nowych danych przez użytkowników. Dane w tabelach wymienionych w poprzednim akapicie są istotne w systemie, ponieważ użytkownicy mają możliwość filtrowania wyświetlanych przedmiotów. Jeżeli użytkownicy mogliby wprowadzać np. własne kolory, doprowadziłoby to do powstania dezorganizacji. Kolejnym czynnikiem decydującym o ograniczeniu możliwości wyboru atrybutów przedmiotu jest fakt, że nie istnieje nieskończona liczba kolorów, czy też rozmiarów. Podczas wystawiania ogłoszenia marka jest jedyną wartością, którą użytkownik może sam wprowadzić. Obecnie na świecie istnieje wiele marek odzieżowych i nie sposób byłoby je wszystkie naraz umieścić w bazie.

Z opisanych tabel reprezentujących atrybuty przedmiotów, korzysta również relacja saved_filter. Przechowuje ona dane dotyczące filtrów wyszukiwania zapisanych przez użytkowników. Klucze obce są powiązane z nimi opcjonalnymi związkami, ponieważ filtry wyszukiwania nie muszą ich zawierać. Przykładowo, można zapisać filtr wskazujący jedynie na zakres cen przedmiotów, które mają zostać wyświetlone.

Płeć w bazie danych przechowywana jest jako enum czyli typ wyliczeniowy. Posiada on trzy wartości: kobieta, mężczyzna oraz płeć nieokreślona. Jest używany w dwóch przypadkach. Pierwszym z nich jest wybór płci przez użytkownika, a drugim wybieranie docelowej płci danego przedmiotu – w tej sytuacji wybranie wartości nieokreślonej będzie oznaczało, że przedmiot jest unisex, czyli może być używany zarówno przez osoby płci męskiej, jak i żeńskiej.

W tabeli kategorii umożliwione zostało zagnieżdżanie się rekordów. Każda kategoria może posiadać wiele podkategorii. Możliwe jest to dzięki kluczowi obcemu – kolumnie parent_id. Zawiera ona powiązanie z rekordem kategorii nadrzędnej. Jeżeli rekord nie posiada odwołania do innego, oznacza to, że reprezentuje kategorię główną. Opisana struktura pozwala na łatwe pobieranie oraz wyświetlanie kategorii po stronie aplikacji klienta.

Tabela user (użytkownik) przechowuje informacje dotyczące osób korzystających z aplikacji. Najważniejszymi informacjami, które przechowuje to m.in. adresy e-mail użytkowników, nazwy profili oraz hasła. Podczas tworzenia nowego użytkownika, tworzony jest również rekord w tabeli verification_token. Przechowuje ona token użytkownika, który może zostać wykorzystany przez użytkownika do aktywacji własnego konta.

Kolejnym ważnym modułem aplikacji, jest moduł komunikacji pomiędzy użytkownikami.

Relacja, która zachodzi pomiędzy użytkownikami a konwersacjami to relacja wiele do wielu – użytkownik może posiadać wiele czatów, natomiast w czacie może brać udział wiele osób. Aplikacja pozwala na tworzenie maksymalnie dwuosobowych konwersacji. Implementacja takiego powiązania w bazie danych jest możliwa poprzez utworzenie tablicy pomocniczej – chat_participant. Dzięki jej istnieniu, powiązanie wiele do wielu zostaje zastąpione powiązaniami typu jeden do wielu.

Pozostałe tabele istniejące w opisywanej bazie danych to tabela komentarzy oraz obserwujących. Pierwsza z nich przechowuje komentarze dodawane do profilu wraz z jego oceną w skali od jednego do pięciu. Tabela obserwujących poza identyfikatorem przechowuje dwa klucze obce, będące odwołaniami do tabeli użytkowników. Zawiera informacje o tym, którzy użytkownicy obserwują innych.

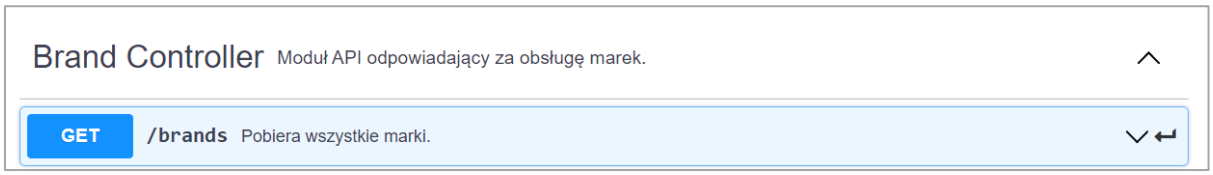
5.3. Dokumentacja API

Na podstawie wymagań funkcjonalnych utworzona została aplikacja serwerowa korzystająca z REST API (ang. *Representational State Transfer Application Programming Interface*), czyli jednego ze stylów architektonicznych. Podstawą REST API jest tworzenie tak zwanych kontrolerów, czyli klas zawierających metody, będącymi węzłami końcowymi, do których aplikacje klienckie mogą się łączyć dzięki protokołowi HTTP. REST udostępnia metody wskazujące na charakter wykonywanego zapytania. W aplikacji użyte zostały cztery z nich:

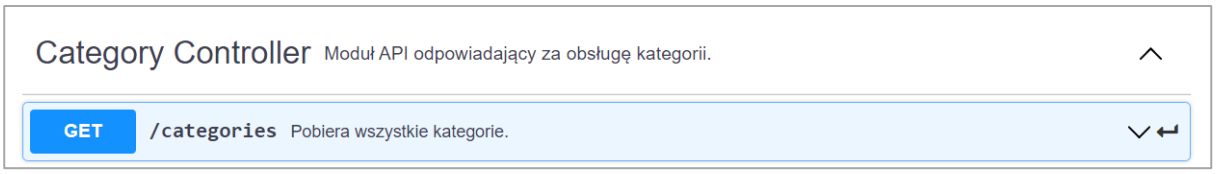
- GET – pobieranie zasobów,
- POST – tworzenie nowych zasobów,
- PUT – edycja istniejących zasobów,
- DELETE – usunięcie istniejących zasobów.

Implementując serwer omawianej aplikacji, utworzonych zostało dziesięć kontrolerów. Każdy z nich odpowiada za inne zasoby systemu.

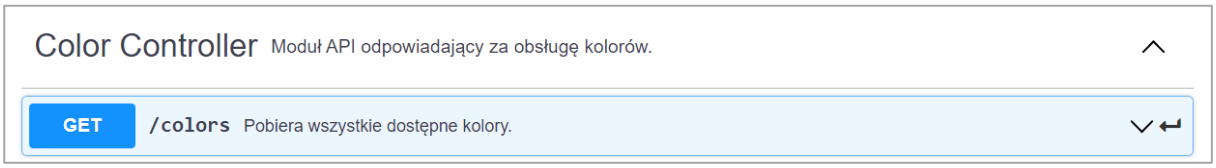
Na rysunkach zamieszczonych w tym podrozdziale zaprezentowane zostaną punkty końcowe wszystkich kontrolerów. Dokumentacja REST API została utworzona z wykorzystaniem narzędzia Swagger, opisanego w czwartym rozdziale.



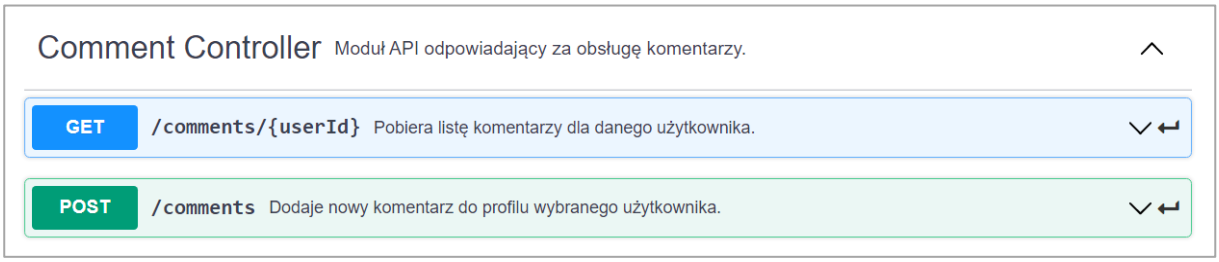
Rys. 5.3. Kontroler marek [opracowanie własne]



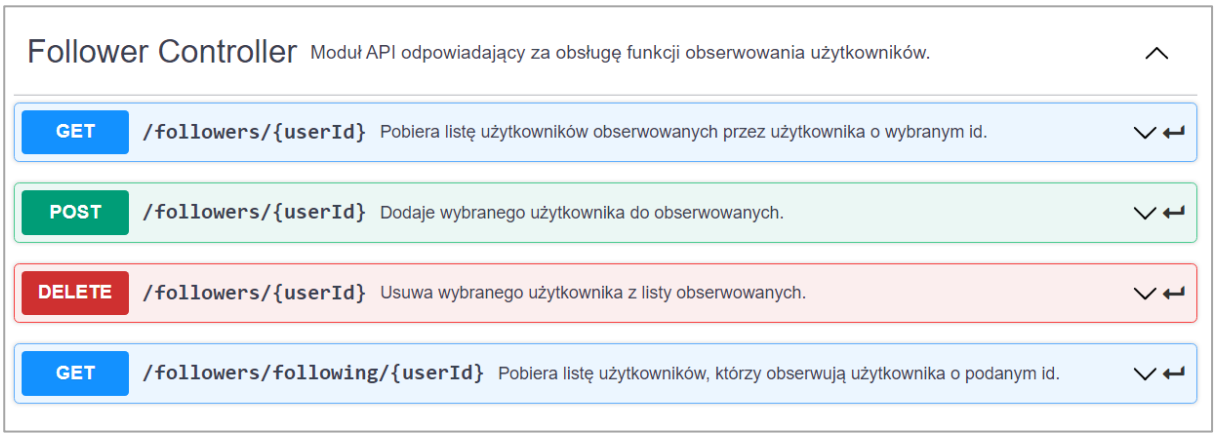
Rys. 5.4. Kontroler kategorii [opracowanie własne]



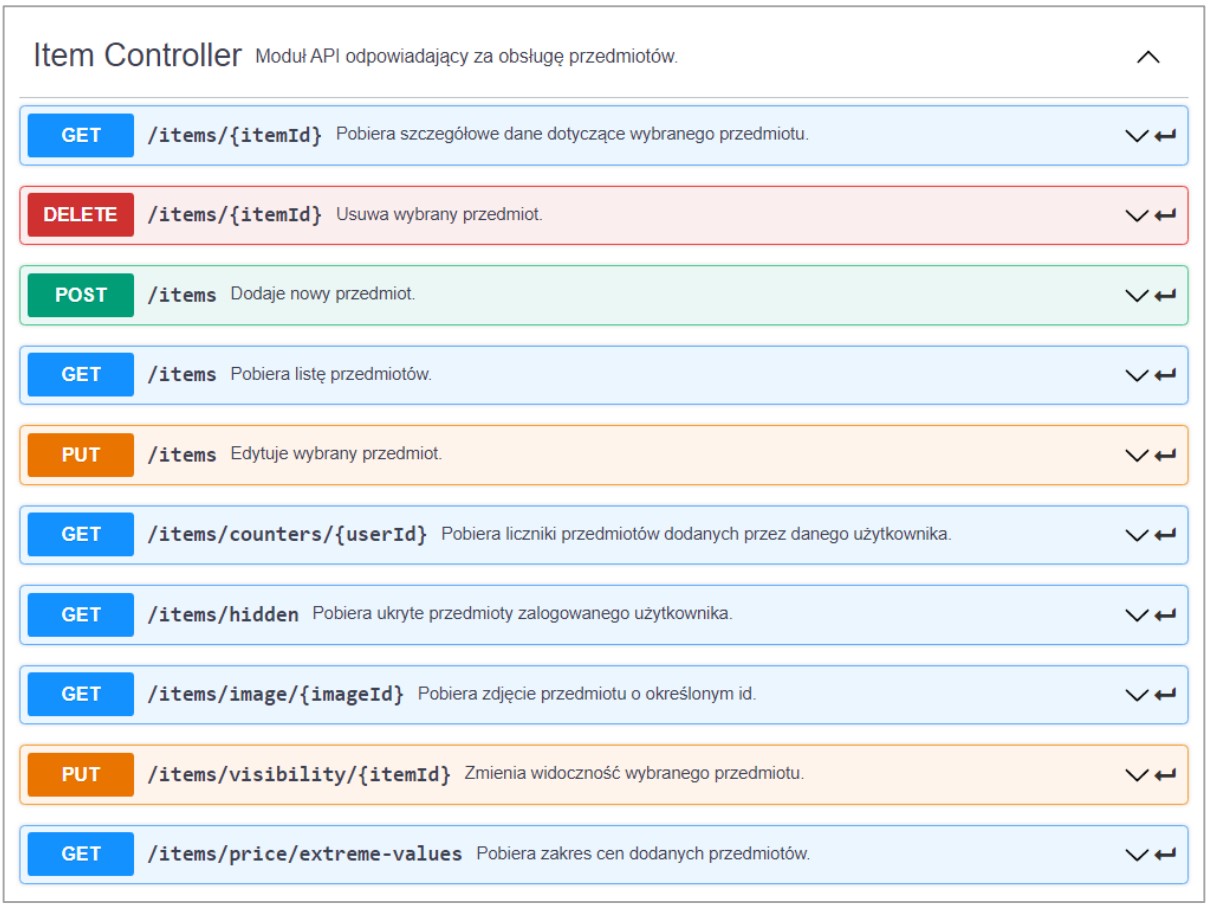
Rys. 5.5. Kontroler kolorów [opracowanie własne]



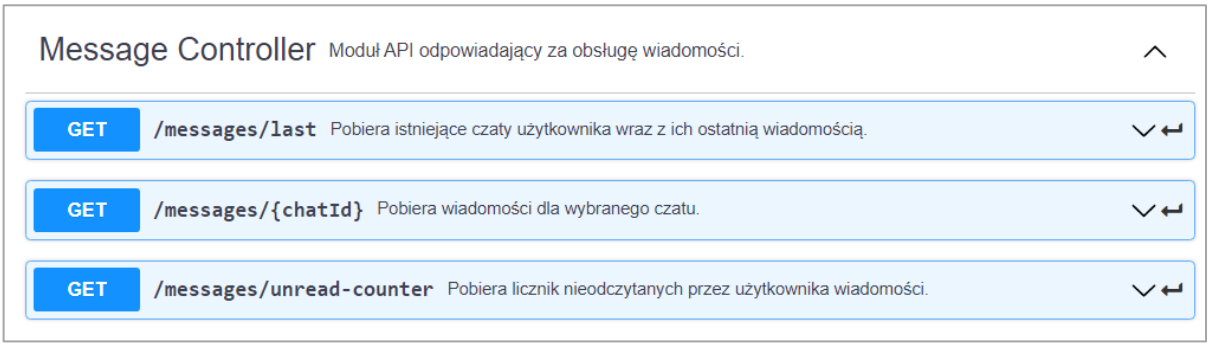
Rys. 5.6. Kontroler komentarzy [opracowanie własne]



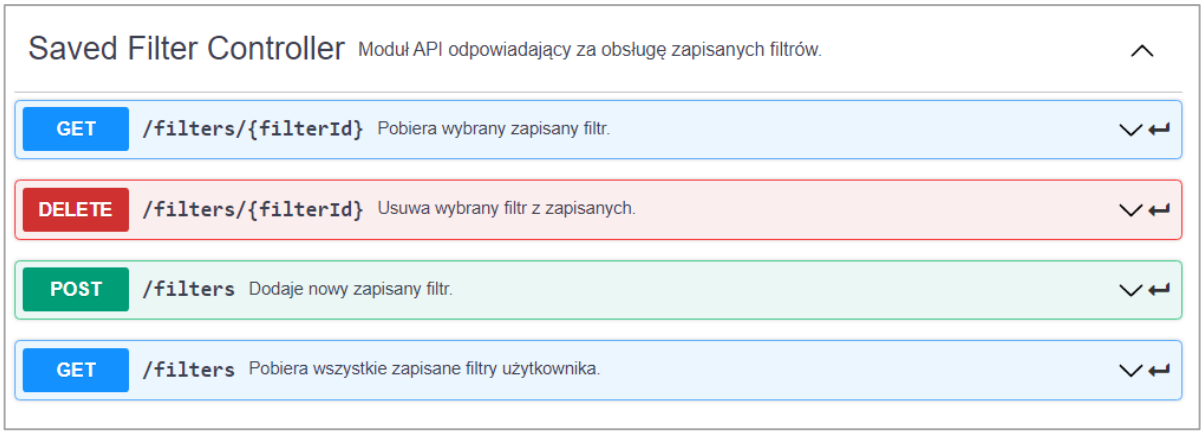
Rys. 5.7. Kontroler obserwujących [opracowanie własne]



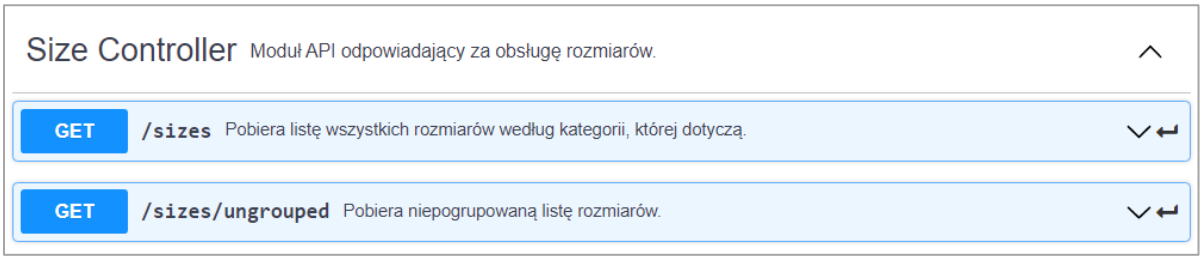
Rys. 5.8. Kontroler przedmiotów [opracowanie własne]



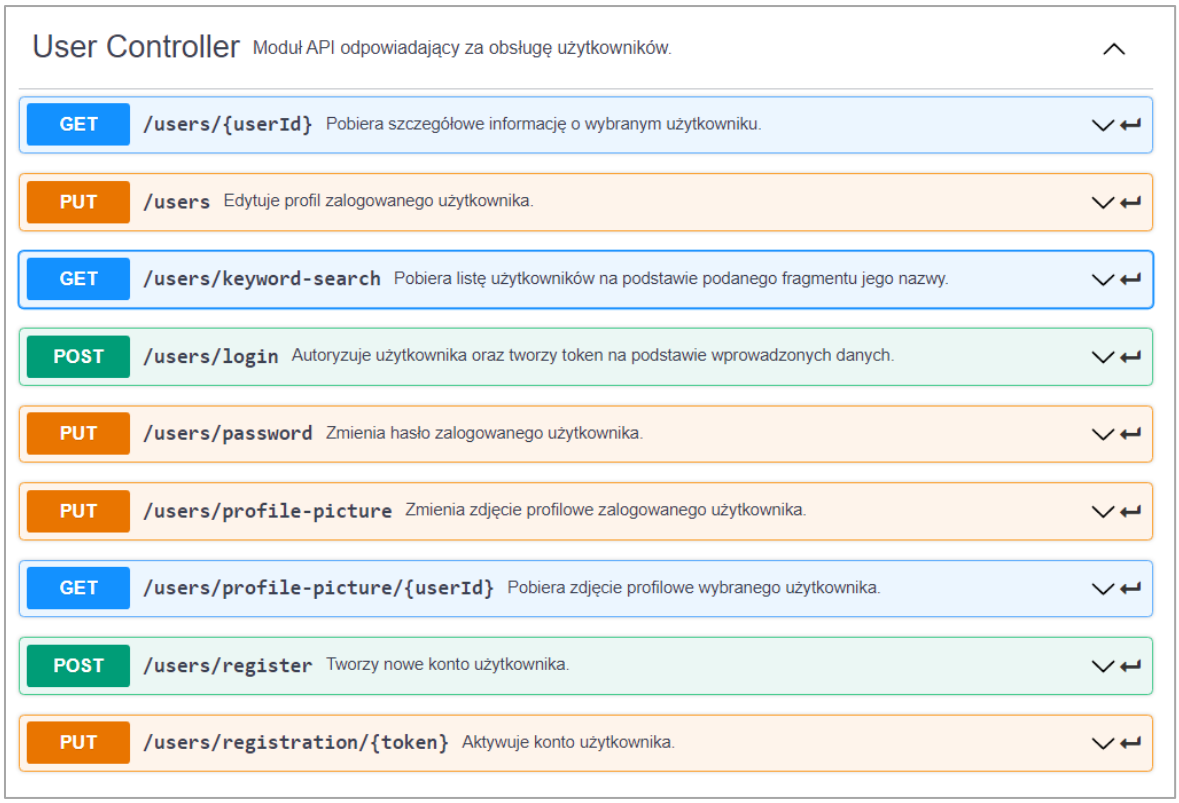
Rys. 5.9. Kontroler wiadomości [opracowanie własne]



Rys. 5.10. Kontroler zapisanych filtrów [opracowanie własne]



Rys. 5.11. Kontroler rozmiarów [opracowanie własne]



Rys. 5.12 Kontroler użytkowników [opracowanie własne]

5.4. Wybrane elementy implementacyjne

5.4.1. Bezpieczeństwo aplikacji

W celu spełnienia wymagań dotyczących bezpieczeństwa aplikacji zastosowany został standard JWT. Kiedy użytkownik loguje się do aplikacji, serwer generuje token, który zostaje przesłany do aplikacji klienta. Token przechowuje następujące dane:

- adres e-mail zalogowanego użytkownika,
- data ważności utworzonego tokena,
- rola użytkownika,
- data utworzenia tokena.

Otrzymany token zostaje zapisany w tak zwanym local storage, czyli w lokalnej pamięci przeglądarki. Przy każdym żądaniu klienta opisany token zostaje przesłany do serwera i na jego podstawie użytkownik zdobywa dostęp do określonych zasobów.

5.4.2. Przesyłanie oraz zapisywanie zdjęć

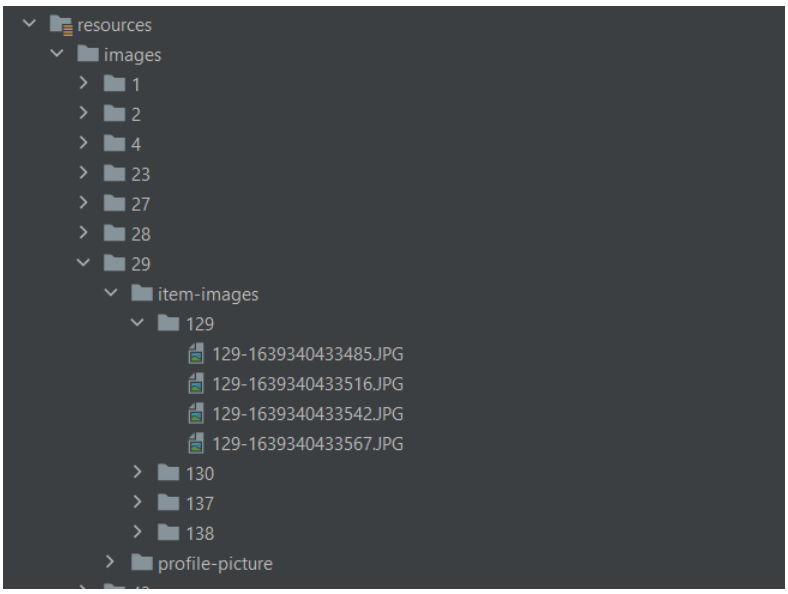
Podczas dodawania nowego przedmiotu użytkownik ma możliwość przesłania zdjęć wystawianego na sprzedaż przedmiotu. Z tego powodu żądanie HTTP odpowiadające za dodanie nowego przedmiotu, przesyła dane typu multipart/form-data. Oznacza to, że treść żądania jest przesyłana jako dane wieloczęściowe. Opisany typ pozwala na przesyłanie plików binarnych.

Każdy plik przesłany do serwera jest reprezentowany przez obiekt interfejsu MultipartFile. Dla każdego z nich wywoływana jest metoda, która zapisuje plik na serwerze. Przedstawiona została ona w listingu 5.1.

Listing 5.1. Zapisywanie zdjęć [opracowanie własne]

```
private void saveUploadedFile(MultipartFile file, Item item, boolean
isMainPicture) throws IOException {

   String MAIN_DIR = "src/main/resources/images/";
   String SUB_DIR = item.getUser().getId().toString() + "/item-images/" +
item.getId().toString() + "/";

   String FILE_LOCATION = SUB_DIR + item.getId().toString() + "-" +
LocalDateTime.now(clock).getNano() + "." +
FilenameUtils.getExtension(file.getOriginalFilename());
   Path newFile = Paths.get(MAIN_DIR + FILE_LOCATION);
   Files.createDirectories(newFile.getParent());
   Files.write(newFile, file.getBytes());
   itemPictureRepository.save(new ItemPicture(null, item, LocalDateTime.now(),
FILE_LOCATION, isMainPicture));
}
```

Metoda ta przyjmuje jako argumenty plik, obiekt reprezentujący przedmiot, do którego zdjęcie ma przynależeć, oraz typ logiczny odpowiadający za informację, czy dane zdjęcie jest jego zdjęciem głównym. Metoda dynamicznie tworzy ścieżkę zapisu zdjęcia na podstawie identyfikatorów przedmiotu oraz użytkownika, będącego jego właścicielem. Przedmioty zapisywane są po stronie serwera w strukturze przedstawionej na rysunku 5.13.



Rys. 5.13. Struktura przechowywania zdjęć [opracowanie własne]

Każdy użytkownik posiada własny folder zapisu, a w nim zamieszczone są dwa katalogi. W jednym znajdują się zdjęcia dotyczące jego przedmiotów, a w drugim aktualne zdjęcie profilowe. W folderze zdjęć przedmiotów każdy przedmiot posiada swój określony katalog, którego nazwą jest jego identyfikator. Ukazany sposób zapisu plików znacząco wpływa na optymalizację pobierania zdjęć w celu wyświetlenia ich na stronie internetowej w przeglądarce. Dzięki temu, że są zapisane bezpośrednio w strukturze serwera, a nie w bazie danych, pobierane są znacznie szybciej. W bazie danych aplikacji przechowywana jest jedynie informacja o ścieżce zapisu danego zdjęcia, na podstawie której później plik jest wyszukiwany.

5.4.3. Filtrowanie ogłoszeń

Filtrowanie ogłoszeń zostało zaimplementowane z wykorzystaniem interfejsu CriteriaBuilder. Pozwala on na dynamiczne definiowanie zapytań SQL. Treść żądania HTTP odpowiadającego za wyszukiwanie przedmiotów jest tworzona na podstawie filtrów wybieranych przez użytkownika. Takim filtrem może być np. identyfikator kategorii. Dla każdego parametru zawartego w treści żądania tworzone są tak zwane predykaty, które umieszczane są w klauzuli where zapytania SQL.

Listing 5.2. Filtrowanie przedmiotów [opracowanie własne]

```
CriteriaBuilder builder = entityManager.getCriteriaBuilder();
CriteriaQuery<Item> query = builder.createQuery(Item.class);
Root<Item> itemRoot = query.from(Item.class);
Join<Item, Category> category = itemRoot.join("category");
List<Predicate> predicates = new ArrayList<>();
if (itemListFiltersDto.getCategoryId() != null) {
   predicates.add(category.get("id").in(categoryIds));
}
{...}
Predicate[] predicatesArray = new Predicate[predicates.size()];
```

```
Predicate predicate = builder.and(predicates.toArray(predicatesArray));
query.where(predicate);
final TypedQuery<Item> finalQuery = entityManager.createQuery(query);
return finalQuery.getResultList();
```

Listing 5.2. przedstawia fragment kodu, odpowiedzialnego za tworzenie zapytania SQL z wykorzystaniem CriteriaBuilder.

W przypadku filtrowania po kategorii istotnym elementem jest zastosowanie rekurencji, czyli sposobu implementacji, który zawiera odwoływanie się określonej metody do samej siebie. Użytkownik, przeglądając przedmioty, może wyświetlić np. wszystkie akcesoria wystawione na sprzedaż. Kiedy wybierze odpowiednią opcję za pośrednictwem interfejsu użytkownika, w żądaniu przesłany zostanie identyfikator kategorii o nazwie „akcesoria", natomiast docelowo wyświetlane przedmioty mogą posiadać odwołanie do każdej podkategorii akcesoriów. W listingu 5.3. przedstawiona została metoda odpowiedzialna na wyszukiwanie identyfikatorów wszystkich podkategorii określonej kategorii.

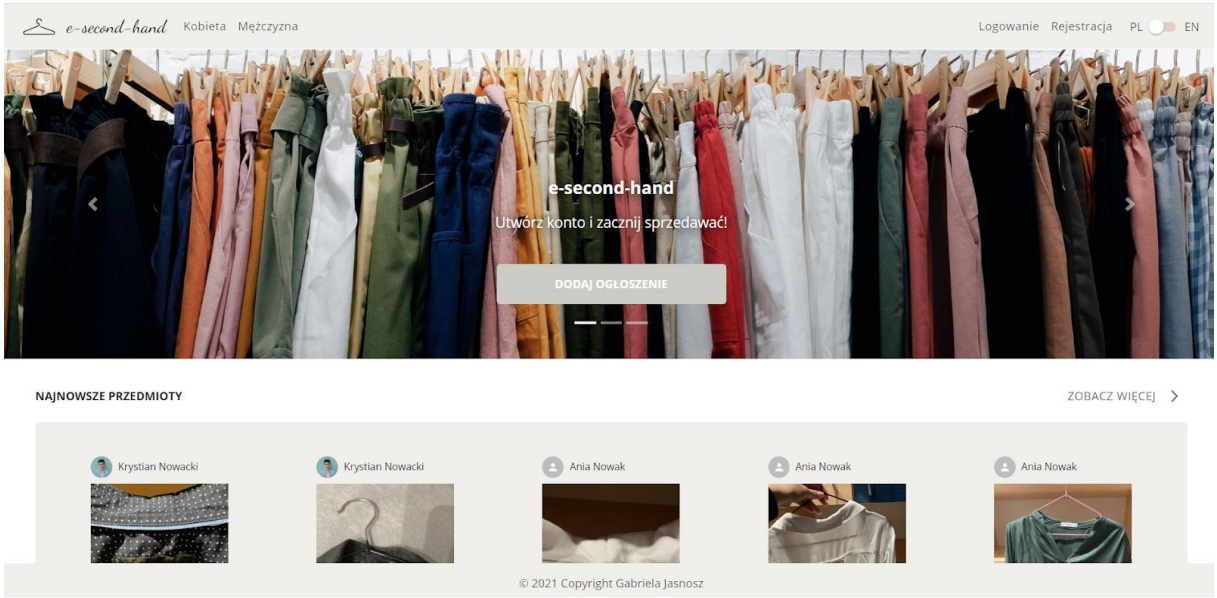
Listing 5.3. Rekurencyjne wyszukiwanie kategorii [opracowanie własne]

```
private List<Long> getCategoryIds(List<Long> resultList, Category category) {
   resultList.add(category.getId());
   if (category.getSubCategories().size() > 0) {
       for (Category subcategory : category.getSubCategories()) {
           getCategoryIds(resultList, subcategory);
       }
   }
   return resultList;
}
```

6. Interfejsy użytkownika

W tym rozdziale przedstawione zostaną najważniejsze interfejsy użytkownika aplikacji, wraz z ich opisami.

6.1. Strona główna aplikacji



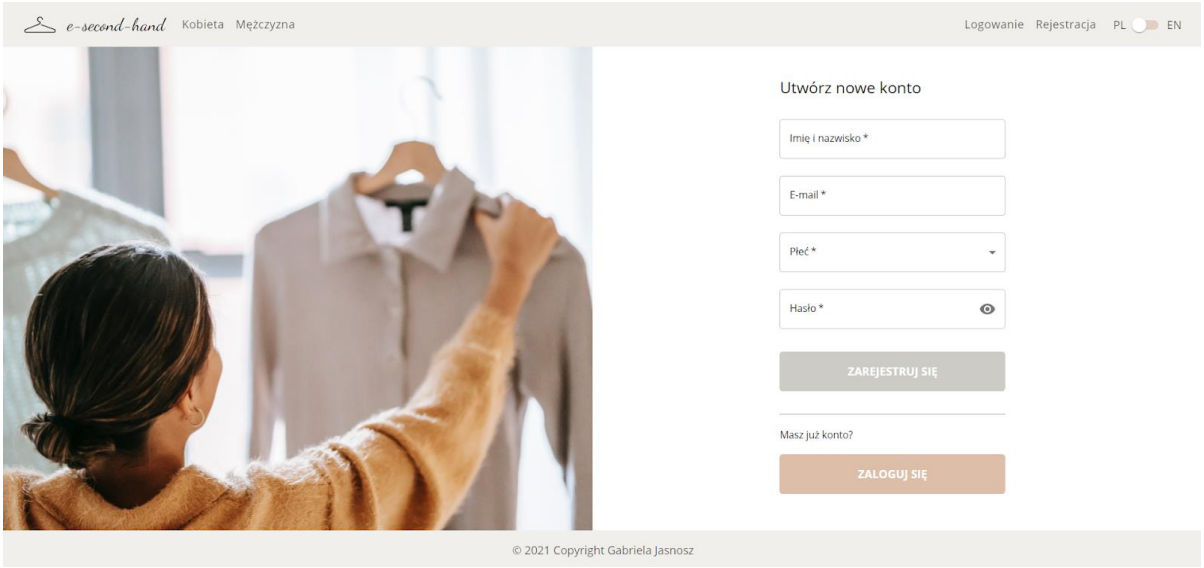
Rys. 6.1. Strona główna użytkownika niezalogowanego [opracowanie własne]

Na rysunku 6.1. przedstawiony został widok ekranu głównego dla użytkownika niezalogowanego. Funkcje, które są dostępne z poziomu tego widoku to:

- przejście na podstrony logowania oraz rejestracji,
- zmiana języka aplikacji,
- przeglądanie kategorii,
- przejście do listy przedmiotów.

Użytkownik niezalogowany widzi przycisk odpowiedzialny za dodanie nowego przedmiotu, natomiast jest on nieaktywny. Przycisk zostaje odblokowany po pomyślnym zalogowaniu na konto.

6.2. Tworzenie i aktywacja konta

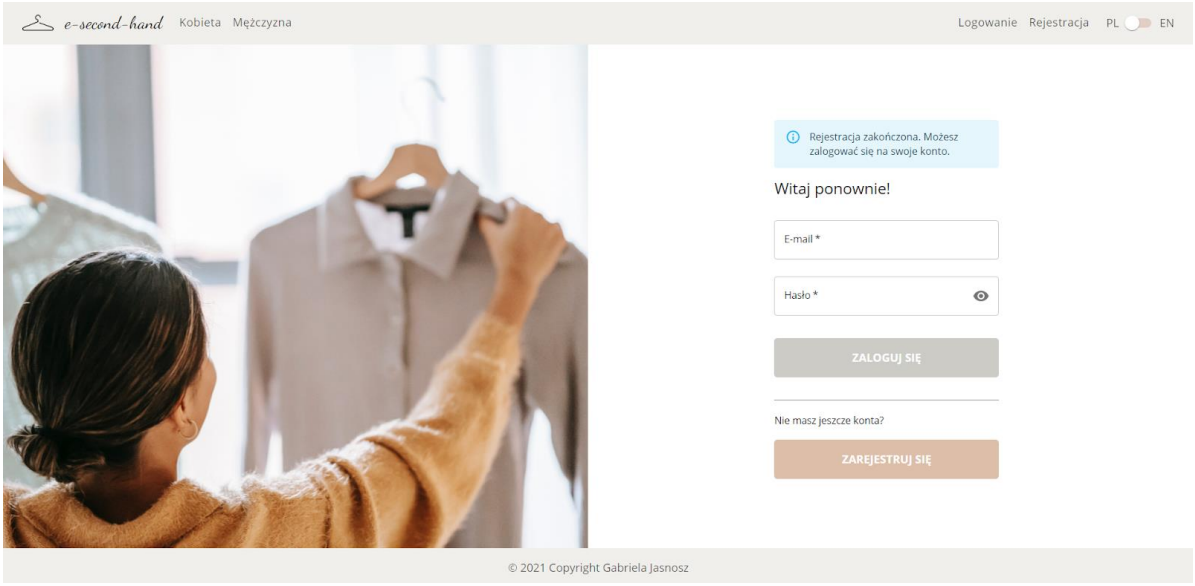


Rys. 6.2. Formularz rejestracji [opracowanie własne]

Użytkownik, tworząc nowe konto, wypełnia formularz widoczny na rysunku 6.2. Wszystkie pola, które zawiera, są wymagane do tego, aby móc go przesłać. Po poprawnym wypełnieniu i potwierdzeniu formularza dane są weryfikowane. Jeżeli wszystkie wprowadzone wartości są prawidłowe, a e-mail nie jest powiązany z żadnym z istniejących już kont, nowy profil zostaje utworzony Na podany adres e-mail zostaje przesłana wiadomość, dzięki której można go aktywować.

Rys. 6.3. Aktywacja konta [opracowanie własne]

6.3. Logowanie

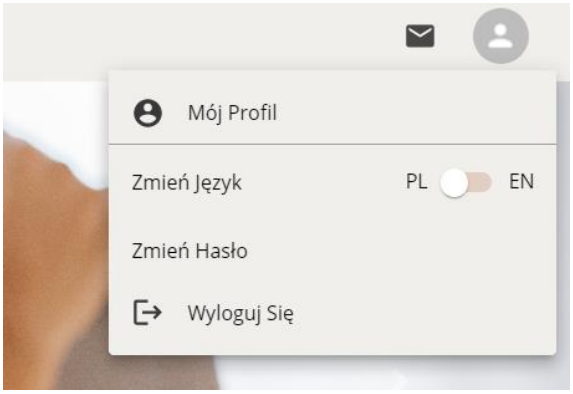


Rys. 6.4. Formularz logowania [opracowanie własne]

Po aktywacji konta, użytkownik zostaje przeniesiony na stronę logowania. Użytkownik, logując się, powinien podać swój adres e-mail oraz hasło.

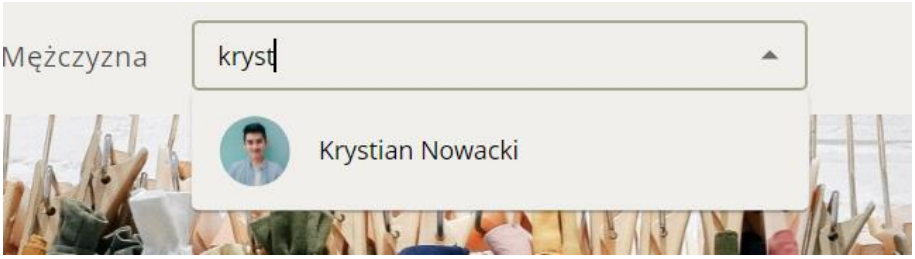
6.4. Panel użytkownika zalogowanego

Użytkownik zalogowany ma dostęp do panelu przedstawionego na rysunku 6.5. Dzięki niemu może przejść na swój profil, zmienić hasło, język oraz wylogować się.



Rys. 6.5. Panel zarządzania kontem [opracowanie własne]

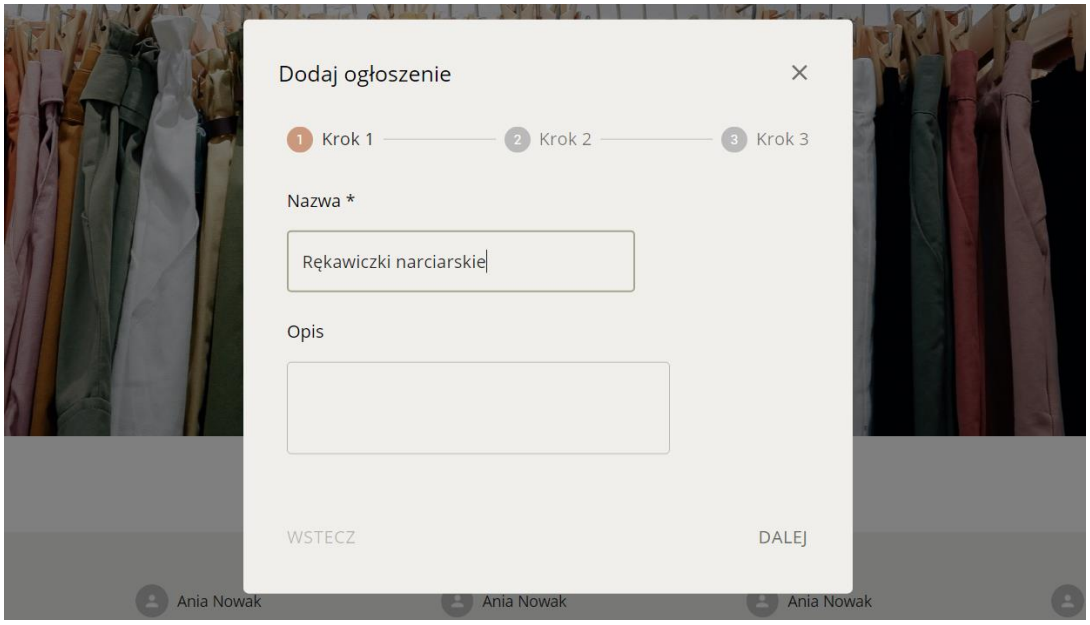
Rys. 6.6. Zmiana hasła [opracowanie własne]



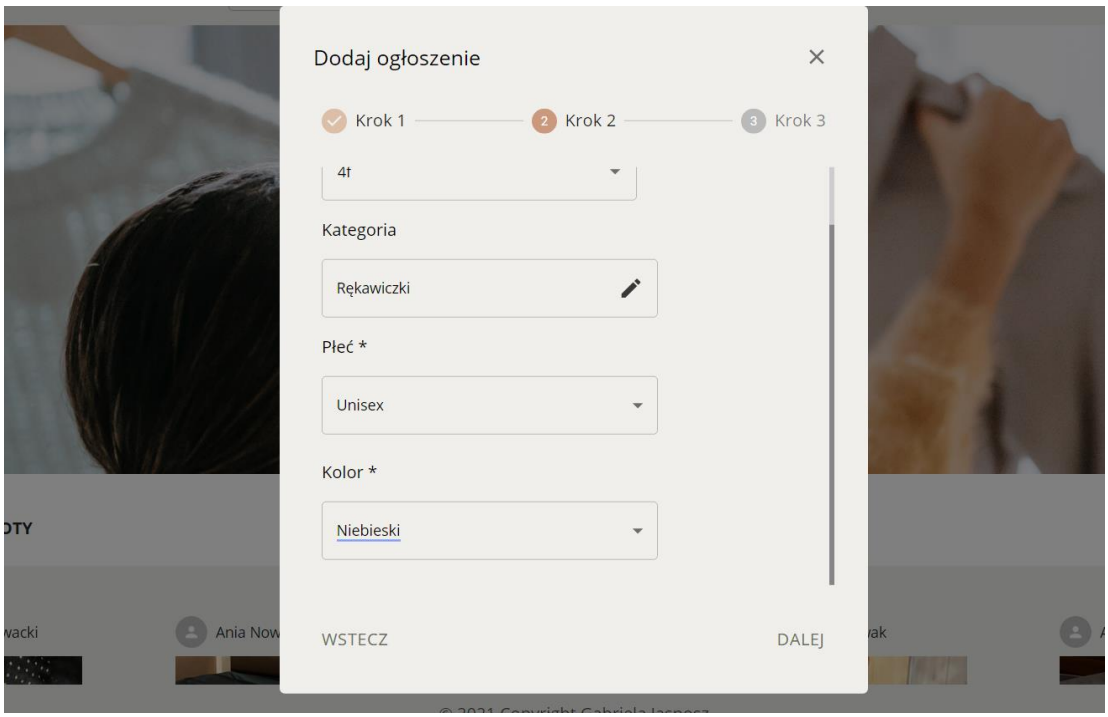
Rys. 6.7. Wyszukiwanie użytkowników [opracowanie własne]

Osoba zalogowana w nagłówku widzi pole, umożliwiające wyszukiwanie innych użytkowników. Zostało ono przedstawione na rysunku 6.7. Wybierając jeden z wyników wyszukiwania, użytkownik zostaje przeniesiony do profilu wybranej osoby.

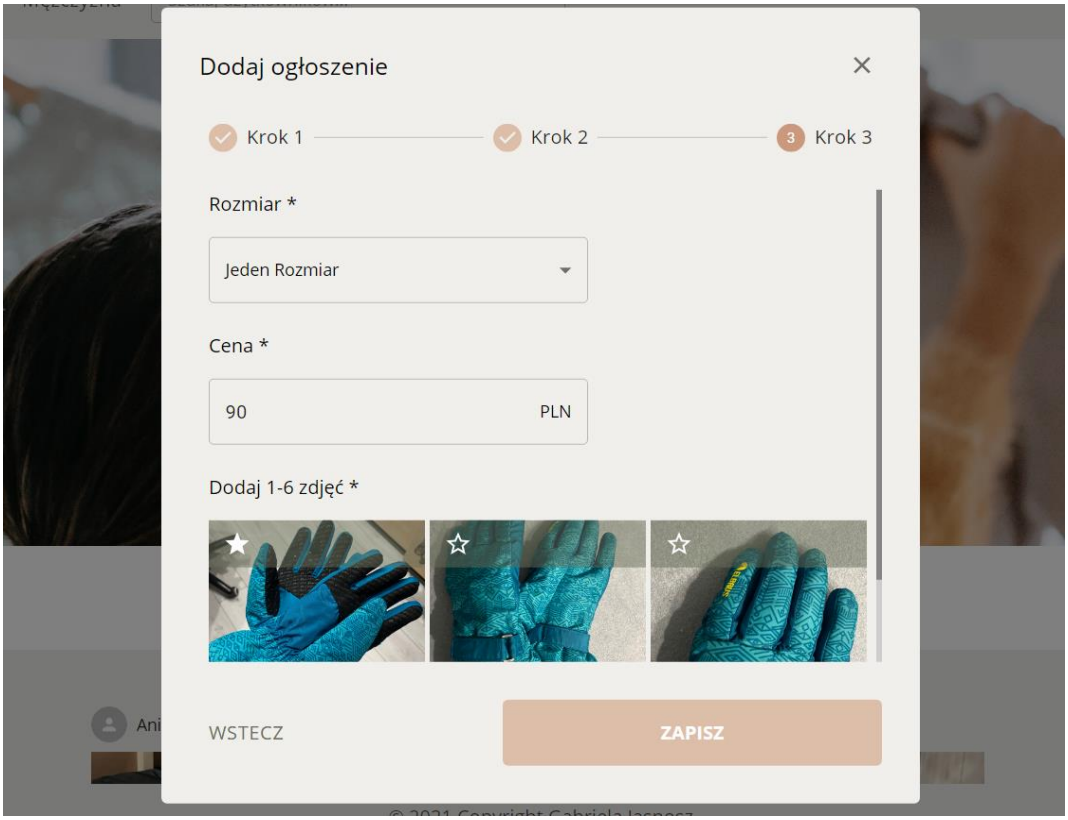
6.5. Dodawanie ogłoszenia



Rys. 6.8. Dodawanie ogłoszenia – krok pierwszy [opracowanie własne]



Rys. 6.9. Dodawanie ogłoszenia – krok drugi [opracowanie własne]

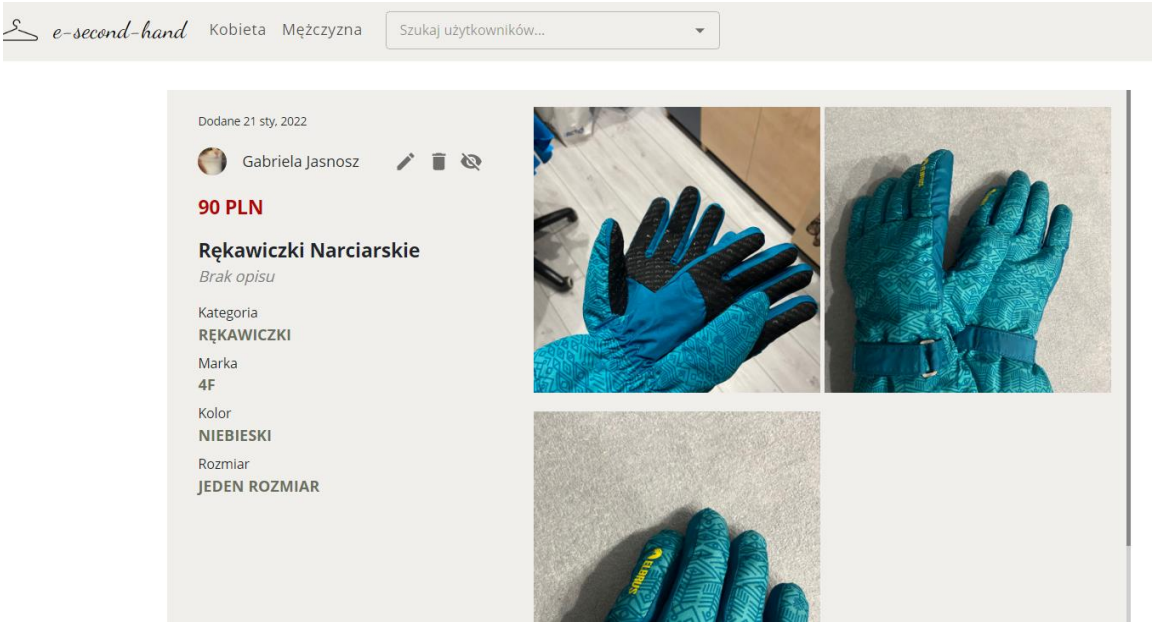


Rys. 6.10. Dodawanie ogłoszenia – krok trzeci [opracowanie własne]

Rysunki 6.8. 6.9. i 6.10. przedstawiają interfejsy użytkownika w trakcie tworzenia nowego ogłoszenia. Kolejność wystąpienia pól w formularzu nie jest przypadkowa. Użytkownik w pierwszej kolejności wybiera kategorię dodawanego przedmiotu, a następnie jest w stanie wybrać jego rozmiar, ponieważ wyświetlane rozmiary są zależne od wybranej kategorii. Jest to związane z faktem, że inaczej określamy rozmiary np. dla ubrań, a inaczej dla butów.

Domyślnie pierwsze przesłane zdjęcie jest wybierane jako zdjęcie główne ogłoszenia, natomiast użytkownik może to zmienić poprzez kliknięcie symbolu gwiazdy nad wybranym zdjęciem.

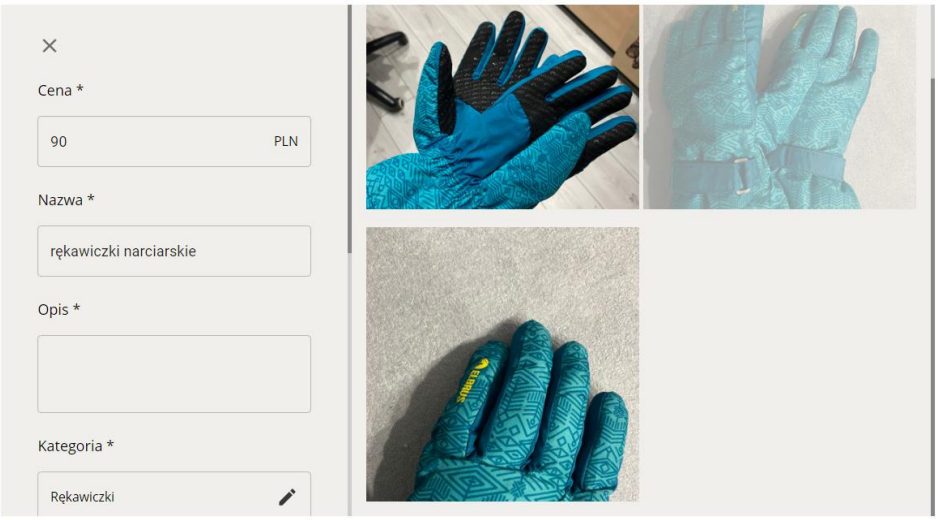
6.6. Szczegóły ogłoszenia



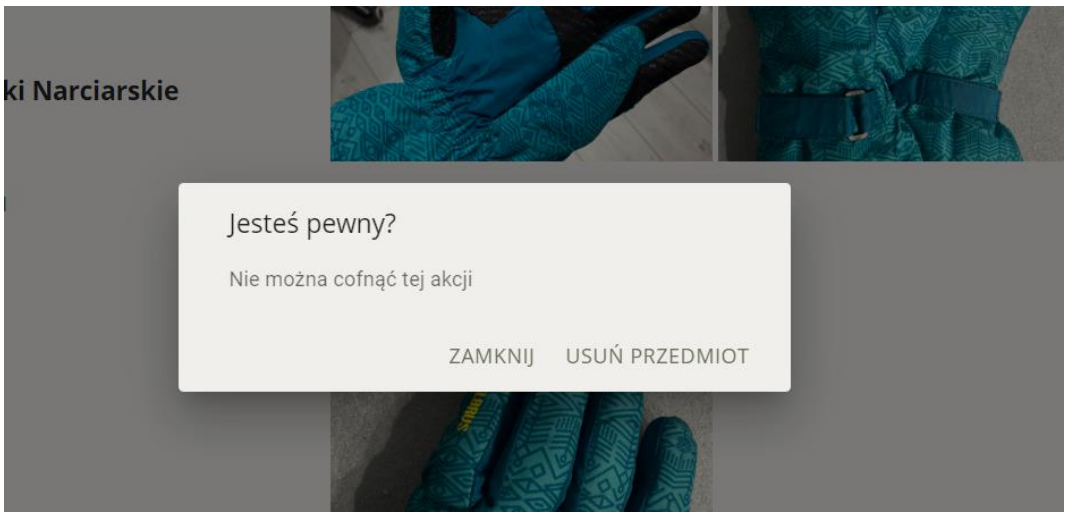
Rys. 6.11. Szczegóły ogłoszenia [opracowanie własne]

Po pomyślnym dodaniu nowego ogłoszenia, użytkownik zostaje przeniesiony na podstronę jego szczegółów. W tym miejscu użytkownik może zarządzać swoim przedmiotem – ma możliwość go ukryć, usunąć lub edytować.

Widok przedstawiony na rysunku 6.11. jest dostępny dla każdego użytkownika systemu, oczywiście poza przyciskami umożliwiającymi funkcje opisane w poprzednim akapicie.



Rys. 6.12. Edycja ogłoszenia [opracowanie własne]

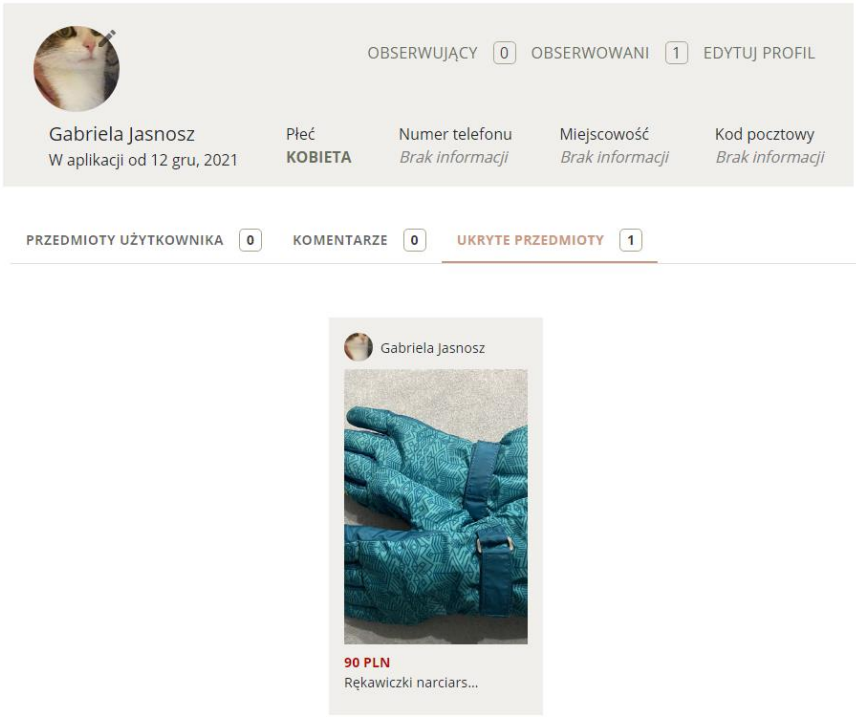


Rys. 6.13. Usunięcie ogłoszenia. [opracowanie własne]

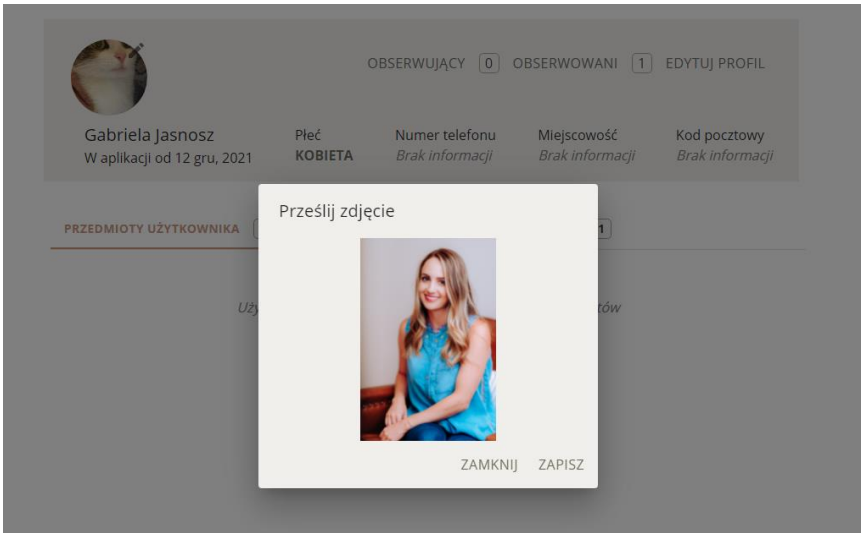
Rys. 6.14. Ukrycie ogłoszenia. [opracowanie własne]

6.7. Zarządzanie własnym profilem

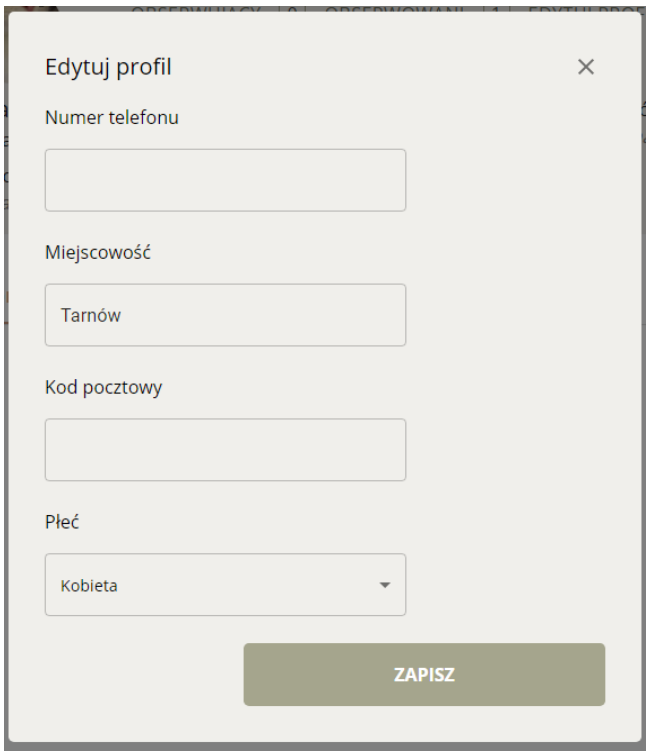
Zalogowany użytkownik, po wejściu na własny profil, może nim zarządzać. Poprzez swój profil ma dostęp do wszystkich swoich ogłoszeń, nawet tych oznaczonych jako ukryte. Rysunki zamieszczone w tym podrozdziale przedstawiają interfejsy użytkownika umożliwiające edycję jego profilu.



Rys. 6.15. Profil zalogowanego użytkownika [opracowanie własne]



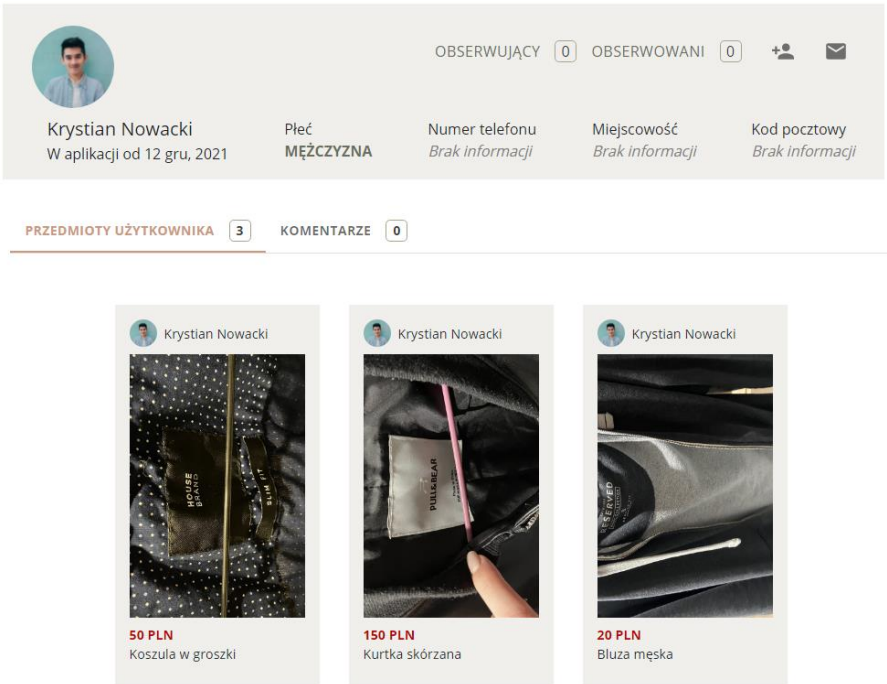
Rys. 6.16. Zmiana zdjęcia profilowego [opracowanie własne]



Rys. 6.17. Edycja danych teleadresowych [opracowanie własne]

6.8. Interakcje z innymi użytkownikami

6.8.1. Profil innego użytkownika

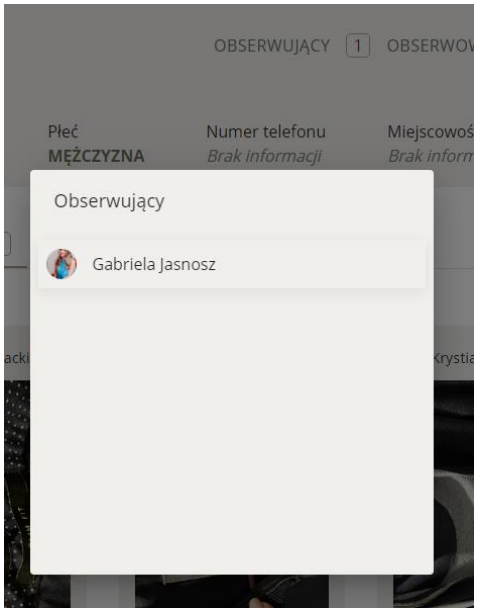


Rys. 6.18. Profil innego użytkownika [opracowanie własne]

Znajdując się na profilu innego użytkownika, możemy przeglądać jego ogłoszenia oraz komentarze dodane do jego profilu. Jesteśmy też w stanie napisać nową wiadomość poprzez wybranie przycisku z ikoną koperty. Kolejną opcją jest dodanie użytkownika do obserwowanych.

6.8.2. Lista obserwujących

Będąc na profilu innego użytkownika, jesteśmy w stanie przeglądać listę osób, które obserwuje, oraz listę osób obserwujących ten profil.

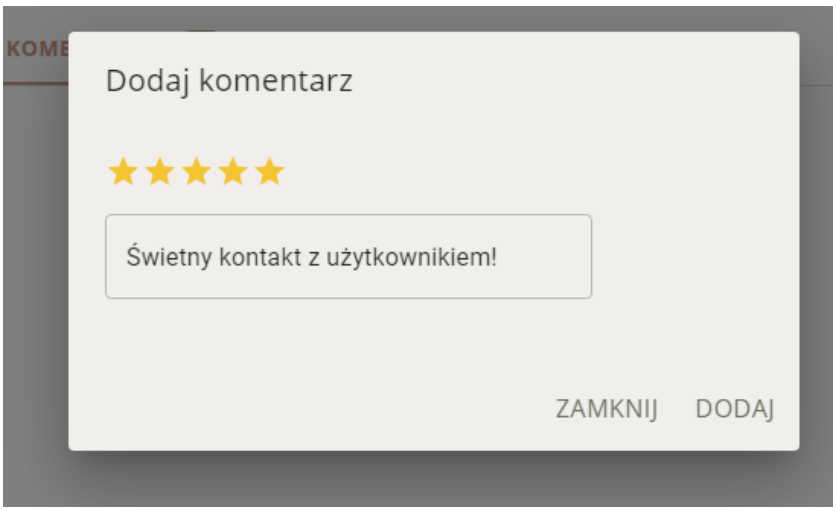


Rys. 6.19. Lista obserwujących [opracowanie własne]

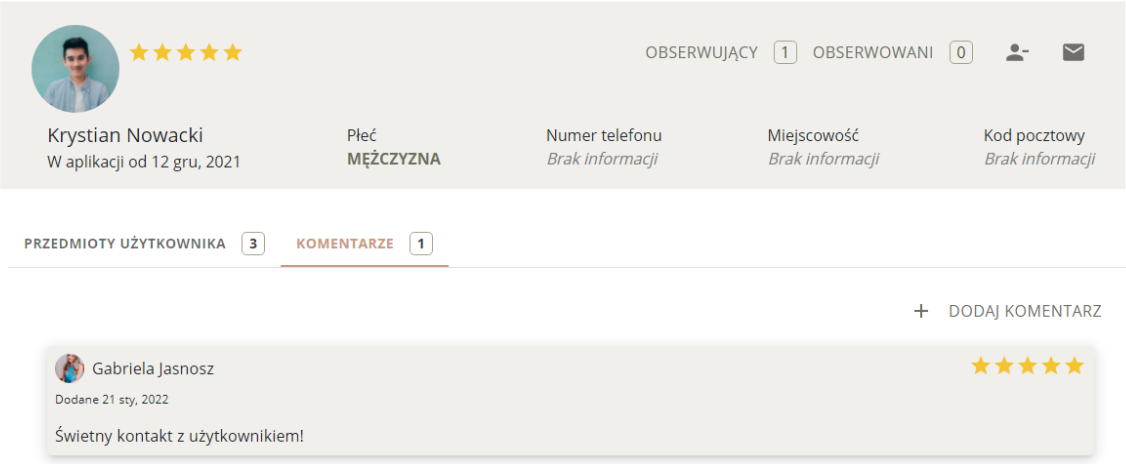
6.8.3. Zakładka komentarzy

Znajdując się na profilu innej osoby, możemy również przeglądać komentarze dodane do jej profilu oraz dodawać nowe. Dodając nowy komentarz, musimy także wystawić użytkownikowi ocenę w skali od jednego do pięciu. Jeżeli użytkownik posiada dodane opinie, obok jego zdjęcia profilowego pojawią się ikony gwiazd obrazujące średnią dodanych mu ocen.

Rys. 6.20. Zakładka komentarzy [opracowanie własne]



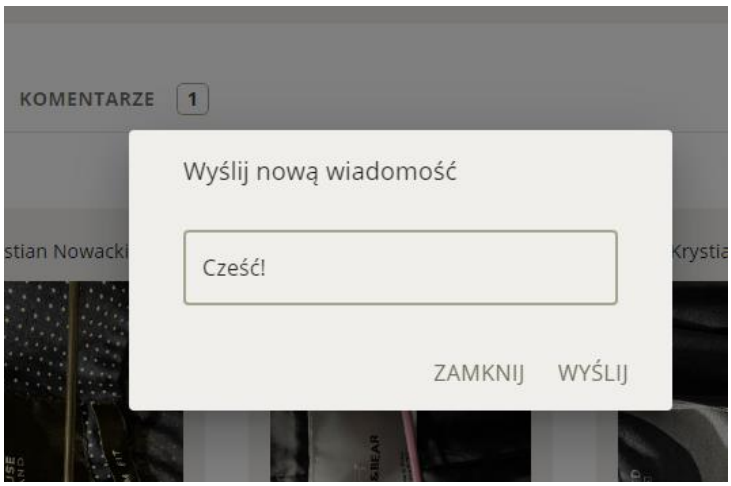
Rys. 6.21. Dodanie nowego komentarza [opracowanie własne]



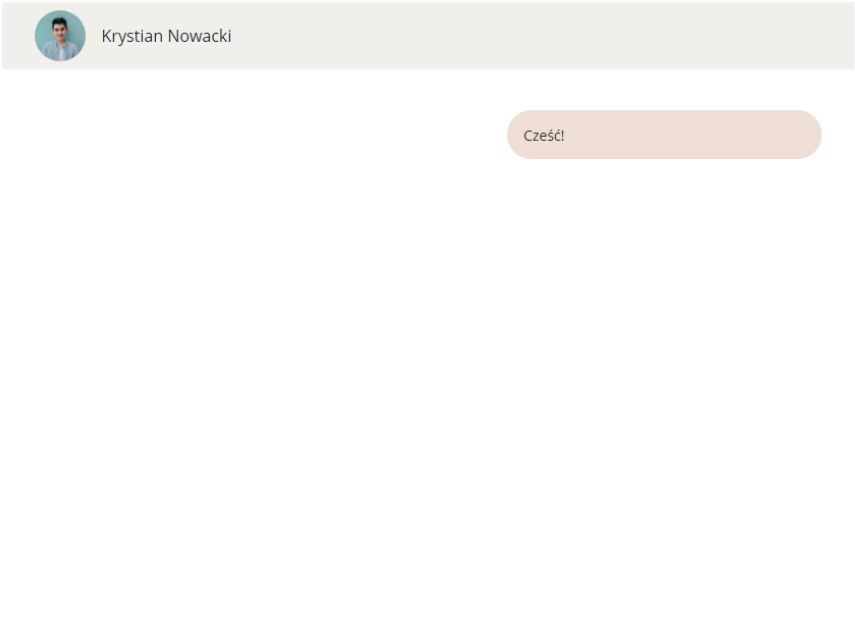
Rys. 6.22. Profil użytkownika z komentarzem [opracowanie własne]

6.8.4. Wysyłanie wiadomości

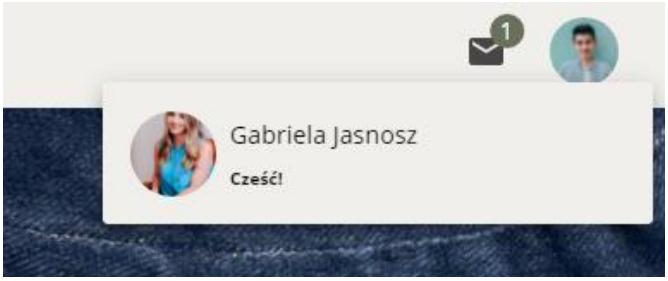
Z poziomu profilu użytkownika jesteśmy w stanie wysłać do niego wiadomość, klikając w ikonę koperty. Jeżeli konwersacja nie istnieje jeszcze w systemie, otwarty zostaje widok umożliwiający przesłanie nowej wiadomości. Widok ten został zaprezentowany na rysunku 6.23. Po wysłaniu wiadomości, użytkownik zostaje przeniesiony na stronę nowego czatu, widocznego na rysunku 6.24. Na rysunku 6.25. zaprezentowany został wygląd ekranu z poziomu użytkownika, który otrzymał nową wiadomość. Nad ikoną koperty w nagłówku strony ukazany zostaje licznik pokazujący liczbę nieodczytanych wiadomości. Klikając w ikonę, otwiera się lista konwersacji, w których użytkownik bierze udział.



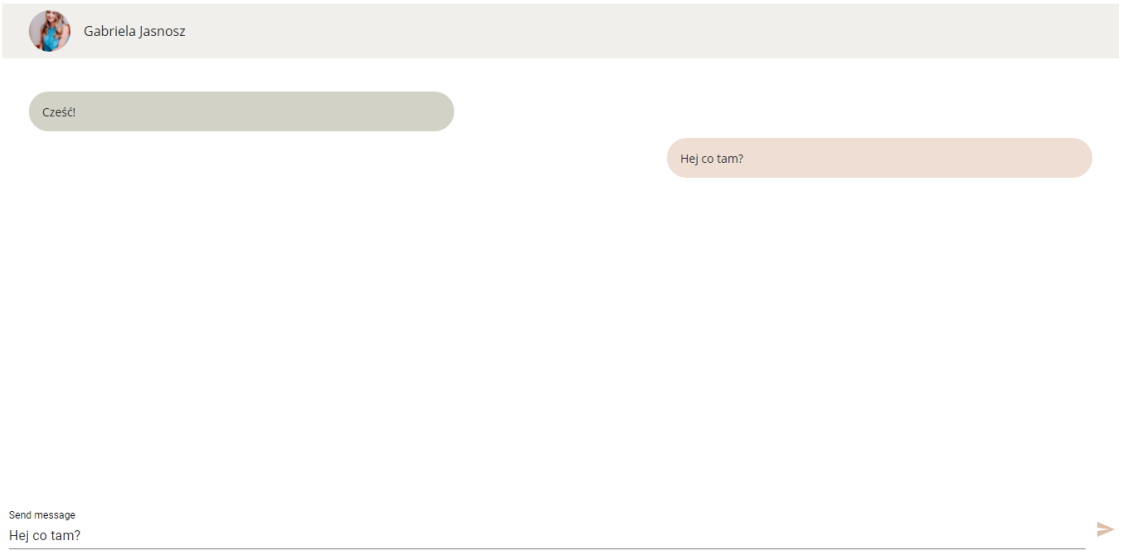
Rys. 6.23. Wysyłanie nowej wiadomości [opracowanie własne]



Rys. 6.24. Nowy czat [opracowanie własne]



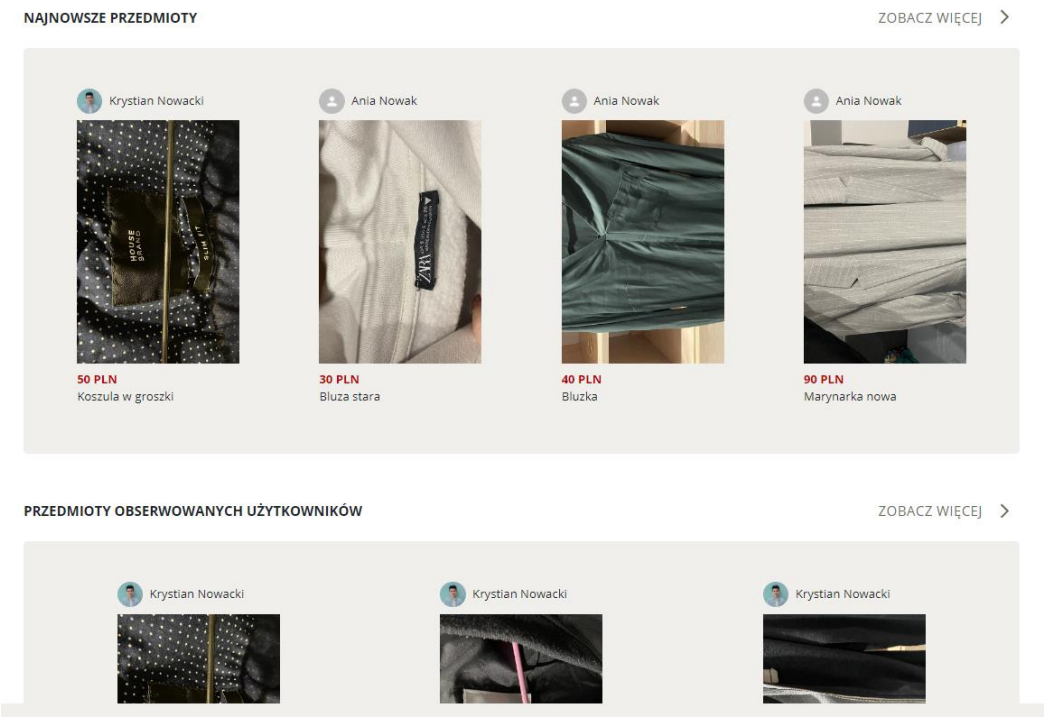
Rys. 6.25. Nowe powiadomienie [opracowanie własne]



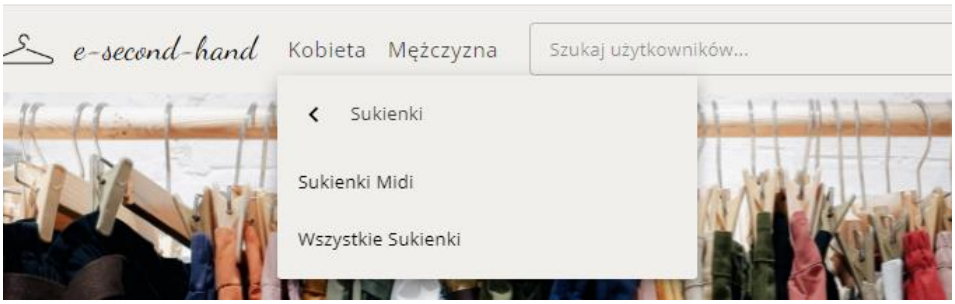
Rys. 6.26. Widok drugiego uczestnika czatu [opracowanie własne]

6.9. Przeglądanie ogłoszeń

Na podstronę zawierającą listę dodanych ogłoszeń, użytkownik jest w stanie wejść z dwóch lokalizacji – wybierając opcję “zobacz więcej”, znajdującą się w jednym z widżetów (rysunek 6.27.) lub wybierając jedną z kategorii (rysunek 6.28.).



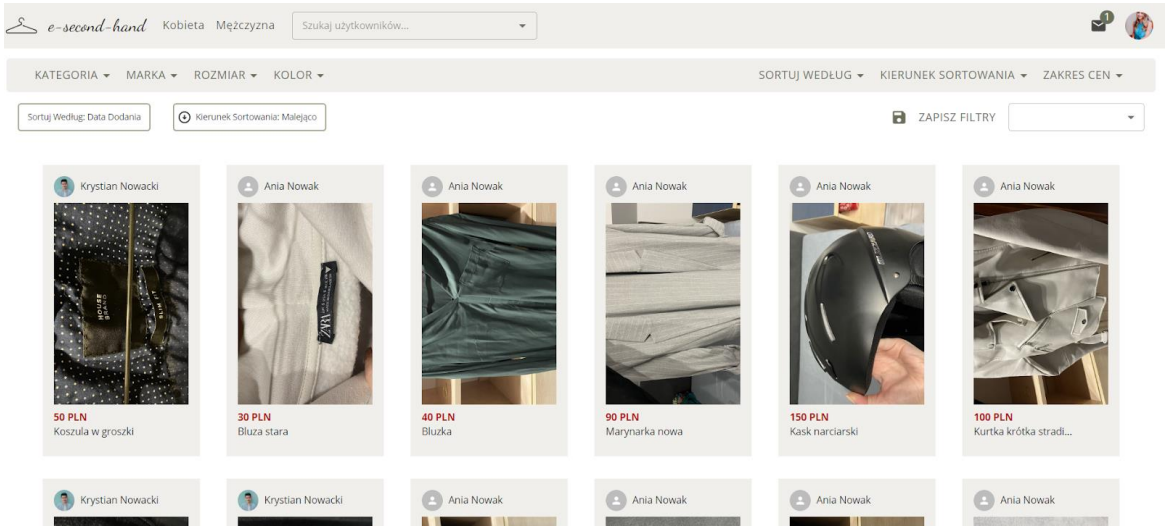
Rys. 6.27. Widżety na ekranie głównym [opracowanie własne]



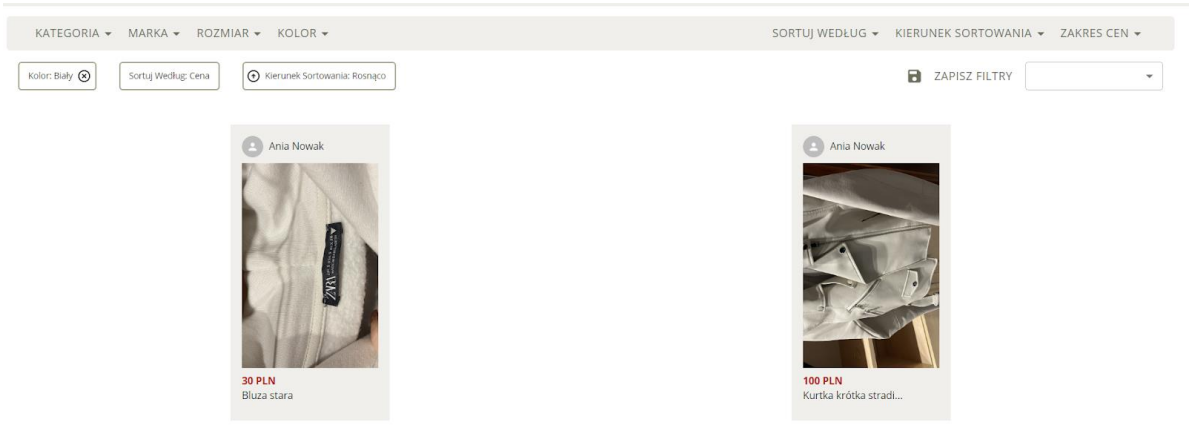
Rys. 6.28. Przeglądanie kategorii [opracowanie własne]

Rysunek 6.29. przedstawia widok zawierający główną listę przedmiotów. Aby zapewnić optymalizację działania tego widoku, lista jest paginowana – podczas przewijania odbywa się automatyczne wczytywanie pozostałych ogłoszeń.

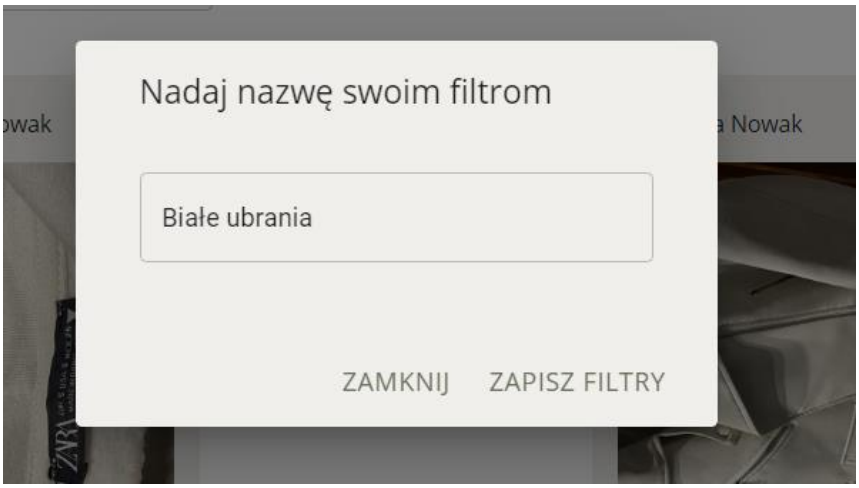
W górnej części strony, przedstawionej na rysunku 6.29., znajdują się filtry oraz opcje sortowania, które użytkownik dowolnie może zmieniać. Użytkownicy zalogowani mają również możliwość zapisywania wybranych filtrów. Po kliknięciu opcji „Zapisz filtry", ukazane zostaje okno, które pozwala na dodanie nazwy wybranym filtrom. Zostało ono przedstawione na rysunku 6.31. Po zapisaniu filtrów użytkownik może zobaczyć je na liście, którą przedstawia rysunek 6.32. Wybierając jedną z opcji listy, filtry zostają załadowane, a lista przedmiotów zostaje według nich filtrowana.



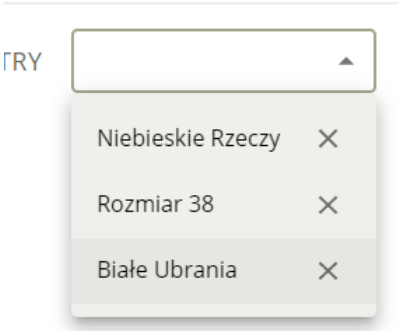
Rys. 6.29. Lista przedmiotów bez filtrów wyszukiwania [opracowanie własne]



Rys. 6.30. Lista przedmiotów z wybranymi filtrami wyszukiwania [opracowanie własne]



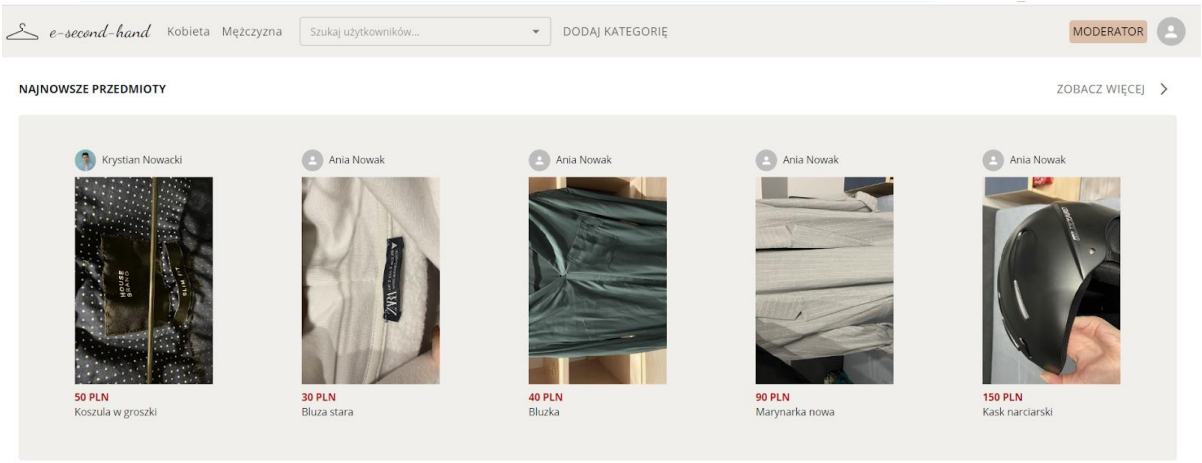
Rys. 6.31. Zapisywanie aktualnych filtrów [opracowanie własne]



Rys. 6.32. Wybór zapisanego filtru.[opracowanie własne]

6.10. Panel moderatora

Kiedy do aplikacji loguje się osoba z nadaną rolą moderatora, zostaje jej udostępniony widok przedstawiony na rysunku 6.33. Moderator to osoba, która posiada dwie główne funkcje – dodawanie nowych kategorii oraz zarządzanie ogłoszeniami.



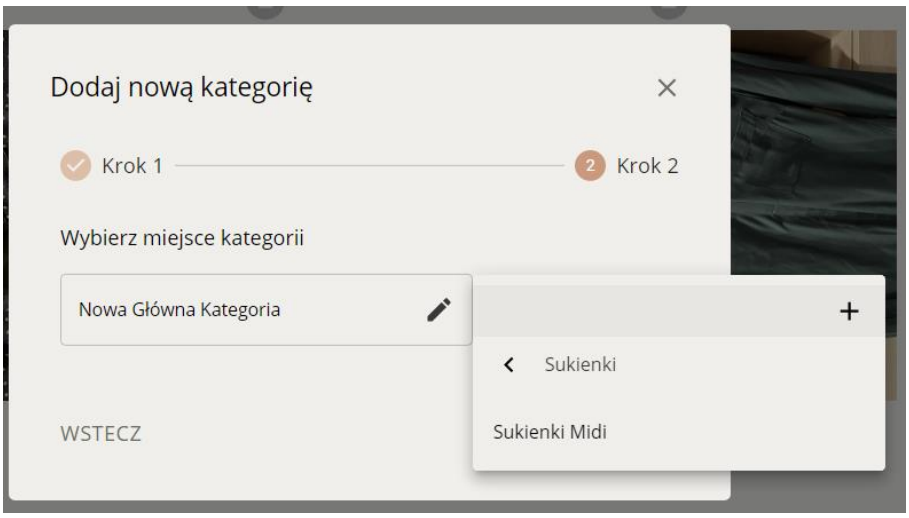
Rys. 6.33. Widok strony głównej moderatora [opracowanie własne]

6.10.1. Dodawanie kategorii

Dodając kategorię, moderator musi wprowadzić jej nazwę oraz wybrać jej umiejscowienie w strukturze już istniejących. Może ją dodać jako nową główną kategorię, ale też ma możliwość dodania jej jako podkategorię innej.

Rysunki 6.34. i 6.35. przedstawiają okna zawierające formularz dodania nowej kategorii.

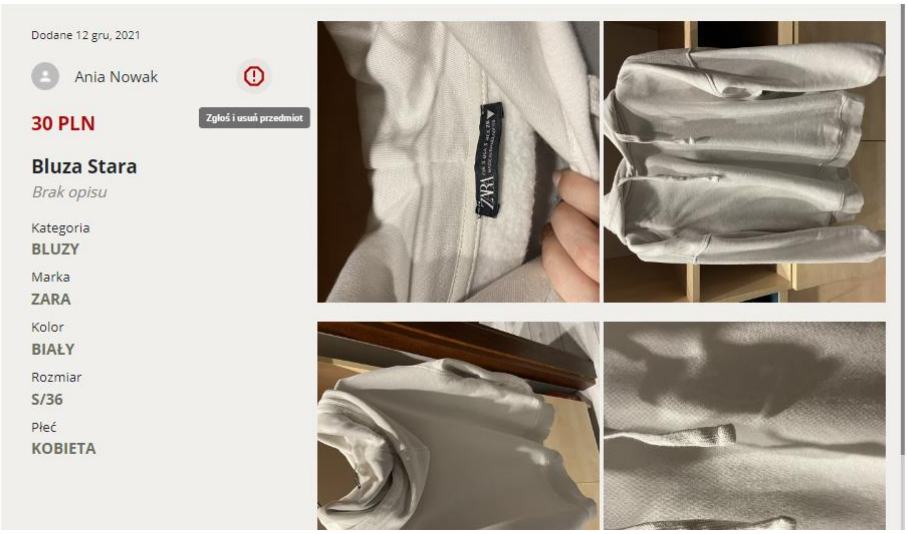
Rys. 6.34. Dodawanie nowej kategorii - krok pierwszy. [opracowanie własne]



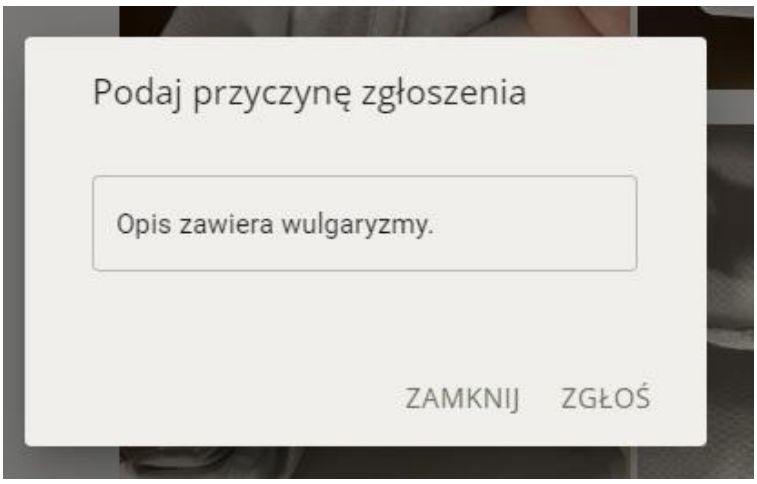
Rys. 6.35. Dodawanie nowej kategorii - krok drugi. [opracowanie własne]

6.10.2. Zgłoszenie przedmiotu

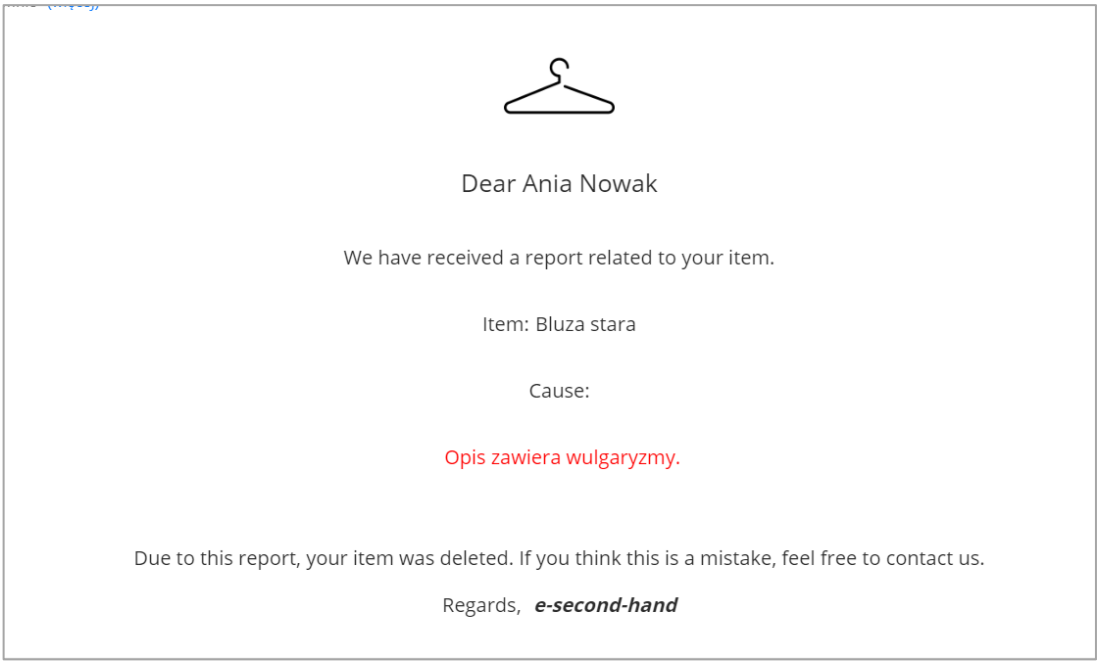
Jeżeli określone ogłoszenie nie spełnia warunków korzystania z aplikacji, moderator ma prawo do jego usunięcia. Wybierając opcję “zgłoś i usuń przedmiot” otwierane jest okno przedstawione na rysunku 6.37., które umożliwia dodanie informacji o powodzie zgłoszenia przedmiotu. Po potwierdzeniu akcji przedmiot zostaje usunięty, a do właściciela usuwanego przedmiotu zostaje przesłana informacja o zaistniałej sytuacji. Wygląd wiadomości e-mail został przedstawiony na rysunku 6.38.



Rys. 6.36. Zgłoszenie przedmiotu [opracowanie własne]



Rys. 6.37. Dodawanie przyczyny zgłoszenia [opracowanie własne]



Rys. 6.38. Widok maila z raportem [opracowanie własne]

7. Testy

W ramach przedstawionej pracy dyplomowej najważniejsze funkcje systemu zostały przetestowane za pomocą testów jednostkowych oraz integracyjnych. Utworzone zostały również testy API, które można umiejscowić pomiędzy testami jednostkowymi a integracyjnymi. Wybrane z nich zostaną zaprezentowane w tym rozdziale.

7.1. Testy jednostkowe

Test jednostkowy to test, który sprawdza poprawność działania odrębnej jednostki, którą najczęściej jest jedna metoda [8]. Testy przedstawione w niniejszym podrozdziale zostały napisane z wykorzystaniem frameworka JUnit. Wykorzystana została również biblioteka Mockito, która umożliwia tworzenie atrap wykorzystywanych w testach obiektów. Dzięki temu jesteśmy w stanie testować działanie pojedynczej metody, bez konieczności właściwego wywołania metod klas używanych w testowanej metodzie.

7.1.1. Test walidacji adresu e-mail

Użytkownik, tworząc nowe konto, musi podać swój adres e-mail. Aby nie dopuścić do możliwości podania przez użytkownika błędnego maila, aplikacja weryfikuje wprowadzoną wartość. Kod przedstawiony w listingu 7.1. przedstawia test sprawdzający poprawność tej walidacji.

Listing 7.1. Test walidacji adresu e-mail [opracowanie własne]

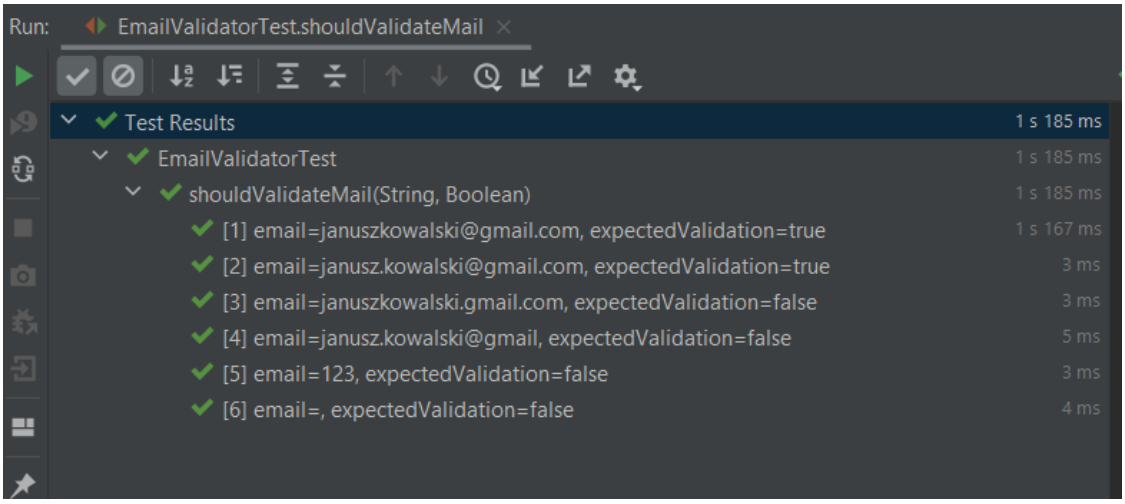
```
@InjectMocks
private EmailValidator emailValidator;

@BeforeEach
public void setUp() {
   MockitoAnnotations.initMocks(this);
}

@ParameterizedTest
@MethodSource("validatorParameters")
public void shouldValidateEmail(String email, Boolean expectedValidation) {
   Boolean result = emailValidator.isValid(email,
any(ConstraintValidatorContext.class));
   assertEquals(result, expectedValidation);
}

private static Stream<Arguments> validatorParameters() {
   return Stream.of(
           Arguments.of("januszkowalski@gmail.com", true),
           Arguments.of("janusz.kowalski@gmail.com", true),
           Arguments.of("januszkowalski.gmail.com", false),
           Arguments.of("janusz.kowalski@gmail", false),
           Arguments.of("123", false),
           Arguments.of("", false)
```

```
    );
}
```

Początkowym etapem jest utworzenie obiektu klasy, której metoda będzie testowana. Obiekt zostaje zainicjalizowany podczas wywołania metody setUp, która dzięki odpowiedniej adnotacji wykonuje się przed każdym testem.

Utworzony test jest testem parametryzowanym. Oznacza to, że zostaje wykonany wielokrotnie, dla każdego elementu podanego mu źródła. W tym przypadku, test zostaje wykonany dla każdego elementu strumienia zwracanego przez metodę validatorParameters. Każdy element zawiera wartość, którą test powinien sprawdzić oraz oczekiwaną wartość zwróconą przez testowaną metodę.

Dzięki wywołaniu metody assertEquals z pakietu Mockito, jesteśmy w stanie zweryfikować czy zwrócona wartość pokrywa się z oczekiwaną. Rysunek 6.39. przedstawia wynik uruchomienia opisanego testu.



Rys. 6.39. Wynik uruchomienia testu walidacji maila [opracowanie własne]

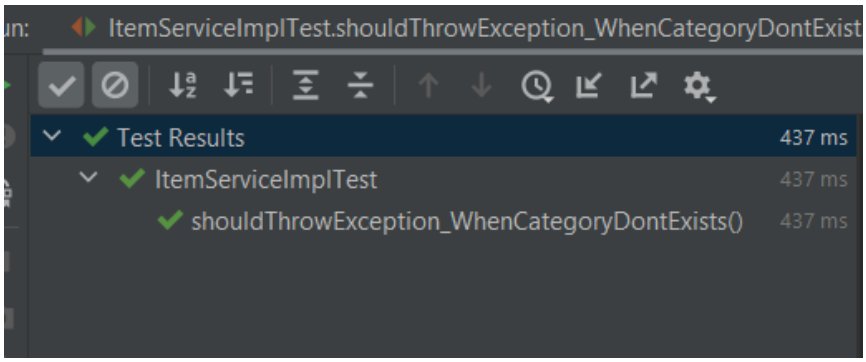
7.1.2. Test weryfikacji atrybutów dodawanego przedmiotu

Test przedstawiony w tym podrozdziale weryfikuje poprawność metody, która odpowiada za zapisywanie nowego przedmiotu w bazie danych. Metoda weryfikuje, czy podane atrybuty są poprawne. Jeżeli został podany identyfikator kategorii, który nie istnieje, metoda powinna zwrócić wyjątek i nie zapisać przedmiotu w bazie.

Listing 7.2. Test weryfikacji atrybutów dodawanego przedmiotu [opracowanie własne]

```
@Test
public void shouldThrowException_WhenCategoryDontExists() {
   ItemEntryDto itemEntryDto = createCorrectEntryDto();
   when(categoryRepository.findById(anyLong())).thenReturn(Optional.empty());
   String expectedMessage = "New item properties are invalid";

   Exception exception = assertThrows(InvalidItemPropertiesException.class, () -
> {
       itemService.saveItem(itemEntryDto);
   });

   assertTrue(exception.getMessage().contains(expectedMessage));
}
```

Ponieważ jest to test jednostkowy, wywołanie metody repozytorium kategorii musi zostać "zamockowane", czyli zasymulowane. Przyjmujemy w teście, że jeżeli wywołana zostanie metoda repozytorium kategorii odpowiedzialna za wyszukiwanie kategorii po jej identyfikatorze, ma ona nic nie zwrócić. Następnie testujemy, czy wywołanie metody zapisującej przedmiot zwróci wyjątek InvalidItemPropertiesException. Na rysunku 6.40. zaprezentowany został wynik uruchomienia opisanego testu.



Rys. 6.40. Wynik uruchomienia testu weryfikacji atrybutów dodawanego przedmiotu [opracowanie własne]

7.2. Testy integracyjne

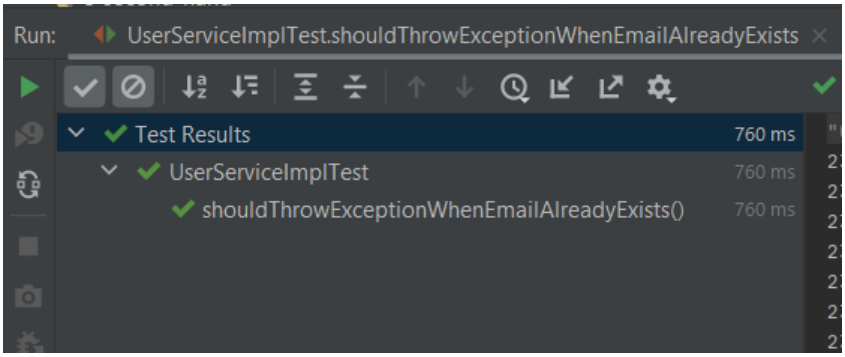
Testy integracyjne, w przeciwieństwie do testów jednostkowych, mogą testować jednocześnie kilka elementów systemu. Mają za zadanie weryfikację poprawności ich współpracy. Do testów integracyjnych wykorzystana została baza danych H2. Dzięki temu wyniki wykonania poleceń SQL nie muszą być symulowane, a zapytania SQL są wykonywane, tylko że na bazie H2. Baza jest tworzona dopiero w momencie uruchomienia jednego z testów, a po jego zakończeniu jest zamykana.

7.2.1. Test tworzenia konta

Test zaprezentowany w listingu 7.3. przedstawia weryfikację metody tworzącej nowe konto. Metoda ma za zadanie zwrócić określony wyjątek w przypadku, jeżeli podany przez użytkownika adres e-mail jest już powiązany z istniejącym kontem. Ponieważ utworzone testy integracyjne na czas ich uruchomienia korzystają z bazy H2, konieczne było jej początkowe wypełnienie. Na początku testu do bazy danych jest dodawany użytkownik z określonym adresem e-mail. Następnie wywoływana jest metoda tworząca nowe konto dla użytkownika z takim samym adresem mailowym. Oczekiwanym rezultatem powinno być zwrócenie wyjątku EmailAlreadyExistsException.

Listing 7.3. Test tworzenia konta [opracowanie własne]

```
@Test
public void shouldThrowExceptionWhenEmailAlreadyExists() throws
EmailAlreadyExistsException {
   when(userMapper.mapRegisterUserDtoToUser(any())).thenCallRealMethod();
   RegisterDto registerDto = createRegisterDto("newEmail@email.com", "second
user");
   userService.save(registerDto);
   RegisterDto secondDto = createRegisterDto("newEmail@email.com", "new user");
   String expectedMessage = "Provided email already exists!";

   Exception exception = assertThrows(EmailAlreadyExistsException.class, () ->
{
       userService.save(secondDto);
   });

   assertTrue(exception.getMessage().contains(expectedMessage));
}
```



Rys. 6.41. Wynik uruchomienia testu tworzenia nowego konta [opracowanie własne]

7.2.2. Test tworzenia nowego czatu

Kiedy użytkownik wysyła nową wiadomość do innego członka systemu, system powinien najpierw utworzyć nową konwersację. Test opisany w tym podrozdziale, weryfikuje poprawność tworzenia nowych rekordów w tabeli czatów oraz uczestników czatów.

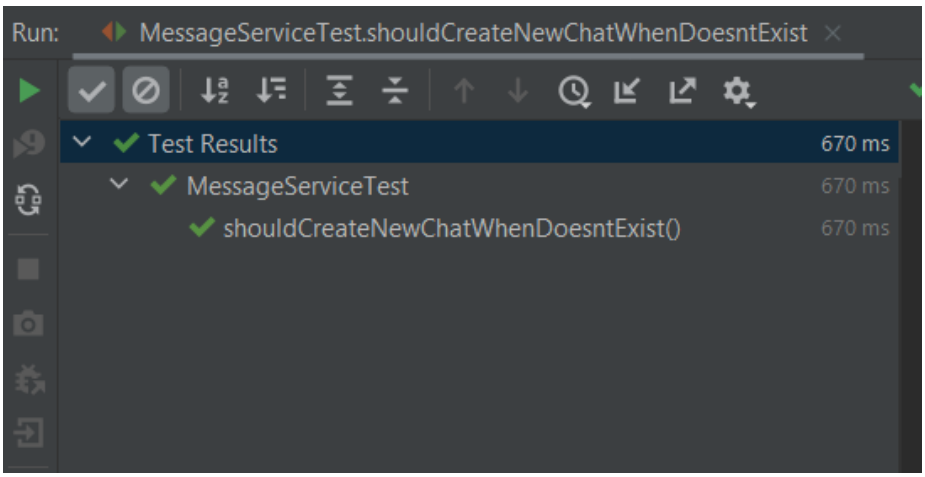
Listing 7.4. Test tworzenia nowego czatu [opracowanie własne]

```
@Test
public void shouldCreateNewChatWhenDoesntExist() {

   User loggedUser = userRepository.save(createAppUser("logged",
"logged@mail.com"));
   User secondUser = userRepository.save(createAppUser("second",
"second@mail.com"));
   Authentication authentication = mock(Authentication.class);
   SecurityContext securityContext = mock(SecurityContext.class);
   Mockito.when(securityContext.getAuthentication()).thenReturn(authentication);
   SecurityContextHolder.setContext(securityContext);
   when((AppUser) SecurityContextHolder.getContext().getAuthentication()
           .getPrincipal()).thenReturn(new AppUser(loggedUser));
   MessageEntryDto messageEntryDto = new MessageEntryDto(null,
secondUser.getId(), "test message");

   messageService.saveMessage(messageEntryDto);

   Chat chat = chatRepository.findChat(loggedUser.getId(), secondUser.getId());
   assertNotNull(chat);
   assertNotNull(chatParticipantRepository.findChatParticipant(loggedUser.getId(
), chat.getId()));
   assertNotNull(chatParticipantRepository.findChatParticipant(messageEntryDto.g
etReceiverId(), chat.getId()));
}
```

Na początku testu, tworzonych jest dwóch użytkowników, którzy będą reprezentować osobę wysyłającą wiadomość oraz jej odbiorcę. Ponieważ korzystanie z funkcji wysyłania wiadomości wymaga bycia zalogowanym, konieczne było zasymulowanie poprawnego uwierzytelnienia pierwszego użytkownika. Następnie wywoływana jest metoda odpowiedzialna za zapisanie nowej wiadomości. Warunkami poprawności testu są:

- utworzenie nowego rekordu w tabeli chat,
- dodanie dwóch rekordów do tabeli chat_participant, które odnoszą się do zdefiniowanych użytkowników oraz nowo utworzonego czatu.

Wynik uruchomienia testu został przedstawiony na rysunku 6.42.



Rys. 6.42. Wynik uruchomienia testu tworzenia nowego czatu [opracowanie własne]

7.3. Testy API

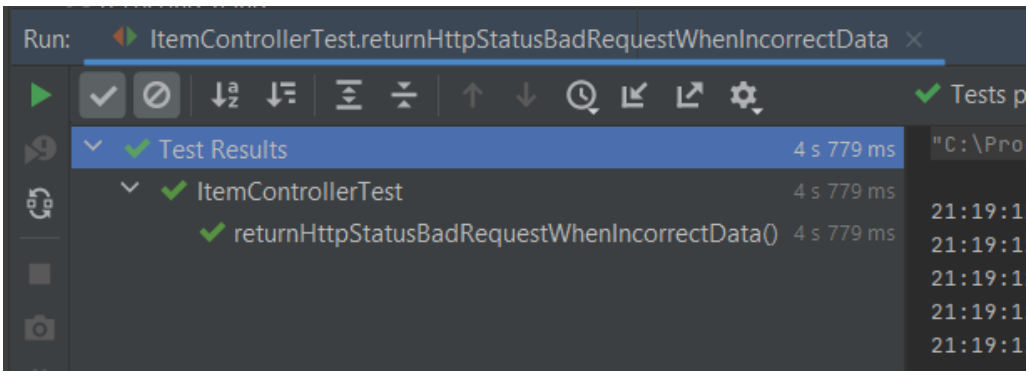
Wykorzystując narzędzie MockMvc, przetestowane zostały również punkty końcowe aplikacji (endpointy). Narzędzie MockMvc umożliwia wywoływanie API w środowisku testowym.

7.3.1. Test żądania dodającego nowe ogłoszenia

W listingu 7.5. przedstawiony został test API dotyczącego dodawania nowego ogłoszenia. Za pomocą biblioteki Mockito symulujemy zwrócenie wyjątku przez metodę zapisującą nowy przedmiot, oznaczającego błędną liczbę przesłanych zdjęć. Wywołując testowane API, oczekujemy, że zostanie zwrócony kod błędu HTTP o wartości 400, wskazujący na błąd po stronie klienta. Rysunek 6.43. przedstawia efekt uruchomienia opisanego testu.

Listing 7.5. Test żądania dodającego nowe ogłoszenie [opracowanie własne]

```
@Test
public void returnHttpStatusBadRequestWhenIncorrectData() throws Exception {
   when(itemService.saveItem(any())).thenThrow(InvalidImagesNumberException.clas
s);
   mockMvc.perform(multipart("/items")
           .param("name", "testname")
           .param("price", String.valueOf(90))
           .param("sex", "WOMAN")
           .param("categoryId", "4")
           .param("sizeId", "8")
           .param("brand", "Zara")
           .param("colorId", "2"))
           .andExpect(MockMvcResultMatchers.jsonPath("$").doesNotExist())
           .andExpect(status().isBadRequest())
           .andDo(print());
}
```



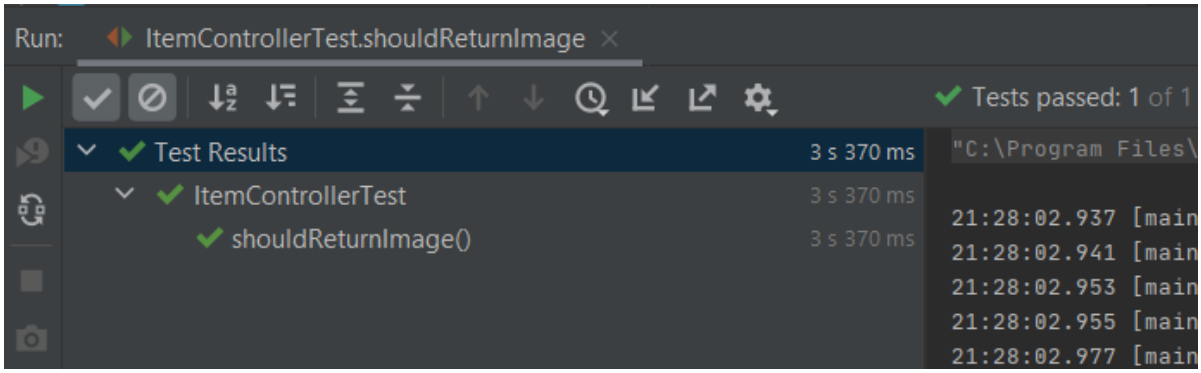
Rys. 6.43. Wynik uruchomienia testu żądania dojącego nowe ogłoszenie [opracowanie własne]

7.3.2. Test żądania pobierającego zdjęcie przedmiotu

Test przedstawiony w listingu 7.6. weryfikuje poprawność zwracania zdjęcia przedmiotu wystawionego na sprzedaż za pomocą aplikacji. Na potrzeby testu, symulujemy zwracanie określonego zdjęcia przez metodę serwisu. Następnie wywoływane jest żądanie HTTP odpowiedzialne na zwrócenie zdjęcia przedmiotu. Warunkiem poprawności testu jest zgodność bajtów pliku zwracanego przez serwis z tym, które zwraca żądanie. Typem treści odpowiedzi serwera powinien być typ IMAGE_JPEG.

Listing 7.6. Test żądania pobierającego zdjęcie przedmiotu [opracowanie własne]

```
@Test
public void shouldReturnImage() throws Exception {
   FileSystemResource image = new
FileSystemResource("src/test/resources/images/test-1234.png");
   when(itemService.find(Mockito.anyLong())).thenReturn(image);
   mockMvc.perform(get("/items/image/1"))
           .andExpect(MockMvcResultMatchers.status().isOk())
           .andExpect(MockMvcResultMatchers.content().contentTypeCompatibleWith(
MediaType.IMAGE_JPEG))
           .andExpect(MockMvcResultMatchers.content().bytes(IOUtils.toByteArray(
image.getURL())));
}
```



Rys. 6.44. Wynik uruchomienia testu żądania pobierającego zdjęcie przedmiotu [opracowanie własne]

8. Podsumowanie

Celem pracy dyplomowej było utworzenie aplikacji webowej, pełniącej funkcję serwisu ogłoszeniowego. Wszystkie z wymagań funkcjonalnych oraz niefunkcjonalnych opisanych w drugim rozdziale zostały zrealizowane:

- aplikacja umożliwia wystawianie na sprzedaż własnych przedmiotów,
- użytkownicy mogą przeglądać oraz filtrować dodane przedmioty,
- umożliwiony został kontakt pomiędzy członkami systemu,
- utworzony został panel moderatora, który umożliwia zarządzanie treścią serwisu, uprawnionym do tego osobom,
- zapewnione zostało bezpieczeństwo systemu,
- zastosowane zostały elementy implementacyjne optymalizujące działanie aplikacji,
- interfejs aplikacji jest spójny oraz jasny.

Utworzona aplikacja posiada wiele możliwości przyszłego rozwoju. Pierwszym etapem rozwoju powinno być rozbudowanie funkcji moderatora. Na obecnym etapie moderator ma możliwość dodawania kategorii, natomiast powinien on móc zarządzać również innymi tabelami, czyli np. markami, kolorami oraz rozmiarami. Ważnym elementem byłoby także umożliwienie moderatorowi zarządzania użytkownikami – powinien móc np. usuwać konta użytkowników, którzy regularnie łamią standardy aplikacji.

Jeśli chodzi o funkcje, które umożliwiłyby przynoszenie korzyści finansowych dla właścicieli aplikacji, jedną z nich mogłaby być funkcja promowania ogłoszeń, tak jak ma to miejsce w przypadku podobnych systemów. Kolejną funkcją, wartą rozważenia, jest możliwość płatności za pośrednictwem aplikacji – przy takiej transakcji pobierana mogłaby być opłata, a użytkownicy korzystający z tej opcji byliby dodatkowo chronieni z jej strony. Pieniądze nie byłyby przelewane na konto właściciela przedmiotu, dopóki kupujący nie potwierdzi, że przedmiot jest zgodny z opisem.

Omówiona praca pozwoliła na udoskonalenie wiedzy uzyskanej w trakcie studiów z zakresu tworzenia aplikacji internetowych. Pozwoliła na poznanie wielu ciekawych, nowych technologii oraz zbadanie funkcjonowania systemów serwisów ogłoszeniowych, których w dzisiejszych czasach nie brakuje na rynku.

Bibliografia

[1] Jon Duckett. *HTML i CSS. Zaprojektuj i zbuduj witrynę WWW. Podręcznik Front-End Developera.* Helion 2017

[2] *HTTP – dokumentacja przeglądarki Mozilla.* [Online] https://developer.mozilla.org/pl/docs/Web/HTTP/ dostęp 01.2022

[3] *Index TIOBE.* [Online] https://www.tiobe.com/tiobe-index/ dostęp 01.2022

[4] Martin Fowler. *Inversion of Control Containers and Dependency Injection Pattern.* [Online] http://www.martinfowler.com/articles/injection.html dostęp 01.2022

[5] Herbert Schildt. *Java. Kompendium programisty*. Wydanie XI. Helion. 2005

[6] *JavaScript – dokumentacja przeglądarki Mozilla.* [Online] https://developer.mozilla.org/pl/docs/Web/JavaScript/ dostęp 01.2022

[7] *JSON - dokumentacja techniczna.* [Online] https://www.json.org/json-en.html dostęp 01.2022

[8] Vincent Massol, Ted Husted. *JUnit in Action.* Manning 2003, First Edition

[9] *JWT – dokumentacja techniczna.* [Online] https://jwt.io/introduction/ dostęp 01.2022

[10] *Maven - dokumentacja techniczna*. [Online] https://maven.apache.org/ dostęp 01.2022

[11] *Modowe cmentarzysko na pustyni Atakama* [Online] https://magazynbiomasa.pl/modowe-cmentarzysko-na-pustyni-atakama/ dostęp 01.2022

[12] *PostgreSQL – dokumentacja techniczna.* [Online] https://www.postgresql.org/about/ dostęp 01.2022

[13] Zdzisław Dybikowski. *PostgreSQL*. Helion wydanie II, 2001

[14] Andrzej Zoła. *Programowanie w języku Java.* 2004

[15] Kirupa Chinnathambi. *React i Redux. Praktyczne tworzenie aplikacji WWW.* Wydanie II, Helion 2019

[16] *React.js – dokumentacja techniczna.* [Online] https://pl.reactjs.org/ dostęp 01.2022

[17] *Sass – dokumentacja techniczna.* [Online] https://sass-lang.com/documentation/ dostęp 01.2022

[18] *Spring – dokumentacja techniczna.* [Online] https://docs.spring.io/ dostęp 01.2022

[19] Craig Walls. *Spring w akcji*. Wydanie IV. Helion 2015.

[20] *Swagger - dokumentacja techniczna.* [Online] https://swagger.io/docs/specification/about/ dostęp 01.2022

[21] Eric Jendrock, Jennifer Ball, Debbie Carson, Ian Evans, Scott Fordin, Kim Haase. *The Java EE 5 Tutorial.* [Online] https://docs.oracle.com/javaee/5/tutorial/doc/bnbpz.html dostęp 01.2022

[22] I.Fette, A.Melnikov. *The WebSocket Protocol*. [Online] https://www.hjp.at/doc/rfc/rfc6455.html dostęp 01.2022

[23] Martin Haverbeke. *Zrozumieć JavaScript. Wprowadzenie do programowania.* Helion 2015, wydanie II

Spis rysunków

Spis listingów