Państwowa Wyższa Szkoła Zawodowa w Tarnowie

Wydział Politechniczny

Kierunek: Informatyka

Specjalność: Informatyka stosowana

Specjalizacja: Inżynieria Systemów Informatycznych

2021/2022

Patryk Starzycki

PRACA INŻYNIERSKA

Aplikacja do rezerwacji wizyt pacjentów w gabinecie fizjoterapii

Promotor pracy:

mgr inż. Tomasz Gądek

Tarnów, 2022

Spis treści

1. Wstęp

W dzisiejszych czasach aplikacje webowe znane również jako strony internetowe, są bardzo popularnym i zarazem użytecznym narzędziem. Każdy portal stanowi wizytówkę danej firmy. Jej głównym zadaniem jest zachęcić użytkownika do ponownego jej odwiedzenia, bądź skorzystania z usług, jakie placówka oferuje. Walka o potencjalnego klienta sprawia, że strony starają się wyróżniać spośród innych np. poprzez zaprojektowanie unikalnego designu, implementacja funkcji, które w znacznej mierze ułatwiają wykonywanie czynności np. rezerwacje online. Dodatkowo rozwój technologii sprawił, iż system na podstawie jego ruchu po stronie jest w stanie szybko odgadnąć preferencje użytkownika, dzięki czemu wyświetlana zawartość trafia w jego gust. Jednak są kategorie usług, w których nie zdecydowano się jeszcze na taką walkę o klienta.

1.1. Wybór tematu

Głównym powodem, dla którego podjąłem się przygotowania aplikacji do rezerwacji wizyt pacjentów w gabinecie fizjoterapii, jest nieduża ilość portali pozwalających na zarządzanie własnymi wizytami online — ich rezerwacje, modyfikacje czy informacje o ewentualnych opóźnieniach w danym dniu. Strony takich firm bardzo często ograniczają się do zaprezentowania własnej oferty — ich opisów, cen oraz prezentacji personelu. Potencjalny pacjent, aby uzyskać więcej informacji, zmuszony jest do kontaktu telefonicznego bądź osobistego, co nieraz może powodować dyskomfort, jakim jest oczekiwanie w kolejkach. Dodatkowym problemem dzisiejszych czasów jest pandemia, której efektem jest wprowadzenie licznych ograniczeń bądź dystansu społecznego. W takiej sytuacji dostęp do aplikacji z takimi możliwościami nie narażałby pacjenta na niepotrzebne niebezpieczeństwa z tego tytułu.

1.2. Cel i zakres projektu

Celem projektu jest zaprojektowanie systemu, którego głównym zadaniem ma być usprawnienie procesu rejestracji pacjenta. System zostanie zaimplementowany w oparciu o architekturę klient-serwer. Dodatkowo wszystkie dane potrzebne do prawidłowego funkcjonowania systemu będą przechowywane w relacyjnej bazie danych. W ramach wersji prezentacyjnej przygotowana zostanie strona internetowa.

System zakłada istnienie 4 ról. W zależności od posiadanej przez użytkownika roli aplikacja umożliwi:

- jako niezalogowany użytkownik:
    - przeglądanie oferty jako całość oraz jako poszczególne zabiegi ze szczegółowymi informacjami celu zabiegu, cenie, czasie trwania, dostępnych specjalistach, ocenie przez innych pacjentów,
    - rejestrację, logowanie, przypomnienie hasła.
- jako pacjent:
    - ocenę zabiegów po wizytach,
    - zapisywanie na wizyty,
    - dostęp do historii wizyt,
    - edycję konta.
- jako lekarz:
    - zmianę statusu wizyty,
    - dodawanie zaleceń do wizyty,
    - edycję konta.
- jako administrator:
    - zarządzanie pacjentami,
    - zarządzanie lekarzami,
    - zarządzanie ofertą.

2. Wybór technologii

2.1. Aplikacja webowa

2.1.1. JavaScript i TypeScript

JavaScript wraz z HTML i CSS stanowią podstawowe narzędzia do tworzenia aplikacji webowych. Jest to dynamicznie typowany język skryptowy. Język JavaScript służy do programowania skryptów uruchamianych po stronie klienta. [8]

TypeScript jest to język programowania, który bazuje na JavaScriptcie oraz dodaje do niego nowe funkcje, takie jak statyczne typowanie czy interfejsy. Ułatwia on utrzymanie kodu, jak również zwiększa jego stabilność.

Zarówno JavaScript, jak i TypeScript mogą pracować w środowisku NodeJS. Dzięki temu możliwe jest tworzenie serwerów w tych językach.

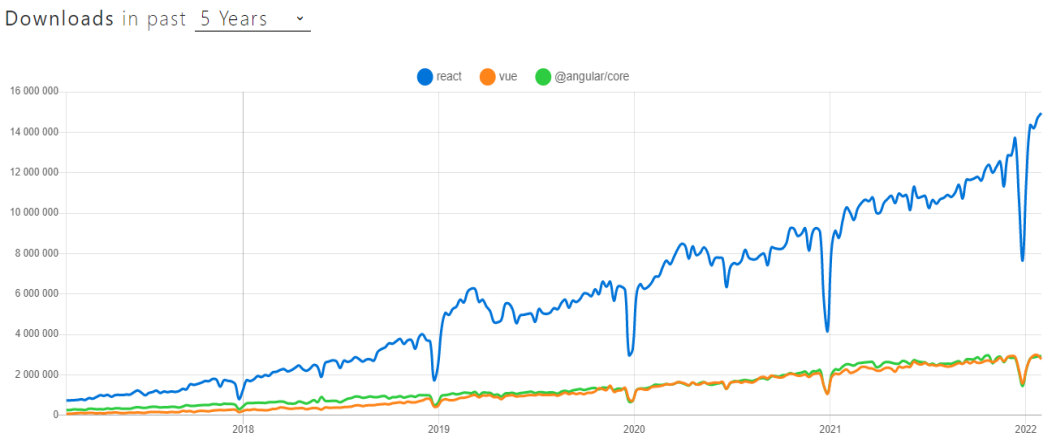
2.1.2. ReactJS

ReactJS jest to dostępna na zasadach open source biblioteka języka JavaScript. Używana do tworzenia interfejsów graficznych. Opracowana została przez jednego z pracowników Facebooka — Jordana Walke, którego głównym celem było rozwiązanie problemów, jakie towarzyszyły procesowi tworzenia SPA - aplikacji jednostronicowej. [13] Jej charakterystyczną cechą jest fakt, że jedyną rzeczą, która ulega zmianie to zawartość poszczególnych elementów (dane oraz kod HTML odpowiadający danemu stanowi).

Wspomniany wcześniej proces tworzenia napotyka jednak kilka poważnych problemów:

- Synchronizacja danych,
- Bardzo długi czas wykonania operacji na modelu DOM,
- Podmienianie fragmentów kodu HTML.

Rozwiązaniem tych trudności miał być właśnie React, który sprawdził się w swojej roli. Dodatkowo wprowadził kilka usprawnień, dzięki czemu budowanie SPA stało się znacznie prostsze.

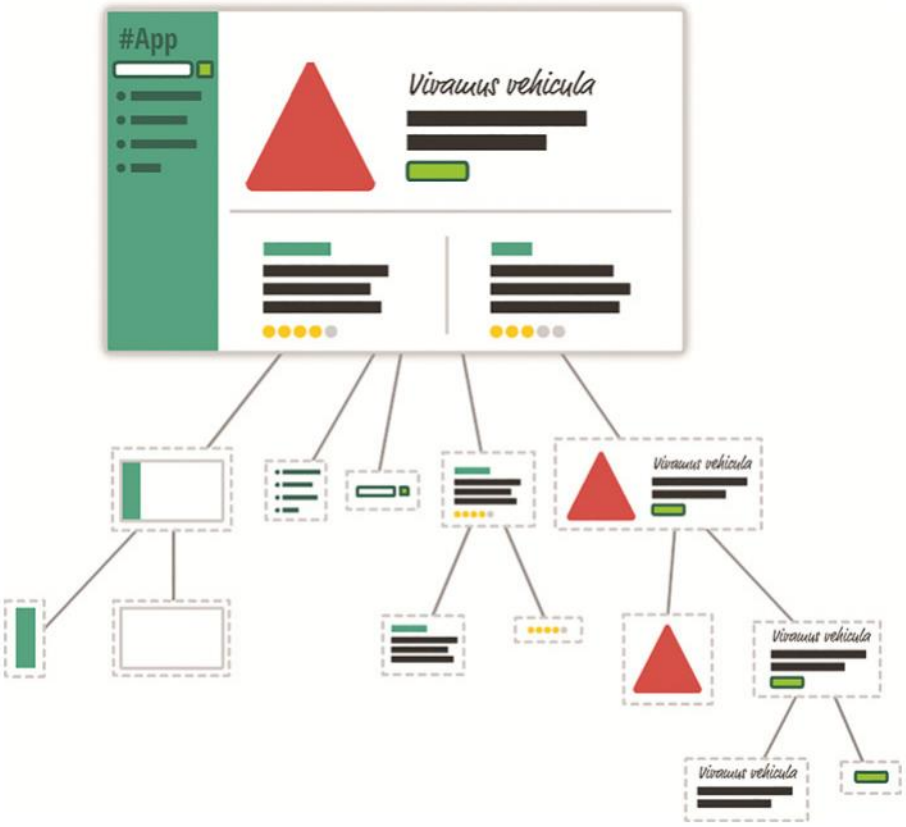


Rysunek 1 Wykres ilości pobrań narzędzi JS z ostatnich 5 lat , źródło: [16]

Wykres przedstawiony na rysunku 1 pokazuje, iż pojawienie się nowych narzędzi nie przeszkodziło React w pozostaniu najpopularniejszą biblioteką JavaScript do tworzenia aplikacji WWW. Została ona wykorzystana przy implementacji wielu znanych serwisów jak np. Netflix, Facebook, Instagram.

Charakterystycznymi elementami Reacta są:

- Automatyczne zarządzanie stanem interfejsu użytkownika, dzięki czemu programista może skupić się na wyglądzie końcowym strony, resztą zajmuje się biblioteka,
- Błyskawiczne modyfikowanie modelu DOM - React wprowadza Virtual DOM, który jest kopią właściwego DOM'a. Dzięki temu rozwiązaniu wprowadzanie zmian w modelu odbywa się wyjątkowo szybko, a biblioteka React aktualizuje w odpowiednim momencie rzeczywisty model DOM. W tym celu porównuje model wirtualny i rzeczywisty. Ocenia, które z dokonanych zmian są istotne i w procesie uzgadniania wprowadza w rzeczywistym modelu DOM minimalną liczbę zmian, aby był on aktualny.
- Podział interfejsu użytkownika na coraz mniejsze komponenty - strona przestała być jednym wielkim, graficznym monolitem. Główną ideą podczas implementacji stało się tworzenie mniejszych, samodzielnych części, z których następnie buduje się bardziej złożone komponenty.

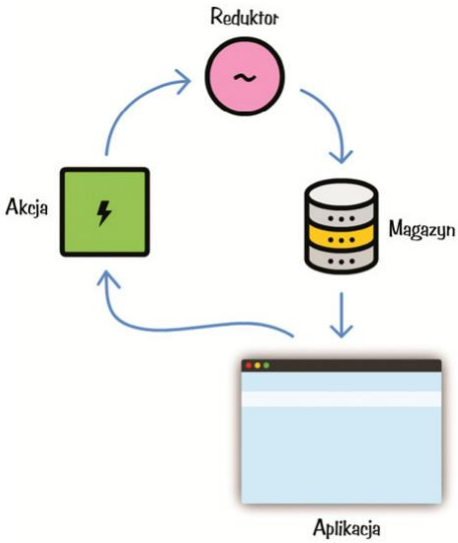


Rysunek 2 Schemat budowy strony przy użyciu React'a, źródło: [13]

Charakterystycznym elementem React'a jest Virtual DOM. Jest to kopia właściwego DOM'a, dzięki której strona internetowa jest w stanie zmienić swoją zawartość bez konieczności odświeżania całej strony. Ponadto każdy tworzony element może być używany w innym miejscu, przez co pozwala to zaoszczędzić czas oraz zwiększyć wydajność pracy. Co więcej, z racji na swoją popularność jest ciągle rozwijany, co sprawia, że jest to stabilne rozwiązanie do tworzenia interfejsów.

2.1.3. Redux

Redux to biblioteka JavaScript, która służy do zarządzania stanem aplikacji. [13] Jest ona rozwinięciem architektury flux - wzorzec architektury aplikacji, której główną cechą jest jednokierunkowy przepływ informacji. Redux pozwala oddzielić warstwę administrowania stanem aplikacji od kodu odpowiedzialnego za jego prezentacje. Cały ów stan przechowywany jest w jednym *magazynie*.



Rysunek 3 Schemat przepływu pracy Redux'a , źródło: [13]

Magazyn ma to do siebie, że dane odczytuje się z niego bardzo łatwo, natomiast umieszczanie w nim informacji to zupełnie inna sprawa. Aby zapisać w magazynie nowy stan (lub zmodyfikować istniejący), trzeba stosować *akcje* opisujące dane, które trzeba zmienić, oraz *reduktor* określający końcowy stan, który jest wynikiem wykonania akcji. [13] Każda z nich jest pojedynczo analizowana, dzięki czemu eliminuje się wszelkie wyścigi, a tym samym ułatwia debugowanie.

Za sprawą Reduxa zarządzanie stanem aplikacji staje się przewidywalne, bezpieczne oraz łatwe do utrzymania .

2.1.4. Bootstrap i Material UI

Obie technologie odpowiedzialne są za część wizualną strony internetowej. Bootstrap jest to biblioteka CSS rozwijana przez programistów Twittera, wydawana na licencji MIT [3]. Jego głównymi zadaniami są przyspieszenie procesu tworzenia strony internetowej, ułatwienie korzystania z responsywnej siatki CSS, a także zapobieganie problemom z wyświetlaniem tych samych elementów na różnych przeglądarkach.

Material UI jest zestawem gotowych komponentów do biblioteki React, zaprojektowane zgodnie z zasadami Google Material Design. Pozwala on na szybsze i łatwiejsze projektowanie interfejsu użytkownika.

2.1.5. Axios

Axios jest biblioteką JavaScript, która umożliwia komunikacje z API (Application Programming Interface) udostępnianym przez serwer. *Aplikacyjny*

interfejs programistyczny jest zestawem reguł definiujących sposób, w jaki aplikacje lub urządzenia mogą się ze sobą łączyć i komunikować. [7] Przy użyciu protokołu HTTP axios wysyła żądania do serwera oraz oczekuję na jego odpowiedź. W przypadku otrzymania odpowiedzi, wydobywane są konkretne dane.

2.2. Serwer

2.2.1. Java

Java to wszechstronne ogólne środowisko programistyczne. Składają się na nie język Java i pomocnicze środowisko wykonawcze zwane maszyną wirtualną Javy (JVM). [10] Powstało ono pod koniec lat 90. ubiegłego wieku. Aktualnie Javą zarządza firma Oracle Corporation, ale w tworzenie jej standardowych implementacji duży wkład mają też takie firmy, jak Red Hat, IBM, HP, SAP, Apple oraz Fujitsu. [10]

Język Java to wysokopoziomowy język programowania zorientowany na programowanie obiektowe. Jego założenia to abstrakcja, hermetyczność, dziedziczenie i polimorfizm. Charakterystyczną cechą jest niezależność od architektury dzięki wykorzystaniu JVM, który interpretuje kod niezależnie od tego w jakim środowisku jest zainstalowana („Write Once, Run Anywhere”).

Obecnie Java używana jest niemal wszędzie. Pozwala ona na tworzenie:

- aplikacji desktopowych, które uruchamiają się na każdym systemie operacyjnym,
- aplikacji serwerowych,
- aplikacji webowych,
- aplikacji dla telefonów komórkowych, odległych urządzeń przetwarzających, mikrokontrolerów, modułów bezprzewodowych, sensorów, bramek, produktów konsumenckich oraz praktycznie wszelkich innych urządzeń elektronicznych. [9]

Dodatkowo JVM stał się bardzo uniwersalnym narzędziem. Obecnie obsługuje on także kod napisany w języku Scala, Groove.

2.2.2. Spring Framework

Spring jest frameworkiem open source, zaproponowanym przez Roda Johnsona. [14] Pomaga implementować aplikacje enterprise. Jego głównym zadaniem jest uproszczenie programowania w J2EE. W tym celu korzysta on z czterech kluczowych strategii:

- programowanie lekkie i niezbyt inwazyjne dzięki użyciu obiektów POJO (Plain Old Java Object),
- luźne wiązanie dzięki wstrzykiwaniu zależności i zorientowaniu na interfejsy,
- używanie wzorców co ogranicza powielanie kodu,
- programowanie deklaratywne z użyciem aspektów i wspólnych konwencji. [14]

Framework składa się z wielu modułów, które zawierają wszystko, co potrzebne do przygotowania aplikacji enterprise. W skład wchodzą:

- Core,
- Testing,
- AOP,
- Spring MVC,
- Integration,
- Data Access.

Z czasem zaczęły powstawać dodatkowe projekty, które rozszerzył możliwości deweloperów m.in. Spring Data, Spring Boot, Spring Security.

2.2.3. Maven

Maven jest narzędziem mającym za zadanie wspomagać programistów Java. Oferuje on wiele możliwości. Ułatwia on m.in. dodawanie zależności do projektu, kompilacje, budowanie oraz przeprowadzanie testów. Do rozpoczęcia użytkowania wystarcza podstawowa konfiguracja. Wszystkie ustawienia przechowywane są w pliku o nazwie pom.xml. W nim możemy m.in. zdefiniować takie wartości jak:

- nazwa projektu,
- opis,
- właściwości projektu - w tym np. wersje Javy,
- zależności,
- dodatkowe pluginy.

Maven po uruchomieniu konfiguruje cały projekt zgodnie z tymi wartościami oraz automatycznie pobiera wszystkie zależności z oficjalnego repozytorium, przez co po skończonym procesie budowania mamy dostęp do wszystkich zadeklarowanych bibliotek.

2.2.4. Spring Data JPA

Spring Data jest to jeden z komponentów Springa. Jego zadaniem jest ułatwienie komunikacji z bazą danych przy pomocy ORM (mapowanie obiektowo-relacyjne). Ułatwia on dostęp do bazy danych poprzez wbudowane zapytania, dzięki którym programista nie potrzebuje pisać zapytań ręcznie. Bazuje on na JPA (Java Persistence Api), którego dostawcą jest Hibernate. [17]

JPA jest specyfikacją zawierającą standardy mapowania ORM. Na JPA składają się interfejsy, które muszą być zaimplementowane, aby móc przeprowadzać operacje na bazach danych. Możliwe jest również pisanie samemu metod dostępu. Wówczas do pisania zapytań używa się JPQL (Java Persistence Query Language) - język zapytań operujący na obiektach.

Najważniejszą zaletą Spring Data jest fakt, że nie trzeba niczego implementować, aby móc przeprowadzać operacje na bazie danych. Wystarczy użyć interfejsu dziedziczącego po JpaRepository. Wówczas otrzymujemy dostęp do podstawowych operacji tj. findAll, findById, save lub deleteById. Dodatkowo w sytuacji, gdy potrzeba bardziej złożonych operacji, wystarczy dopisać odpowiednią metodę, zgodnie z konwencją nazewniczą, a Spring sam wygeneruję do niej implementacje.

2.2.5. Spring Security

Spring Security jest to jeden z komponentów Springa. Pozwala on na dodanie do aplikacji warstwy uwierzytelnienia i kontroli dostępu. Jego głównym elementem są filtry. Każde zapytanie przechodzi przez zestaw „kryteriów”, z których każde może zdecydować o tym, czy odmówić dostępu, udzielić go lub przekazać decyzję dalej. [6] Kryteria te mogą dotyczyć pojedynczych zapytań bądź całych grup URL. Dodatkowo pozwala na szeroką konfigurację. Efektem tego może być np. umożliwienie logowania za pomocą zwykłego e-maila i hasła bądź też kont z popularnych serwisów tj. Google, Facebook itp.

2.3. Serwer firebase

Firebase to opracowana przez firmę Firebase Inc. następnie wykupiona przez Google platforma do tworzenia aplikacji mobilnych oraz webowych. Jest to rozwiązanie typu BaaS (Backend as a Service). Jego głównym celem jest umożliwienie tworzenie aplikacji bez znajomości zagadnień backendowych. Dzięki swoim modułom jest w stanie całkowicie przejąć zadania serwera, co przyspiesza cały proces tworzenia aplikacji. Dodatkowo podstawowy, darmowy plan nie ogranicza jego funkcjonalności. Narzędzie jest w stanie także współpracować z innymi serwerami. Dzięki czemu możliwy jest podział zadań w zależności od jego charakteru.

Najważniejsze moduły:

- Cloud Firestore - jest to baza NoSQL w chmurze. Jest ona elastyczna i skalowalna. Dane przechowywane są w dokumentach, które mogą być gromadzone w kolekcjach. W dokumencie następnie można dodawać kolejne podkolekcje itd. Buduje się dzięki temu przejrzystą, hierarchiczną strukturę danych. Największą zaletą Cloud Firestore jest możliwość pobierania danych w czasie rzeczywistym. Dzięki temu aplikacja otrzymuje dane w momencie ich zmiany w bazie danych. Ponadto możliwe jest tworzenie lokalnych baz na urządzeniu, które umożliwiają działanie aplikacji bez połączenia z internetem. Dane są ponownie synchronizowane, gdy połączenie zostanie nawiązane. Dodatkowym atutem jest prosta obsługa.
- Firebase Authentication - służy do uwierzytelnienia i autoryzacji użytkowników. Obsługuje uwierzytelnienia za pomocą haseł, numerów telefonów oraz popularnych dostawców tożsamości tj. Google, Facebook, Apple i wielu innych, czy też pozwala na używanie konta anonimowego. Uwierzytelnianie Firebase jest ściśle zintegrowane z innymi usługami Firebase i wykorzystuje standardy branżowe, takie jak OAuth 2.0 i OpenID Connect. [5] Cały ciężar weryfikacji spoczywa na serwerach firebase. Ponadto usługa udostępnia narzędzia do wysyłania wiadomości na telefon bądź e-mail, mające za zadanie resetowanie haseł, bądź weryfikacje konta.
- Firebase Storage - usługa ta służy do przechowania plików w chmurze. Pozwala na przesyłanie i pobieranie danych niezależnie od jakości sieci. Oznacza to, że w przypadku przerwania połączenia działanie jest wstrzymywane,

a następnie wznowione w tym samym miejscu po przywróceniu dostępu do sieci. Dzięki temu oszczędzany jest czas oraz użycie danych użytkownika.

- Google Analytics - jest to zbiór narzędzi pozwalających zrozumieć, w jaki sposób użytkownicy korzystają z aplikacji. Po poprawnym skonfigurowaniu usługa automatycznie przechwytuje wiele zdarzeń - zarówno wbudowanych, jak i niestandardowych, opracowanych przez programistę. Głównym celem jest umożliwienie prowadzenia statystyk strony, poznanie potrzeb użytkowników tak, aby móc podejmować słuszne decyzje dotyczące marketingu.[4] Dodatkowo zapewnia bardzo przyjemne dla oka przejrzyste UI, w którym przedstawione są zebrane dane - od danych ogólnych strony, po szczegółowe dane statystyczne np. dane demograficzne, czy też informacje o stronie z danej chwili.

2.4. Relacyjna baza danych

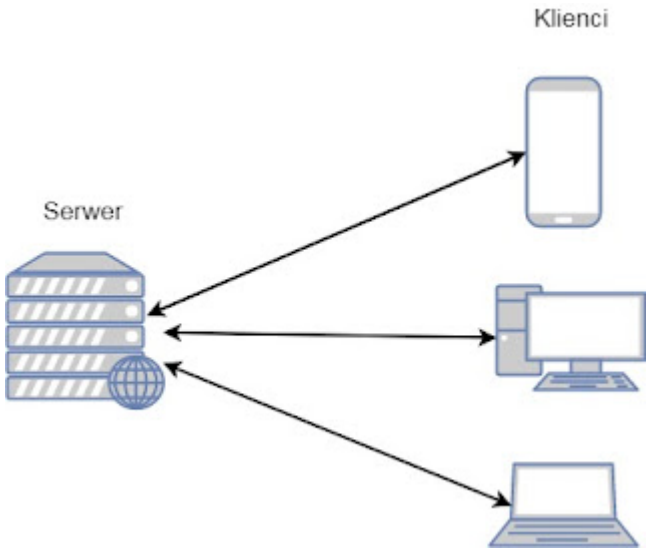
Baza danych przechowuje dane w odpowiedniej strukturze. Charakterystycznym językiem baz jest SQL (Structured Query Language). Dzięki niemu możliwy jest dostęp do zapisanych informacji. Dodatkowo pozwala on przeprowadzać konkretne operacje na bazie tj. zapis, odczyt, modyfikacja czy usuwanie. Istnieje wiele rodzajów baz. Najpopularniejszym jest relacyjna baza danych. Jej najczęściej używane implementacje to: PostgreSQL, MySql czy SQLite. Bazuje ona na modelu relacyjnym, za którą stoi algebra relacji. W tym modelu dane grupowane są w relacje, które reprezentowane są przez tabele. Tabele są zbiorem rekordów o identycznej strukturze. Z nich zaś zbudowany jest schemat bazy danych. [12] Takie podejście sprawia, że dużo prostsze staje się wprowadzanie zmian, dodatkowo zmniejsza szanse na pomyłkę. Minusem jednak jest wydajność.

3. Dokumentacja techniczna

3.1. Opis architektury

Projekt powstał zgodnie z modelem klient-serwer. Jest to architektura systemu komputerowego, w szczególności oprogramowania, umożliwiająca podział zadań. [1] Podział polega na ustaleniu, iż serwer zapewnia usługi dla klientów, którzy zgłaszają się z żądaniami obsługi. Wyróżniamy 2 formy tej architektury:

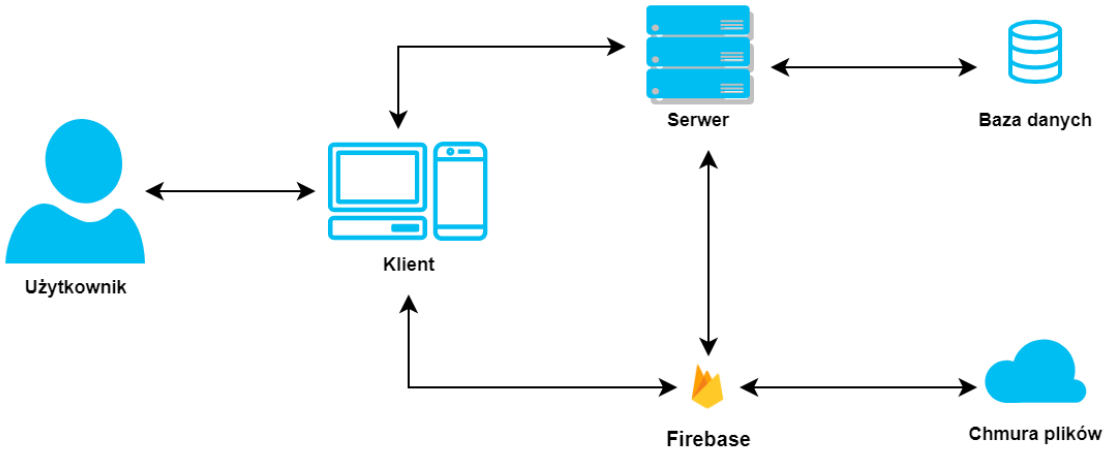
- dwuwarstwowa, w której istnieje tylko jeden serwer,
- wielowarstwowa, w której istnieje więcej niż jeden serwer.



Rysunek 4 Model architektury klient-serwer, źródło: [2]

Warunkiem udanej komunikacji jest używanie tego samego protokołu zarówno przez serwer, jak i klientów. Charakterystycznymi cechami warstwy klienckiej jest posiadanie interfejsów, których zadaniem jest wyświetlanie otrzymanych danych z serwera oraz umożliwienie interakcji użytkownika. Serwer zaś może mieć dostęp do bazy danych, z którą łączy się w sytuacjach, gdy potrzebuje konkretnych informacji tam przechowywanych.

Do zalet tego modelu architektury można zaliczyć zwiększone bezpieczeństwo danych, gdyż to serwer decyduje o dostępie oraz łatwiejsze zarządzanie tymi danymi z racji, że baza danych jest scentralizowana. Największą wadą jest zależność od serwera — w sytuacji, gdy serwer przestaje działać, dostęp do danych jest, niemożliwy co bardzo często oznacza awarie całego systemu. Ponadto w sytuacji, gdy duża ilość klientów łączy się z serwerem mogą powstać różne problemy związane z przepustowością łącza.



Rysunek 5 Model architektury aplikacji ,
źródło: opracowanie własne

System wykorzystuje model w formie wielowarstwowej. Rysunek 5 pokazuje przepływ danych pomiędzy poszczególnymi elementami systemu.

3.2. Charakterystyka elementów systemu

System obsługi rezerwacji pacjentów został zaprezentowany na rysunku 5. Składa się z 5 elementów. Są to: aplikacja webowa, serwer do obsługi bazy danych, serwer firebase, baza danych oraz chmura plików. Ściśle ze sobą współpracują poprzez wykonywanie przypisanych im zadań.

3.3. Charakterystyka aktorów systemu

W systemie istnieją 4 rodzaje aktorów:

- użytkownik niezalogowany - gość,
- użytkownik zalogowany - pacjent,
- użytkownik zalogowany - lekarz,
- administrator.

Każda z nich ma inne uprawnienia, czego efektem jest dostęp do odpowiedniego panelu w aplikacji webowej. Tabele 1, 2, 3 oraz 4 przedstawiają ich pełną charakterystykę.

Tabela 1 Opis roli - gość, źródło: opracowanie własne

Użytkownik niezalogowany - gość	
Opis	Jest to każda nie zalogowana osoba przebywająca na stronie. Posiada dostęp do podstawowych stron aplikacji.
Uprawnienia	● przeglądanie oferty, ● kontakt z administracją, ● rejestracja, ● logowanie, ● resetowanie hasła.

Tabela 2 Opis roli - pacjent, źródło: opracowanie własne

Użytkownik zalogowany - pacjent	
Opis	Jest to osoba zalogowana na wcześniej utworzone konto pacjenta. Posiada dostęp do panelu pacjenta, który umożliwia rejestracje na zabiegi oraz wgląd w historię własnych wizyt.
Uprawnienia	● rejestracja na zabiegi, ● wgląd w historię wizyt, ● edycja konta, ● dezaktywacja konta, ● zmiana hasła.

Tabela 3 Opis roli – lekarz, źródło: opracowanie własne

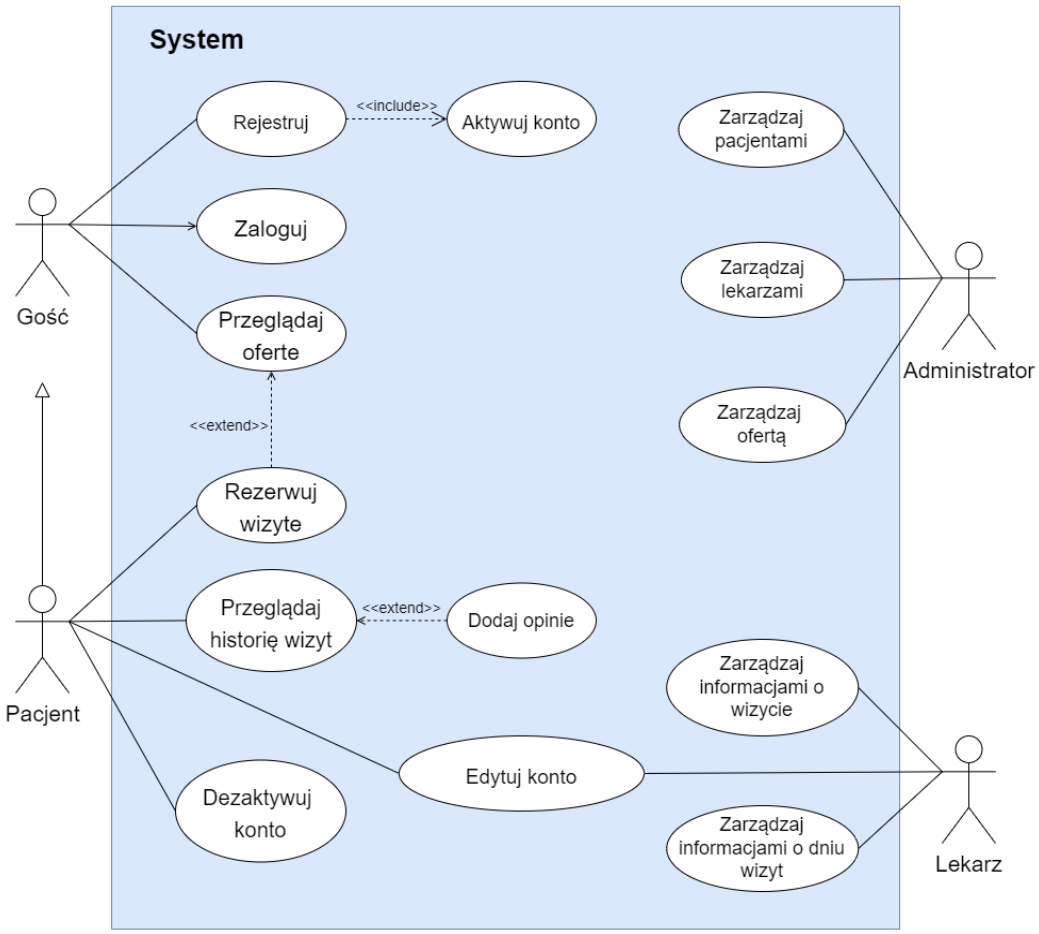
Użytkownik zalogowany - lekarz	
Opis	Jest to osoba zalogowana na utworzone przez administracje konto lekarza. Odpowiedzialna za obsługę zaplanowanych wizyt. Posiada dostęp do panelu lekarza.
Uprawnienia	● aktualizacja informacji o danym dniu, ● aktualizacja statusu wizyty oraz zaleceń, ● edycja konta, ● zmiana hasła.

Tabela 4 Opis roli – administrator, źródło: opracowanie własne

Administrator	
Opis	Osoba zalogowana na specjalnie utworzone konto, odpowiedzialna za poprawne funkcjonowanie strony. Zarządza każdym elementem systemu. Posiada dostęp do panelu administratora w aplikacji webowej.
Uprawnienia	● zarządzanie pacjentami ○ aktywacja/dezaktywacja konta pacjenta ● zarządzanie lekarzami ○ edycja konta lekarza ○ dodawanie/usuwanie przypisanych do konta zabiegów ● zarządzanie ofertą ○ dodawanie nowych zabiegów ○ dodawanie nowych kategorii

3.4. Diagram przypadków użycia

Zgodnie z założeniami projektu, aplikacja do rezerwacji wizyt pacjentów w gabinecie fizjoterapii, powinna umożliwiać wykonywanie pewnych akcji. Diagram przypadków użycia ukazany na rysunku 6 przedstawia je wszystkie. Są one przypisane zgodnie z podziałem na role.

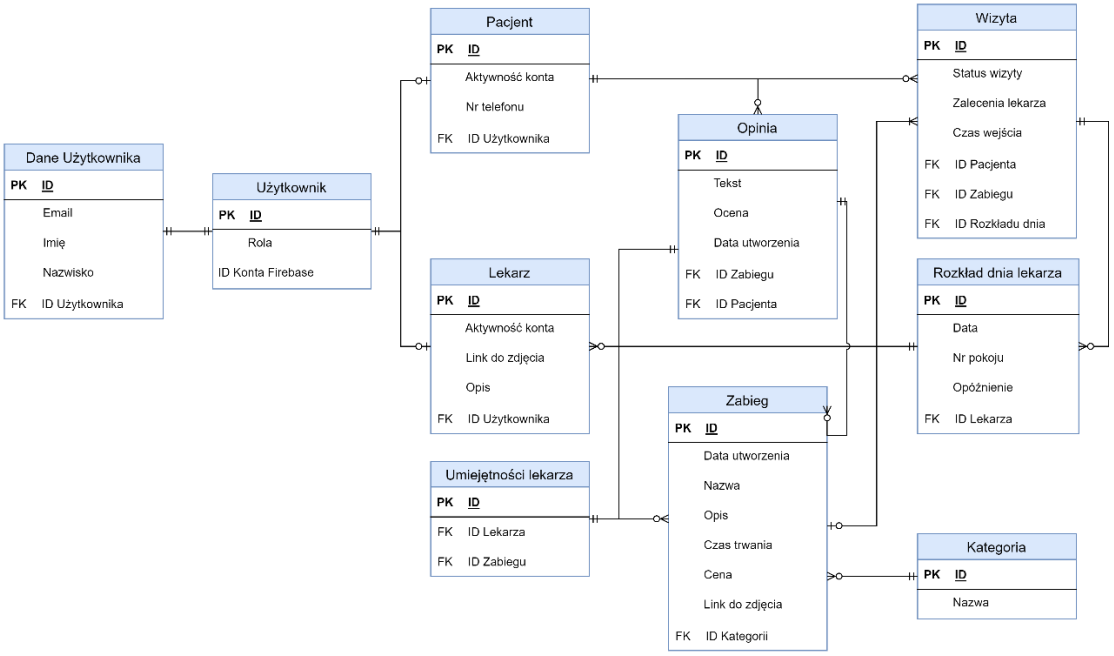


Rysunek 6 Diagram przypadków użycia, źródło: opracowanie własne

3.5. Charakterystyka bazy danych

Do prawidłowego funkcjonowania aplikacja potrzebuje przechowywać duże ilości danych. Informacje o zabiegach, specjalistach, kategoriach muszą być zapisane w bazie danych. Dzięki temu każdy użytkownik korzystający z aplikacji będzie widział to samo. Ponadto system powinien na bieżąco zapisywać informacje o rezerwacjach, recenzjach, nowych użytkownikach.

Mając na uwadze powyższe wymagania, zaprojektowana została baza danych, którą strukturę przedstawiono na rysunku 7.



Rysunek 7 Model bazy danych, źródło: opracowanie własne

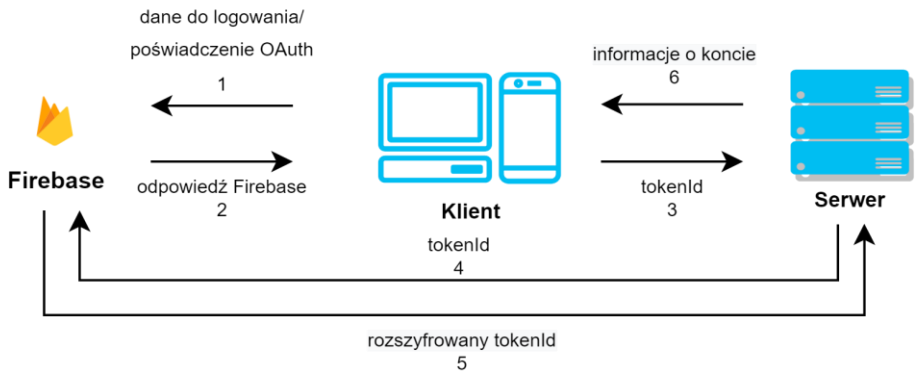
3.6. Bezpieczeństwo w aplikacji

Zawarte w rozdziale dotyczącym aktorów informacje, pokazują istnienie w systemie czterech ról. Z tego względu koniecznym stało się zaprojektowanie mechanizmu pozwalającego rozpoznać typ użytkownika. Wykorzystany do tego został moduł Firebase'a - Authentication oraz Spring Security. Cały proces opiera się na danych zwracanych przez serwer Firebase, po pozytywnej weryfikacji. W ich skład wchodzą:

- idToken - jest to token autoryzacyjny Firebase,
- refreshToken - token odświeżania tokenu identyfikatora,
- e-mail - adres e-mail uwierzytelnionego użytkownika,
- expiresIn - liczba sekund, w których wygasa token ID.

Do otrzymywania tych informacji konieczne jest posiadanie konta w aplikacji. Rejestracja polega na dodaniu użytkownika do bazy Firebase projektu. Następnie na podstawie informacji zawartych w zwróconym tokenie autoryzacji zostaną utworzone potrzebne relacje w bazie danych aplikacji.

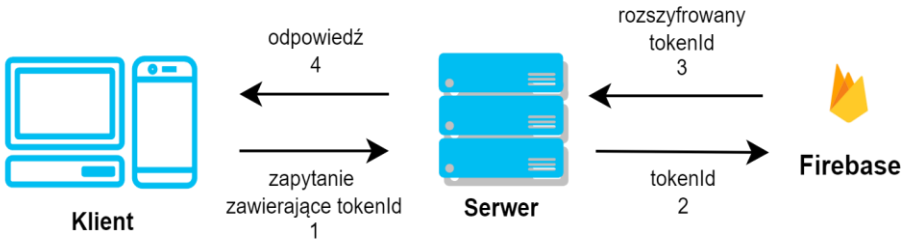
Sam Mechanizm wykorzystywany jest podczas logowania oraz każdorazowego odpytywania prywatnych endpointów. Schemat logowania został przedstawiony na rysunku 8.



Rysunek 8 Schemat przepływu pracy podczas logowania, źródło: opracowanie własne

Rozpoczyna się od przesłania przez klienta danych logowania bądź poświadczenia OAuth przez podłączonych do aplikacji dostawców tożsamości (w przypadku tej aplikacji jest to Google) do serwera Firebase (nr 1). W przypadku pozytywnej weryfikacji aplikacja otrzymuje odpowiedź zawierająca m.in. tokenId (nr 2). Kolejnym etapem jest połączenie z endpointem odpowiedzialnym za dostarczenie informacji o koncie, do którego zostaje przekazany token autoryzacyjny (nr 3). Praca na serwerze rozpoczyna się od sprawdzenia dostarczonego tokenu poprzez Spring Security. W tym celu konieczne jest jego rozszyfrowanie. Następuje to na serwerze Firebase, z racji, iż nie udostępnia on kluczy potrzebnych do jego dekrypcji (nr 4). Jeżeli token nie wygasł, zwrócone zostaną informacje o koncie w Firebase (nr 5), z których używany jest userId - jest on identyfikatorem konta w bazie aplikacji. Na jego podstawie zostaną pobrane, a następnie wysłane do klienta informacje o zalogowanym użytkowniku, w tym m.in. rola (nr 6).

Każdorazowe odpytywanie zabezpieczonych endpointów działa na podobnej zasadzie. Rozpoczyna się od połączenia, w którym oprócz wymaganych danych przesyłany jest również tokenId (nr 1). Tak jak w przypadku procesu logowania, token jest rozszyfrowany, a następnie sprawdzana jest jego ważność (nr 2 i 3). Gdy ów proces zakończy się powodzeniem, serwer weryfikuje uprawnienia użytkownika, a następnie zwraca odpowiednie informacje do klienta (nr 4).



Rysunek 9 Schemat przepływu pracy podczas odpytywania zabezpieczonych adresów, źródło: opracowanie własne

Ponadto Firebase wymaga HTTPS. Stanowi to dodatkową warstwę zabezpieczenia danych.

3.7. Dokumentacja API

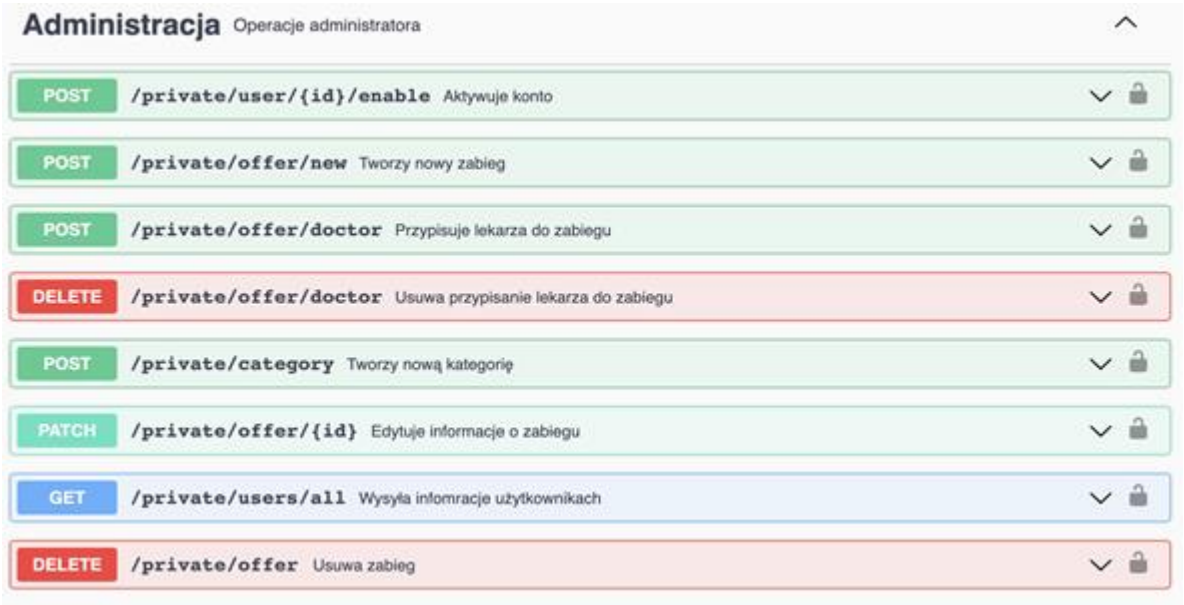
Serwer aplikacji opracowany został jako sieciowy interfejs RESTApi. Oczekuje on na żądania wysyłane przez klienta. Endpointy podzielono na grupy:

- Użytkownik (rysunek 10),
- Administracja (rysunek 11).
- Rezerwacje (rysunek 12),
- Oferta (rysunek 13),

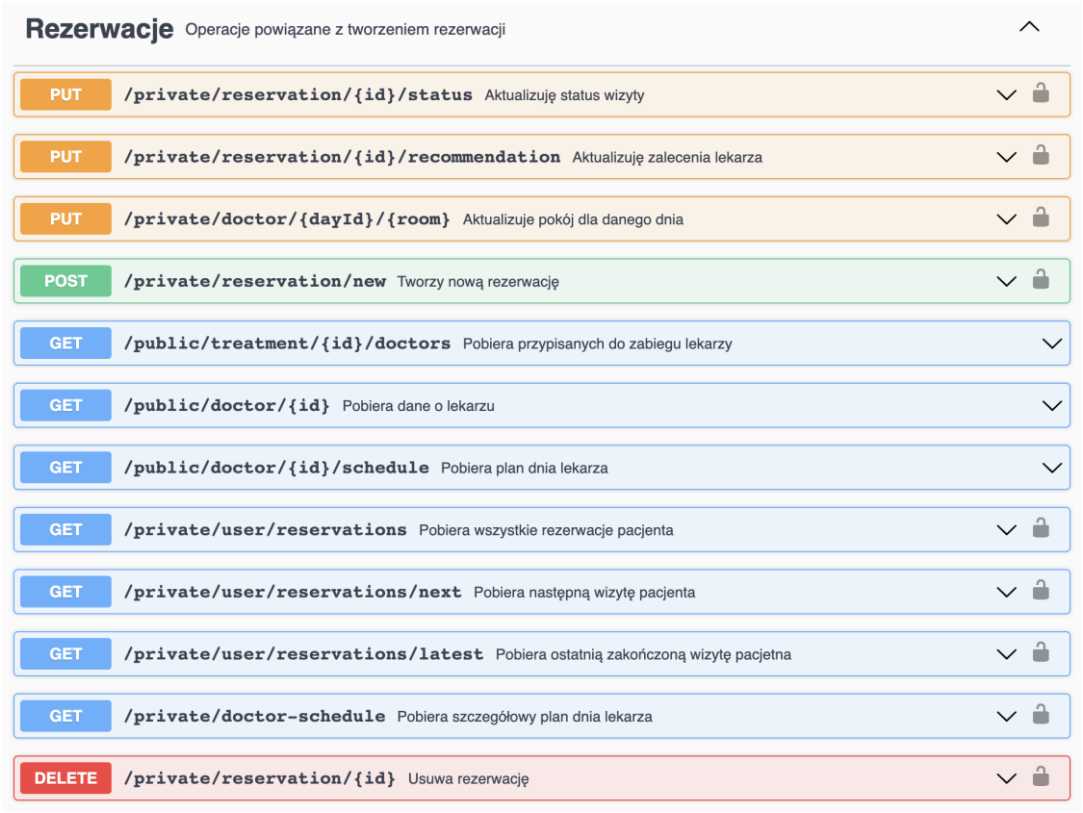
Każdy z nich odpowiada za obsługę innych części aplikacji. Ponadto wyróżnić można 2 rodzaje adresów sieciowych:

- rozpoczynające się od */private/* - wymagające autoryzacji przy użyciu odpowiedniego tokenu. Dodatkowo sprawdzane są uprawnienia przypisane do roli użytkownika,
- rozpoczynające się od */public/* - niewymagające dodatkowej autoryzacji.

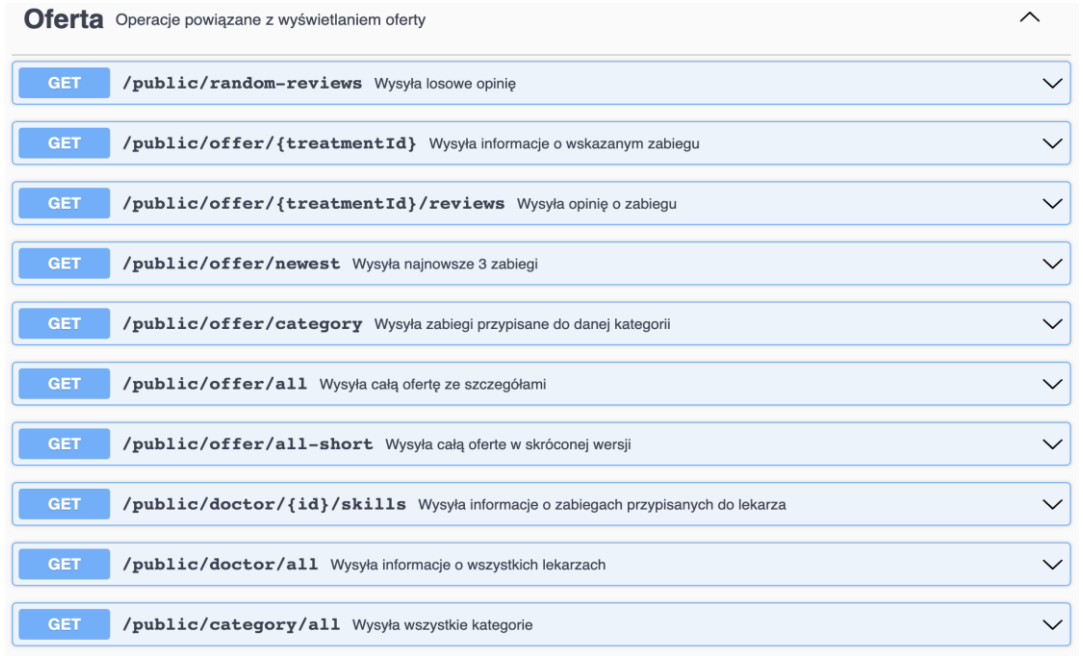
Rysunek 10 Interfejs API – użytkownik, źródło: opracowanie własne



Rysunek 11 Interfejs API – administracja, źródło: opracowanie własne



Rysunek 12 Interfejs API – rezerwacje, źródło: opracowanie własne



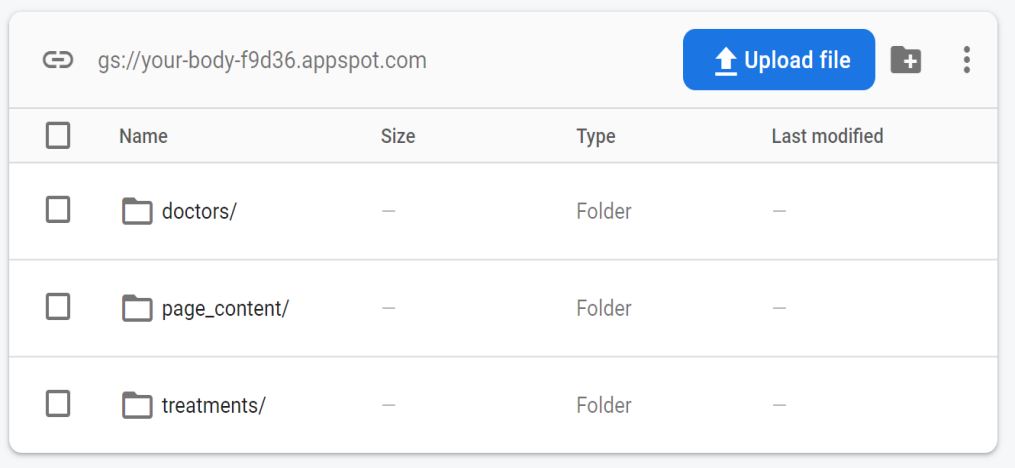
Rysunek 13 Interfejs API – Oferta, źródło: opracowanie własne

3.8. Przechowywanie plików

Podstawowym elementem gabinetu rehabilitacji jest jego oferta oraz lekarze. W celu ciekawszej ich prezentacji system pozwala na dodawanie opisów oraz zdjęć. Do przechowywania plików wykorzystany został moduł Firebase - Storage. Umożliwia on zapisywanie plików multimedialnych w bucketach Google Cloud Storage. Mając na uwadze różne użycie zdjęć, są one przetrzymywane w osobnych, tematycznych folderach. Dzięki temu cała struktura jest bardziej przejrzysta oraz łatwiejsza do utrzymania (rysunek 14).

Sam proces przesyłania zdjęć odbywa się w aplikacji klienckiej. Po wybraniu przez użytkownika zdjęcia następuje jego wysłanie do chmury. Dzięki podanym w funkcji atrybutom ma ono charakterystyczną nazwę oraz trafia do odpowiedniego folderu. Gdy operacja się zakończy, zwracany jest link URL do pliku, który następnie zostanie zapisany do bazy danych.

Dodatkowym atutem chmury jest możliwość jej zabezpieczenia przed nieautoryzowanym dostępem. W przypadku aplikacji gabinetu funkcja ta została wykorzystana do ograniczenia praw zapisu plików. Wymaga to bowiem odpowiedniej roli.



Rysunek 14 Struktura Firebase Storage, źródło: opracowanie własne

4. Implementacja interfejsu

W tym rozdziale przedstawione zostaną elementy implementacyjne klienta systemu do rezerwacji wizyt w gabinecie fizjoterapii. W pierwszym podrozdziale zostaną zaprezentowane widoki elementów strony internetowej wraz z opisem funkcji, jakie sprawują w całej strukturze. Natomiast w kolejnym zademonstrowane zostaną podstawowe scenariusze użycia aplikacji. Wszystkie rysunki prezentujące wygląd danych części interfejsu pochodzą z uruchomionej w przeglądarce strony internetowej aplikacji.

4.1. Interfejs użytkownika

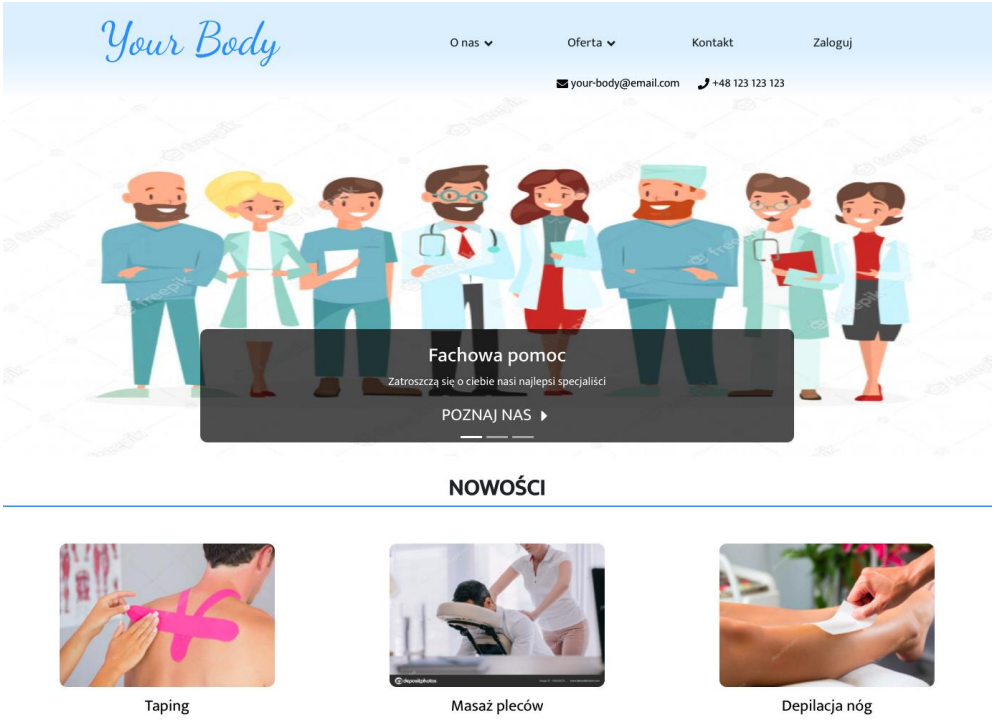
Interfejs strony umożliwiającej rezerwacje wizyt w gabinecie fizjoterapii składa się z dwóch części. Pierwszą z nich jest warstwa prezentacyjna. Jej głównym zadaniem jest przedstawienie oferty klientowi w taki sposób, aby zachęcić go do przejścia do drugiej warstwy. Składają się na nią:

1. Strona Główna,
2. „O nas”,
3. „Nasz zespół”,
4. „Oferta”,
5. „Cennik”,
6. „Kontakt”.

Głównym elementem drugiej części jest panel użytkownika. Dzięki niemu zalogowana do systemu osoba ma możliwość korzystania z elementów ukrytych dla gości. W zależności od roli przypisanej do konta zmienia się zawartość panelu oraz jego przeznaczenie.

4.1.1. Strona główna

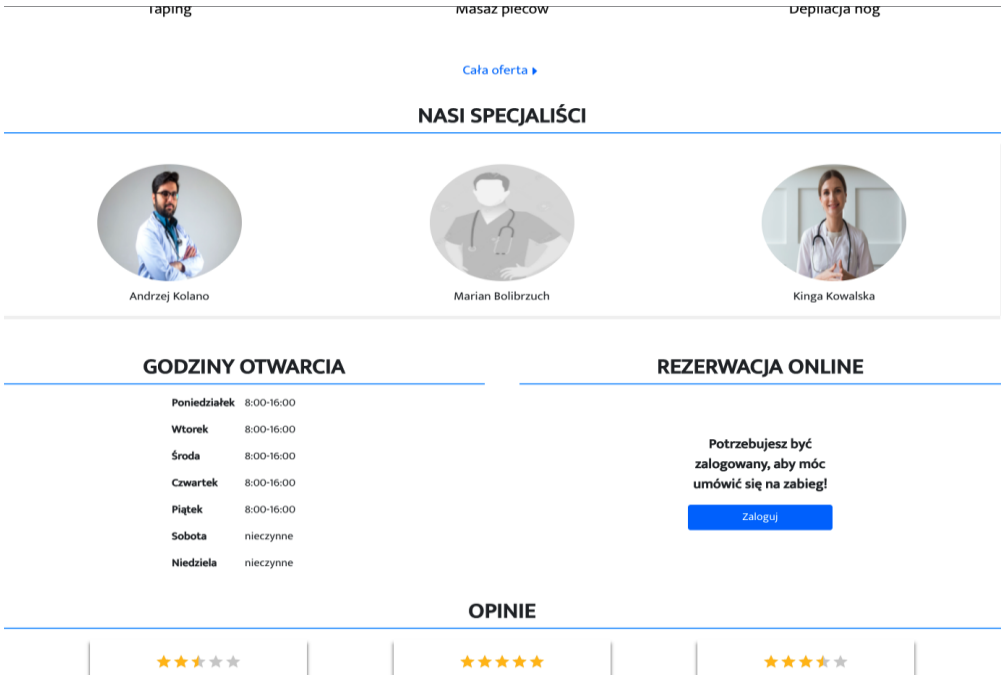
Stroną, którą widzi każdy odwiedzający, jest „Strona główna”. Zawiera ona streszczenie najważniejszych informacji, które mogą zainteresować użytkownika. Na przedstawionych rysunkach widać jej wygląd. Jest podzielona na 5 części, nie licząc nagłówka oraz stopki.



Rysunek 15 Widok Strony głównej cz. 1, źródło: opracowanie własne

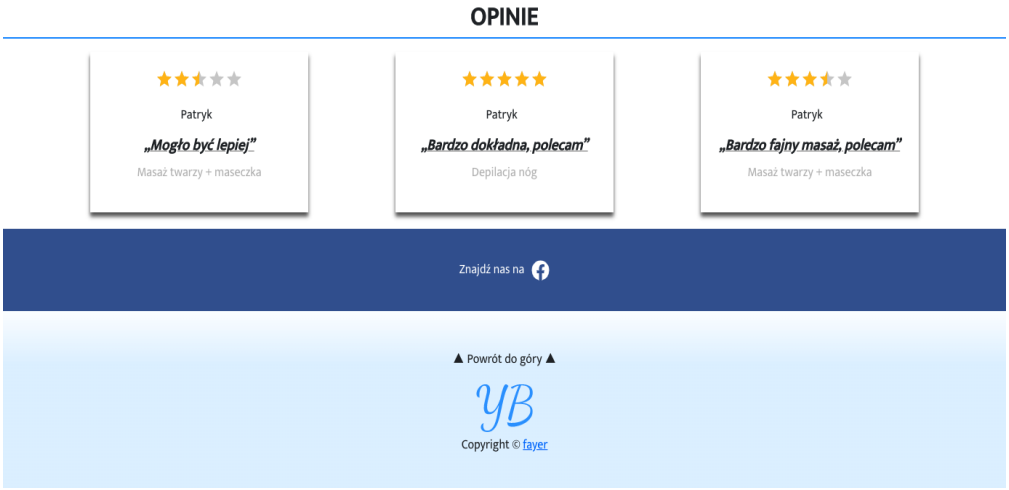
Pierwszą z nich jest slider, który zawiera odpowiedzi na najczęściej zadawane pytania. Ponadto ułatwia przechodzenie do konkretnych części strony poprzez naciśnięcie tytułu wraz z ikoną „▶”.

W kolejnym fragmencie, nazwanym „Nowości” użytkownik widzi trzy najnowsze zabiegi dodane do oferty. Każdy element zawiera zdjęcie zabiegu oraz jego nazwę. Po naciśnięciu którykolwiek z nich, użytkownik przenoszony jest do strony dotyczącej tego zabiegu. Dodatkowo widoczny na zdjęciu przycisk „Cała oferta” umożliwia przejście do części z ofertą.



Rysunek 16 Widok Strony głównej cz. 2, źródło: opracowanie własne

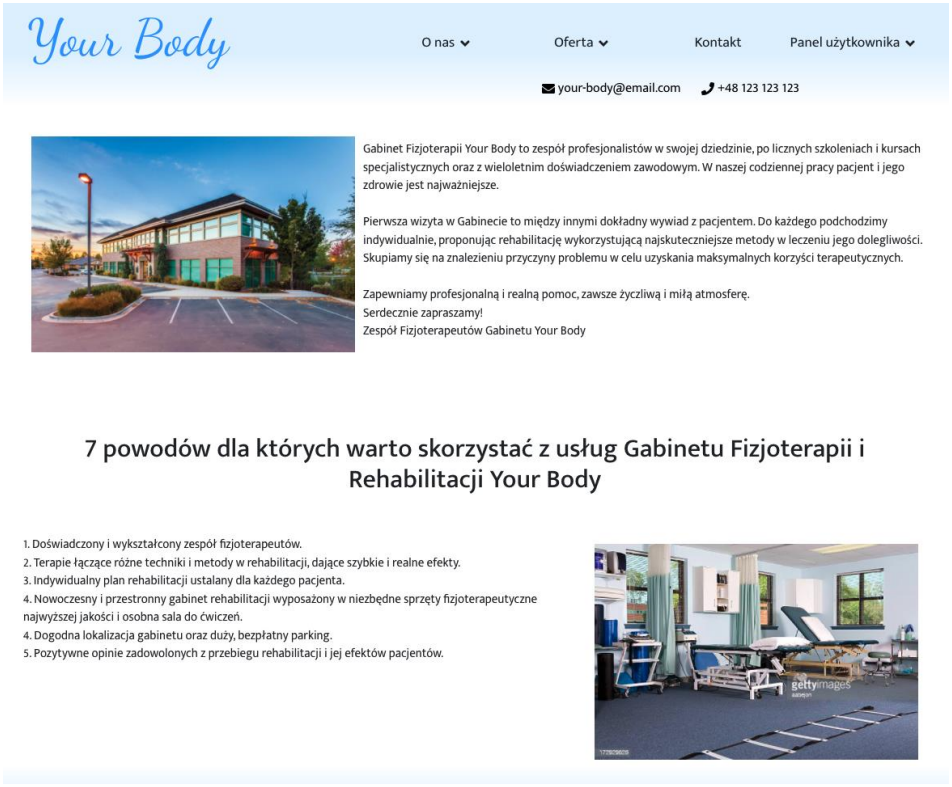
Kolejnymi dwoma sektorami są „Nasi specjaliści” oraz połączone w jedno „Godziny otwarcia” oraz „Rezerwacja online”. Pierwszy z nich ma za zadanie krótkie przedstawienie zespołu pracującego w gabinecie, używając zdjęć oraz imienia i nazwiska specjalisty. Kolejne części służą poinformowaniu o godzinach pracy oraz możliwości rejestracji online.



Rysunek 17 Widok Strony głównej cz. 3, źródło: opracowanie własne

Ostatnią częścią strony głównej są opinie. Prezentuje ona trzy losowo wybrane opinie pacjentów. Zawierają one informacje na temat oceny, komentarza oraz nazwie przeprowadzonego zabiegu.

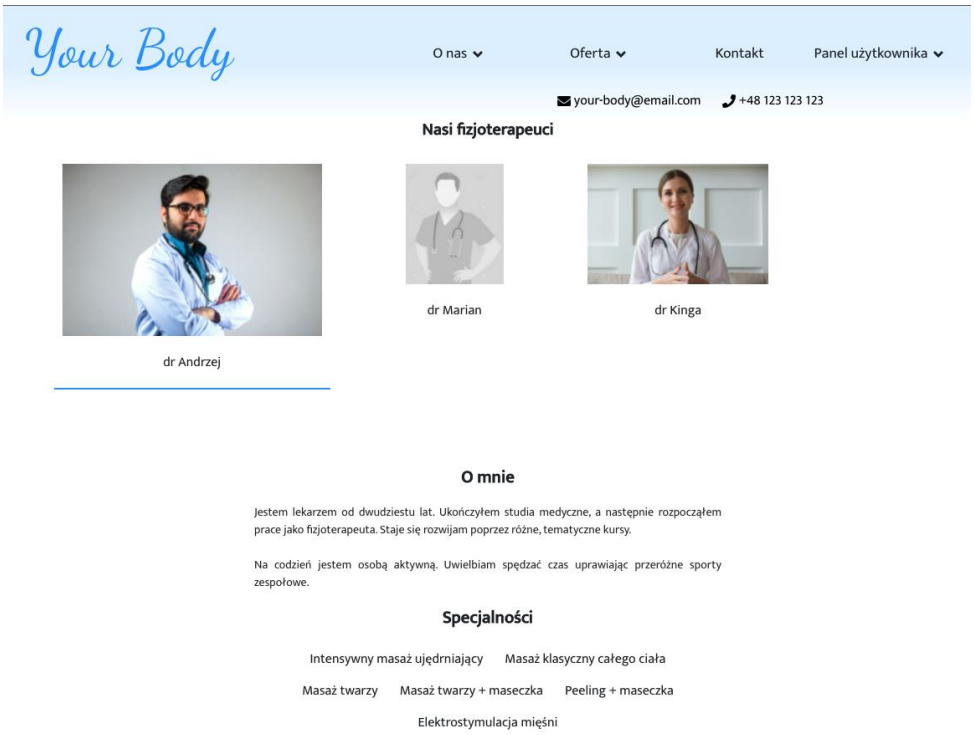
4.1.2. O nas



Rysunek 18 Widok strony „O nas", źródło: opracowanie własne

Zakładka o nazwie „O nas" przenosi użytkownika na stronę prezentującą gabinet. Można tam przeczytać najważniejsze informacje na temat pracowni, zobaczyć jak wygląda, jaką ma strategię rozwoju.

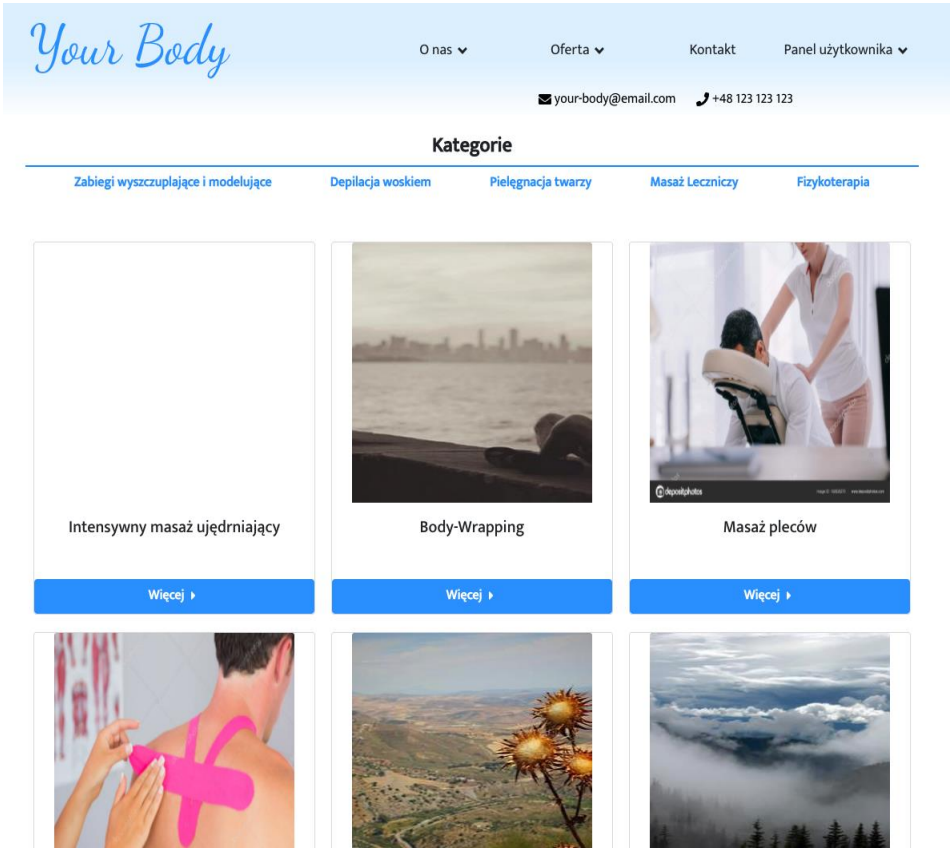
4.1.3. Nasz zespół



Rysunek 19 Widok strony „Nasz zespół", źródło: opracowanie własne

Jest to strona, której głównym zadaniem jest przybliżenie zespołu pracującego w gabinecie. Jak zostało ukazane na rysunku 19, użytkownik może wybierać pomiędzy wszystkim pracownikami („Nasi fizjoterapeuci"). Pod fotografią specjalisty, pacjent może zapoznać się z umiejętnościami lekarza, dzięki krótkiemu opisowi („O mnie"). Dodatkowo można zorientować się w wykonywanych zabiegach, na które możemy się zapisywać („Specjalności").

4.1.4. Oferta



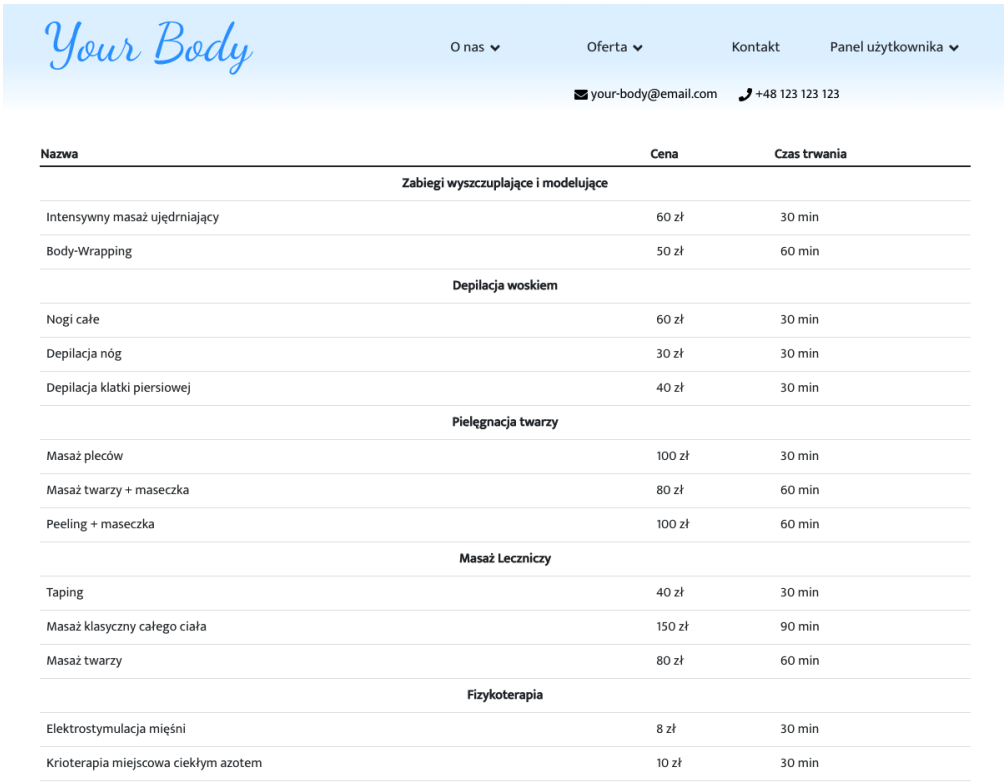
Rysunek 20 Widok strony „Oferta", źródło: opracowanie własne

Na rysunku 20 został ukazany wygląd strony przedstawiającej ofertę gabinetu. Każdy element zawiera zdjęcie zabiegu oraz jego nazwę. Dodatkowo przycisk umieszczony pod zabiegiem umożliwia przejście użytkownika do szczegółowych informacji o nim. Strona dodatkowo pozwala na filtrowanie zawartości przy użyciu kategorii. Naciśnięcie na interesującą kategorię, spowoduje wybranie pasujących zabiegów.

Po przejściu do konkretnego zabiegu użytkownik przenoszony jest na kolejną stronę, pokazaną na rysunku 21, na której może uzyskać bardziej szczegółowe informacjach na jego temat. Dodatkowo może zapoznać się z ogólną oceną pacjentów, w postaci gwiazdek oraz sprawdzić, do jakich specjalistów można się umówić.

Rysunek 21 Widok szczegółowej strony oferty, źródło: opracowanie własne

4.1.5. Cennik



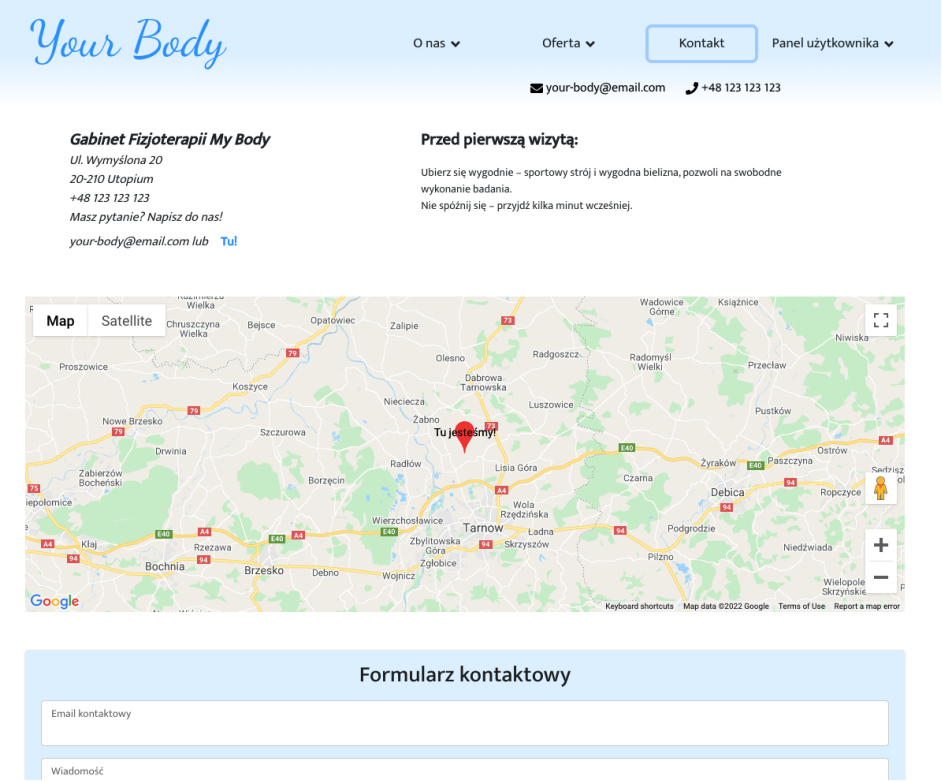
Nazwa	Cena	Czas trwania
Zabiegi wyszczuplające i modelujące		
Intensywny masaż ujędrniający	60 zł	30 min
Body-Wrapping	50 zł	60 min
Depilacja woskiem		
Nogi całe	60 zł	30 min
Depilacja nóg	30 zł	30 min
Depilacja klatki piersiowej	40 zł	30 min
Pielęgnacja twarzy		
Masaż pleców	100 zł	30 min
Masaż twarzy + maseczka	80 zł	60 min
Peeling + maseczka	100 zł	60 min
Masaż Leczniczy		
Taping	40 zł	30 min
Masaż klasyczny całego ciała	150 zł	90 min
Masaż twarzy	80 zł	60 min
Fizykoterapia		
Elektrostymulacja mięśni	8 zł	30 min
Krioterapia miejscowa ciekłym azotem	10 zł	30 min

Rysunek 22 Wygląd strony „Cennik", źródło: opracowanie własne

Cennik jest podstroną, której głównym zadaniem jest zestawienie kosztów oraz czasu trwania zabiegów. Jak pokazano na rysunku 22 strona ma postać tabeli, w której

zabiegi posortowane są według kategorii. Każdy wiersz zawiera jego nazwę, cenę oraz czas trwania.

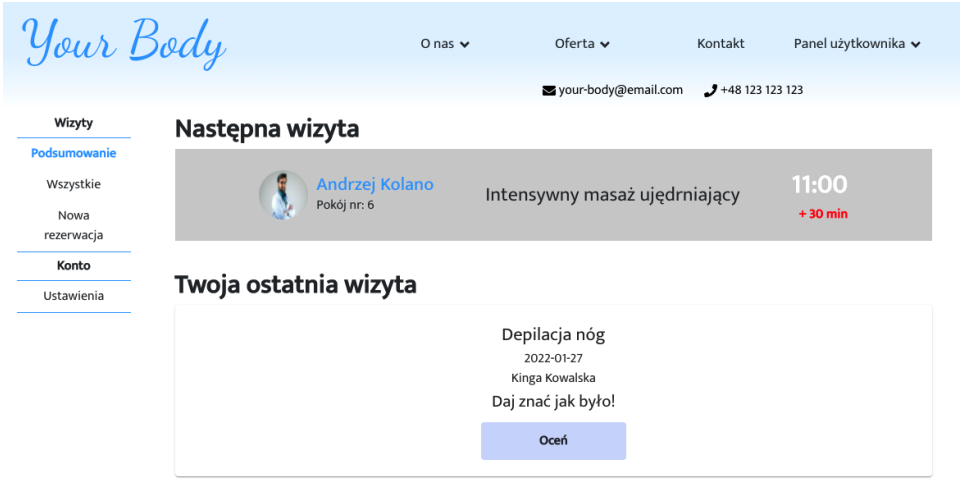
4.1.6. Kontakt



Rysunek 23 Widok strony „Kontakt", źródło: opracowanie własne

Głównym zadaniem tej podstrony jest zaprezentowanie danych kontaktowych. Na zaprezentowanym rysunku 23 wyróżnić można 3 elementy: Informacje o pierwszej wizycie, dane kontaktowe, interaktywną mapę Google pokazującą lokalizację gabinetu oraz formularz kontaktowy. Dodatkowo użytkownik ma możliwość kontaktu poprzez znajdujące się w nagłówku przyciski zawierające e-mail oraz numer telefonu.

4.1.7. Panel użytkownika



Rysunek 24 Widok panelu użytkownika jako pacjent, źródło: opracowanie własne

Cała warstwa funkcyjna zawarta jest w zakładce „Panel użytkownika”. Dostęp do niej ograniczony jest tylko dla osób zalogowanych na wcześniej utworzone konto. Jej zawartość zależy od przypisanej do użytkownika roli.

W tej części pacjent może zarządzać własnymi wizytami, edytować informacje o koncie oraz dodawać kolejne rezerwacje. Każda funkcja znajduję się w osobnych kartach nawigacji umieszczonej w lewej części strony.

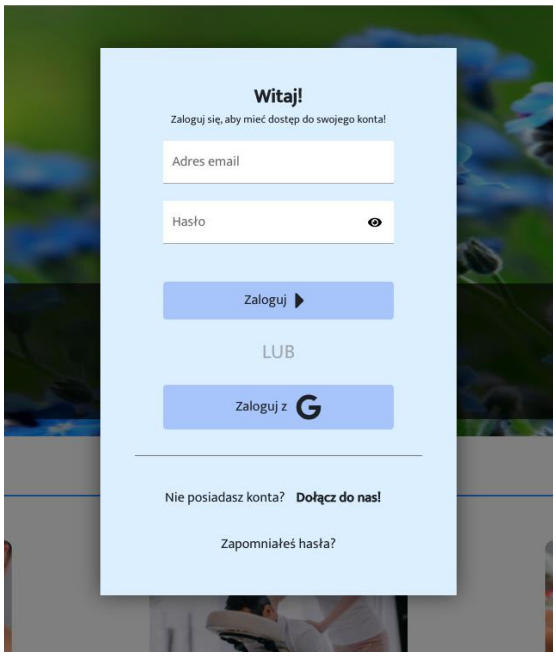
Z kolei lekarz wykorzystując tę zakładkę ma dostęp do rozkładu dnia, w którym znaleźć można zaplanowane na ten termin wizyty. Dodatkowo ma też wgląd do historii rezerwacji, która również jest edytowalna.

Ostatnim możliwym typem użytkownika jest Administrator. Interfejs administratora pozwala na zarządzanie treścią strony. Ma możliwość dodawania nowych elementów do oferty oraz kontroli kont pacjentów bądź lekarzy.

4.2. Podstawowe scenariusze użycia

Ukazane w tym rozdziale scenariusze, zostaną zaprezentowane przy użyciu widoków poszczególnych elementów strony internetowej oraz kont posiadających różne role systemu.

4.2.1. Rejestracja

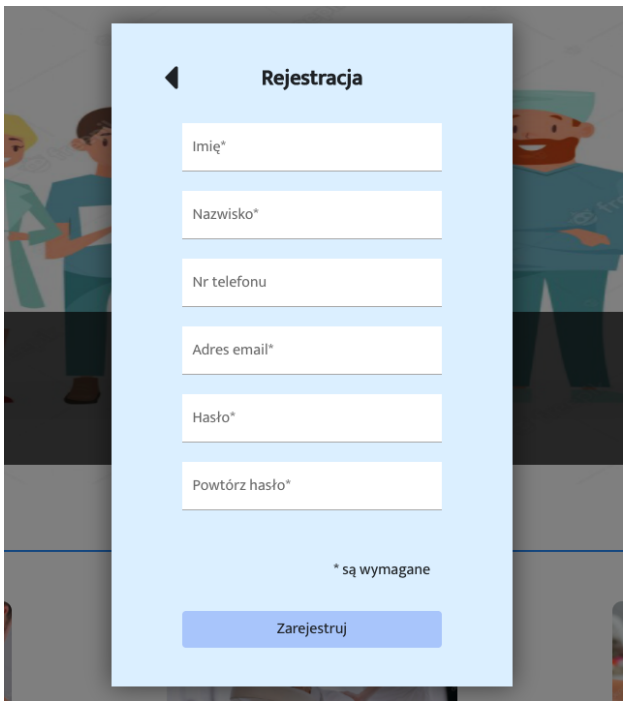


Rysunek 25 Widok okna logowania, źródło: opracowanie własne

Procesem, od którego zaczyna każdy użytkownik chcący w pełni korzystać z aplikacji, jest rejestracja. Aplikacja umożliwia tworzenie konta pacjenta. Pierwszym krokiem po otworzeniu strony jest przejście do zakładki Logowania. Użytkownik chcący otrzymać pełny dostęp do strony, może założyć konto, korzystając z dwóch sposobów:

1) przy użyciu adresu e-mail oraz hasła – „Dołącz do nas!”
2) przy użyciu konta Google „Zaloguj z Google”

Używając pierwszej opcji, użytkownik przenoszony jest do formularza, w którym proszony jest o wypełnienie wszystkich potrzebnych do założenia konta informacji. Po zakończeniu oraz naciśnięciu przycisku rejestracji, dane są walidowane, aby następnie założyć konto bądź poprosić o poprawienie formularza.



Rysunek 26 Widok okna rejestracji, źródło: opracowanie własne

Druga opcja jest prostsza w użycia, gdyż po naciśnięciu przycisku, korzystający z aplikacji zostaje poproszony o zalogowanie do swojego konta Google. Następnie wszystkie potrzebne informacje zostają automatycznie przepisane z serwera Google.

Po udanej rejestracji użytkownik zostaje automatycznie zalogowany do aplikacji.

4.2.2. Logowanie

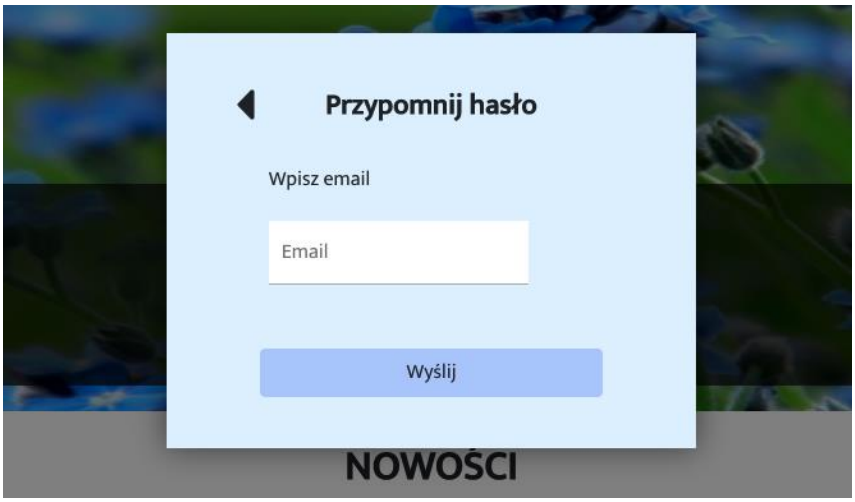
Każdy użytkownik posiadający konto w aplikacji może użyć opcji logowania, aby otrzymać dostęp do ukrytych elementów strony. Aby rozpocząć cały proces, użytkownik powinien odnaleźć na panelu nawigacji opcję „Zaloguj”. Zostanie wówczas otworzone okno logowania, identyczne z tym, które używane jest w procesie rejestracji. Tak jak w przypadku rejestracji, można zalogować się do aplikacji również na dwa sposoby:

1. przy użyciu adresu e-mail oraz hasła,
2. przy użyciu konta Google.

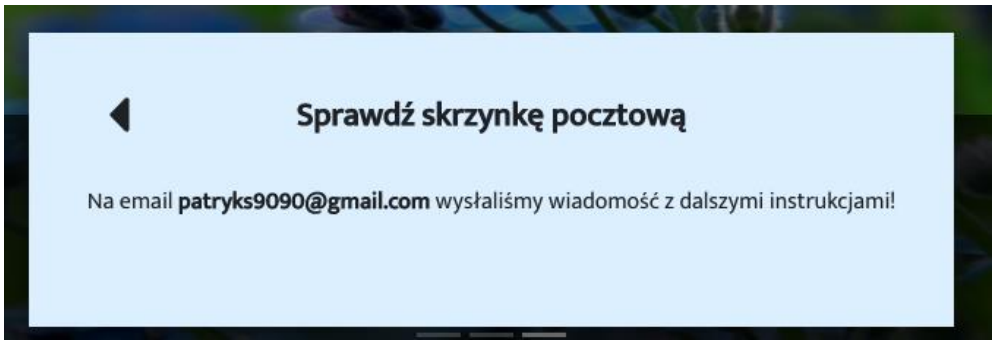
Chcąc skorzystać z danych podanych podczas rejestracji należy je wpisać w pokazanym na rysunku 25 formularzu, a następnie zatwierdzić. Logowanie kontem Google wymaga posiadanie konta na owej stronie. Po kliknięciu w przycisk „Zaloguj z Google” otwierane jest nowe okno, w którym następuje logowanie. Po zatwierdzeniu, system sprawdza dane oraz udziela bądź odmawia dostępu.

4.2.3. Przypominanie lub zmiana hasła

Hasło jest wymaganym elementem do logowania się w aplikacji. Bardzo często spotykaną sytuacją jest zapomnienie hasła do utworzonego konta. Użytkownikowi udostępniana jest możliwość jego zresetowania. Aby rozpocząć proces zmiany hasła należy przejść do okna logowania poprzez zakładkę „Zaloguj". Następnie w wyświetlonym oknie odnaleźć opcję „Zapomniałeś hasła?". Po naciśnięciu na nią zawartość okna zmieni się na formularz z miejscem na wpisanie adresu e-mail konta użytkownika. Po jego wypełnieniu i zatwierdzeniu, na podany adres e-mail zostanie wysłana wiadomość z linkiem do strony służącej do ustawienia nowego hasła.



Rysunek 27 Widok okna przypominania hasła, źródło: opracowane własne



Rysunek 28 Widok okna potwierdzenia wysłania wiadomości, źródło: opracowanie własne

Zmianę hasła można przeprowadzić po zalogowaniu się na własne konto. Opcja ta znajduje się w „Panelu użytkownika". Na pasku nawigacji należy odnaleźć zakładkę o takiej nazwie. Kolejnym krokiem jest przejście do ustawień konta. Widoczne będą różne formularze. Jednym z nich będzie formularz „Zmiana hasła". Po jego wypełnieniu oraz zatwierdzeniu, hasło zostanie zaktualizowane.

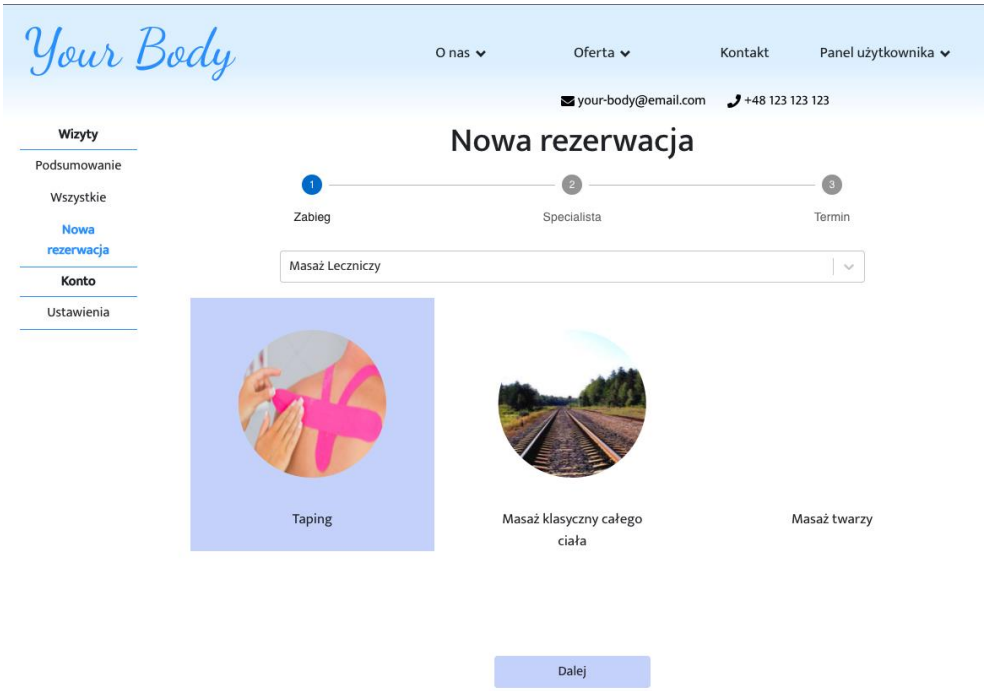
Rysunek 29 Widok okna zmiany hasła w panelu użytkownika, źródło: opracowanie własne

4.2.4. Umawianie na wizyty

Możliwość umawiania na wizyty jest najważniejszą funkcją w całym systemie. Aby rozpocząć tę operację, należy zalogować się na konto pacjenta. Do kolejnego kroku można przejść na dwa sposoby:

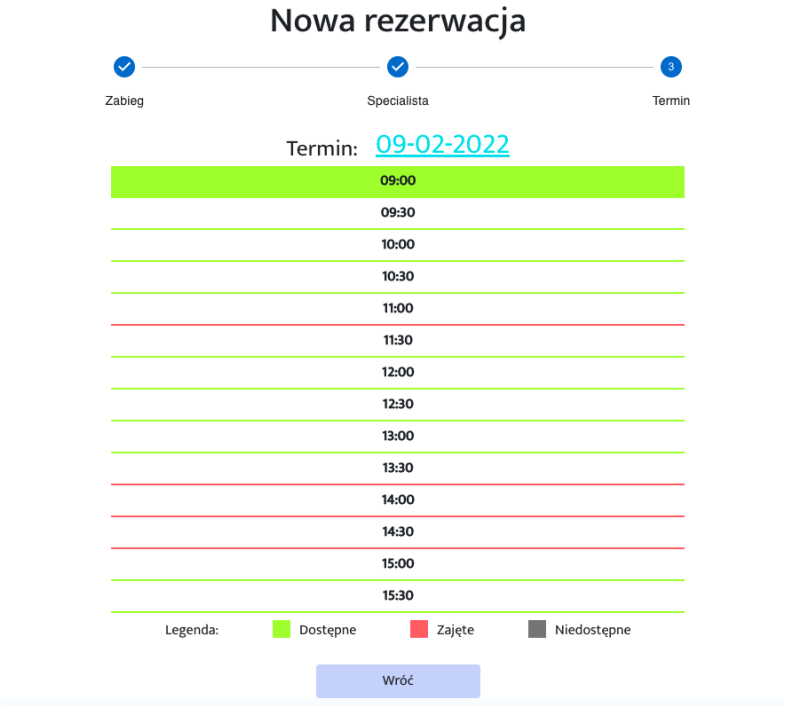
- pierwszym z nich jest przycisk znajdujący się na stronie głównej w sekcji „Rezerwacja Online”. Po naciśnięciu na niego zostaniemy przeniesieni do miejsca tworzenia rezerwacji,
- drugim sposobem jest przejście do panelu użytkownika, znajdującego się w nawigacji, a następnie otworzenie karty „Nowa rezerwacja”.

Dalszy proces wygląda tak samo w obu przypadkach. Tworzenie rezerwacji podzielone jest na 3 etapy: wybór zabiegu, wybór specjalisty oraz wybór wolnego terminu.



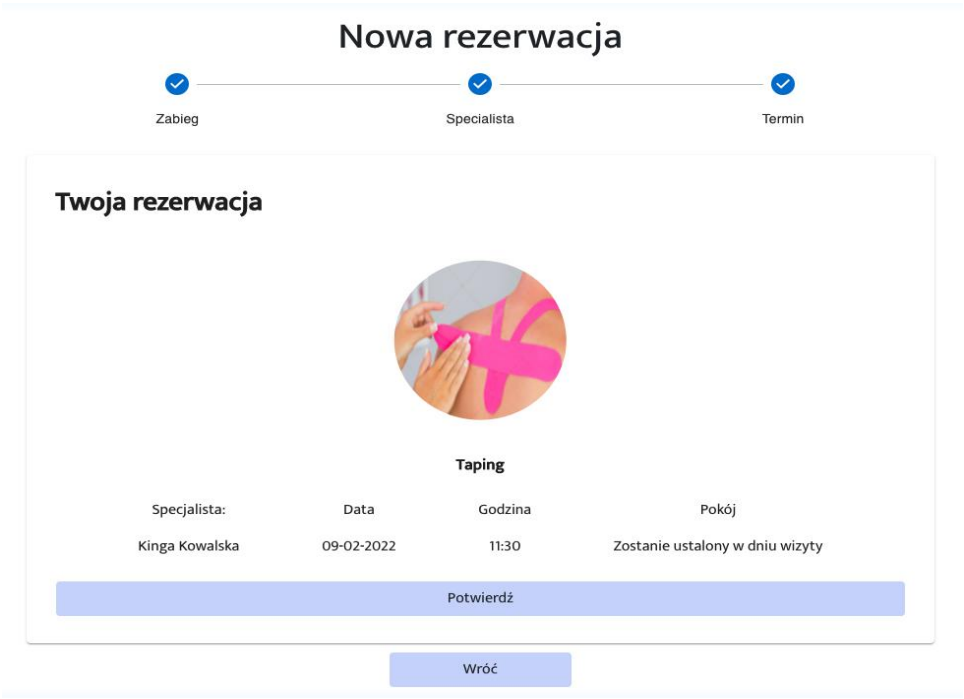
Rysunek 30 Widok strony tworzenia rezerwacji cz. 1, źródło: opracowanie własne

Przystępując do pierwszego z nich, należy wybrać kategorię, a następnie dany zabieg (rysunek 30). Po przejściu dalej wyświetleni zostaną specjaliści do niego przypisani. Należy zaznaczyć jednego z nich. Do kolejnego kroku pacjent zostanie automatycznie przeniesiony.



Rysunek 31 Widok strony tworzenia rezerwacji cz. 2, źródło: opracowanie własne

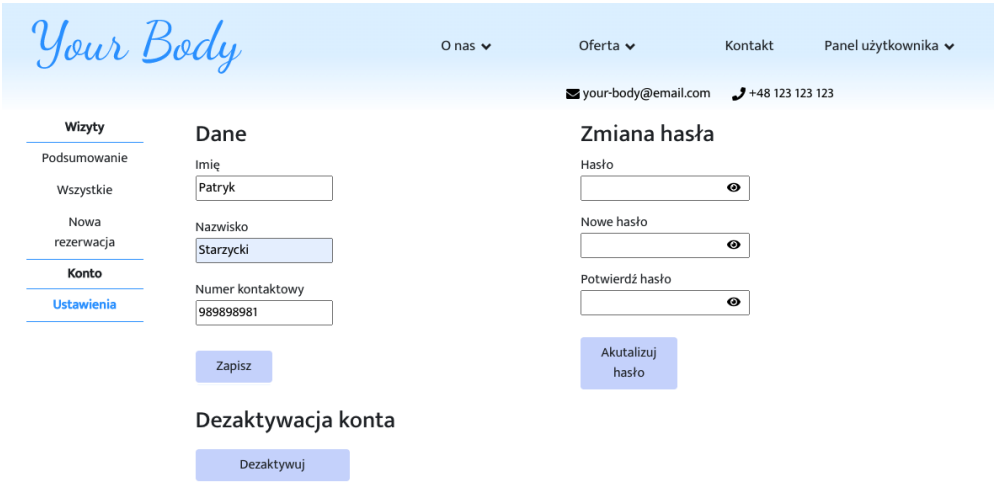
W następnym etapie wyświetlony zostanie plan dnia, który zaprezentowano na rysunku 31. Użytkownik ma możliwość wyboru interesującej go daty oraz wyboru dostępnej godziny. Następnie zostanie przekierowany do strony podsumowującej rezerwacje (rysunek 32). Wówczas należy ostatecznie zadecydować o chęci rezerwacji oraz ją zatwierdzić.



Rysunek 32 Widok okna potwierdzenia tworzonej rezerwacji, źródło: opracowanie własne

4.2.5. Edycja konta

Użytkownik posiadający konto ma możliwość zarządzania informacjami, które podał podczas jego tworzenia. W tym celu konieczne jest zalogowanie się do niego. Następnie należy przejść do panelu użytkownika, gdzie do wyboru mamy kartę „Ustawienia” w kategorii „Konto”. Po jej otworzeniu wyświetlona zostanie strona zawierająca 3 elementy, jak zaprezentowano na rysunku 33.

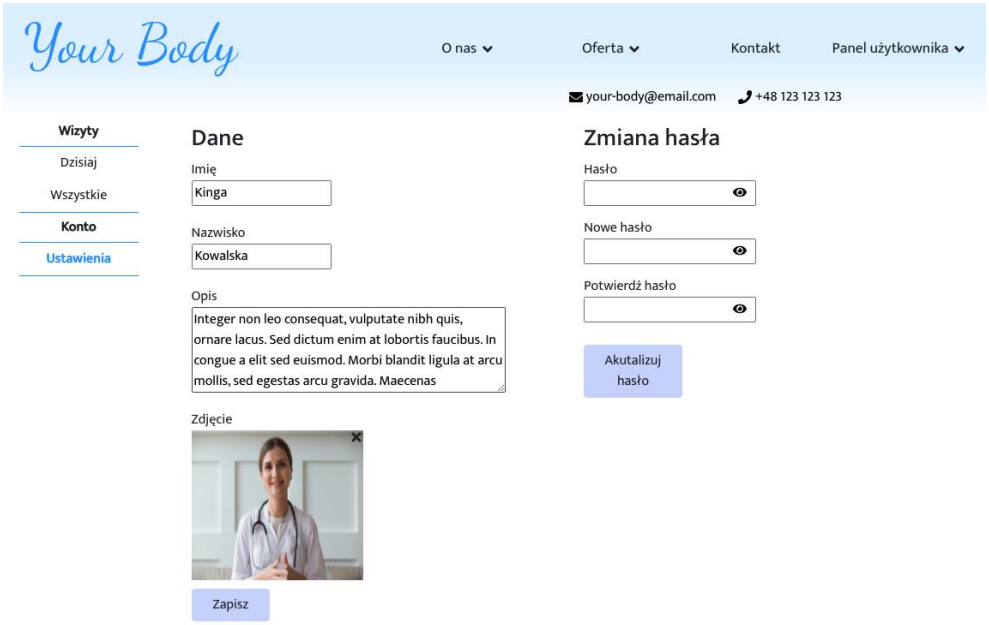


Rysunek 33 Widok karty „Ustawienia" w panelu pacjenta, źródło: opracowanie własne

Są to:

1. Dane,
2. Zmiana hasła,
3. Dezaktywacja konta.

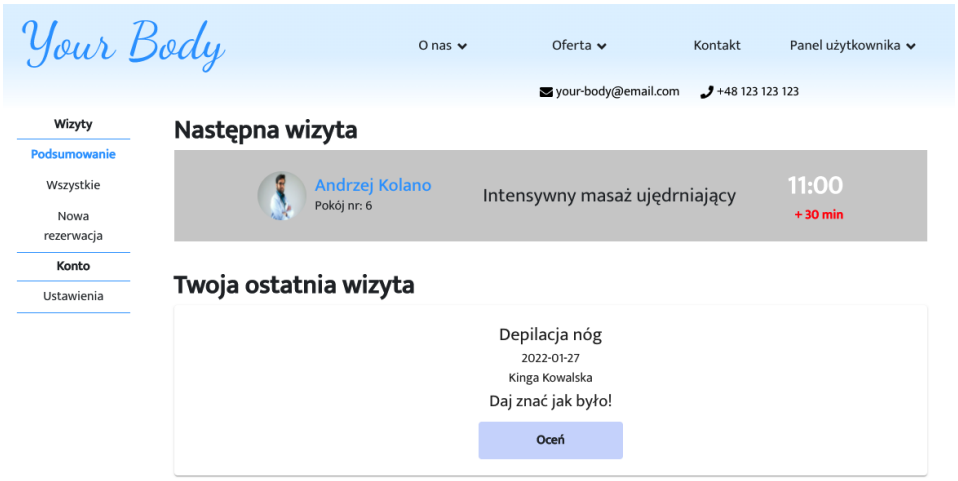
Aby edytować dane osobiste tj. imię, nazwisko bądź numer kontaktowy, należy uzupełnić formularz pod częścią „Dane", a następnie zatwierdzić zmiany przyciskiem „Zapisz". Po udanej edycji zostanie wyświetlona informacja. Kolejną możliwą operacją jest zmiana hasła. W tym celu należy uzupełnić odpowiedni formularz, wpisując obecne hasło, oraz dwa razy nowe hasło. Po zweryfikowaniu wprowadzonych danych hasło zostanie zaktualizowane. Ostatnią częścią jest „Dezaktywacja konta". Służy ona do zablokowania dostępu do strony przy użyciu tego konta. Po naciśnięciu odpowiedniego przycisku użytkownik zostanie zapytany o potwierdzenie swojej decyzji z racji, iż wiąże się to z wylogowaniem oraz utratą dostępu do strony.



Rysunek 34 Widok karty „Ustawienia" w panelu lekarza, źródło: opracowanie własne

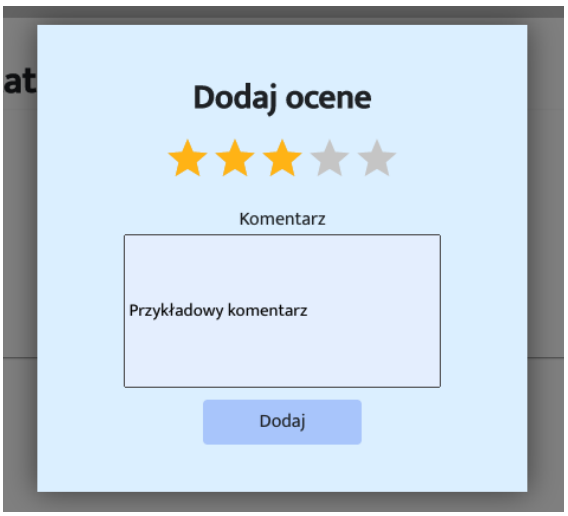
4.2.6. Dodawanie oceny

Pacjent po zakończonej wizycie ma możliwość oceny jej przebiegu. Aby do tego przystąpić, konieczne jest, aby użytkownik był zalogowany na własne konto, aby mieć dostęp do panelu użytkownika. Na przedstawionym rysunku 35 widoczny jest widok tej części. Zawiera on informacje o ostatniej oraz następnej wizycie.



Rysunek 35 Widok karty podsumowującej wizyty pacjenta, źródło: opracowanie własne

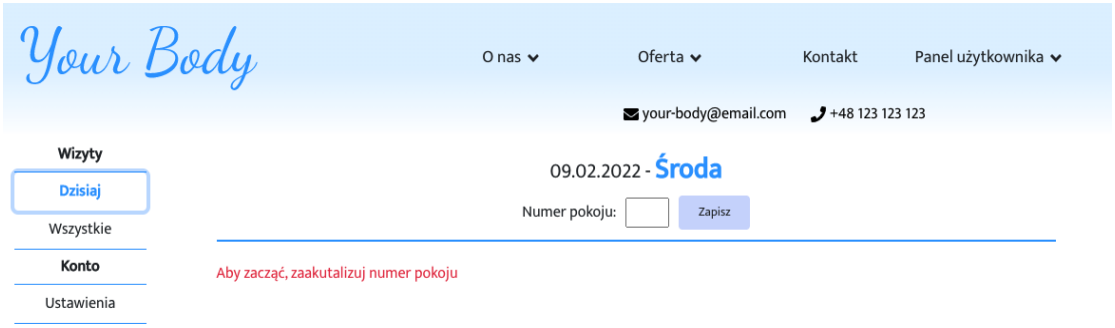
Odnajdując okno z interesującym użytkownika zabiegiem, zobaczy on przycisk o nazwie „Oceń". Po jego naciśnięciu pojawi się okno odpowiedzialne za dodania opinii (rysunek 36). Po wprowadzeniu komentarza oraz zaznaczeniu oceny należy użyć przycisku „Dodaj" w celu zatwierdzenia operacji.



Rysunek 36 Widok okna dodawania oceny, źródło: opracowanie własne

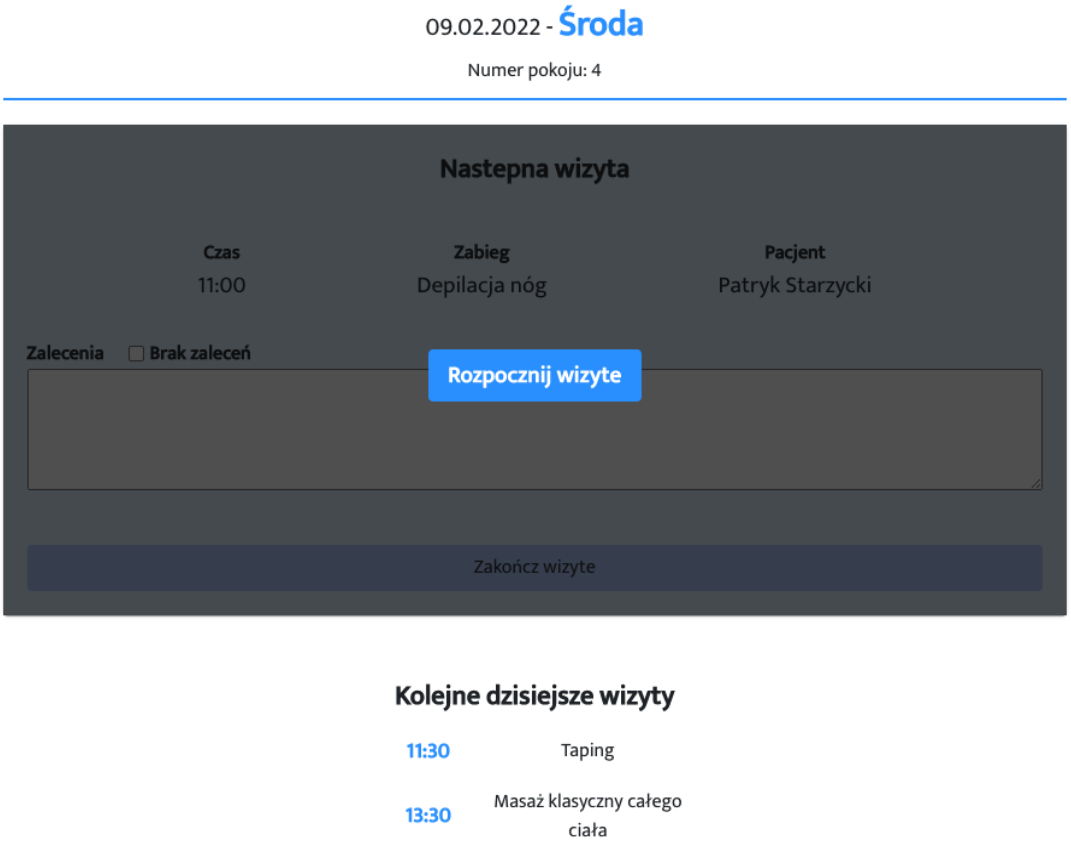
4.2.7. Przebieg wizyty

Oprócz samej rezerwacji wizyt system pozwala na aktualizację jego statusu oraz zaleceń. W tym celu lekarz powinien zalogować się na swoje konto, a następnie przejść do panelu użytkownika. Wówczas jako pierwsza zostanie wyświetlona karta „Dzisiaj", jak widać na rysunku 37. W niej odbywają się wszystkie operacje mające na celu przeprowadzenie wizyty.



Rysunek 37 Widok karty „Dzisiaj" w panelu lekarza, źródło: opracowanie własne

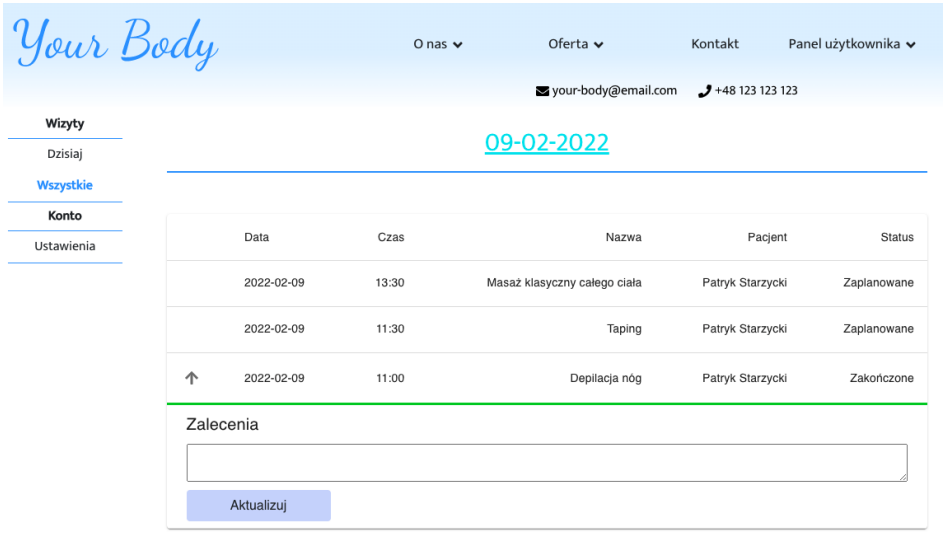
Aby rozpocząć dany dzień, system wymaga aktualizacji numeru pokoju, w którym tego dnia przebywa lekarz. Ma to ułatwić pacjentom znalezienie odpowiedniego pomieszczenia. Po tej operacji wyświetlona zostanie najbliższa wizyta, a pod nią plan dnia, jak zaprezentowano na rysunku 38. Okno wizyty pozwala na jej rozpoczęcie oraz zakończenie z dodanymi zaleceniami, jak również bez.



Rysunek 38 Widok okna wizyt w panelu lekarza, źródło: opracowanie własne

4.2.8. Aktualizacja zaleceń lekarza

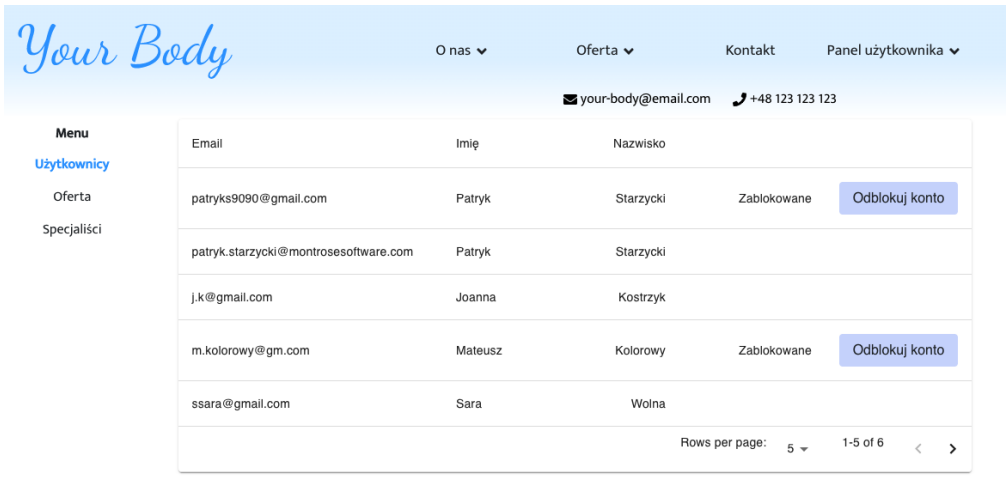
Po zakończonej wizycie lekarz ma możliwość wglądu w historię wizyt danego dnia. Widok tego interfejsu ukazany został na rysunku 39. Znajduje się on w panelu użytkownika. Ma on formę tabeli, co ułatwia przeglądanie. Służy on głównie do aktualizacji zaleceń danej rezerwacji. W tym celu lekarz powinien odnaleźć interesującą go wizytę. Poprzez widoczną po lewej stronie wiersza strzałkę, ma on możliwość sprawdzenia ukrytej części, zaprojektowanej do aktualizacji zaleceń. Po wypełnieniu przeznaczonego na to pola oraz naciśnięciu „Aktualizuj” operacja zostaje wykonana.



Rysunek 39 Widok karty do zarządzania wszystkimi wizytami, źródło: opracowanie własne

4.2.9. Odblokowanie konta pacjenta

W celu odblokowania konta pacjenta należy w pierwszym kroku zalogować się na konto z odpowiednimi uprawnieniami. Następnie administrator powinien przejść do panelu użytkownika, znajdującym się w nagłówku strony, skąd ma możliwość zarządzania zawartością strony. Wówczas jako pierwsza zostanie otworzona strona widoczna na rysunku 40.

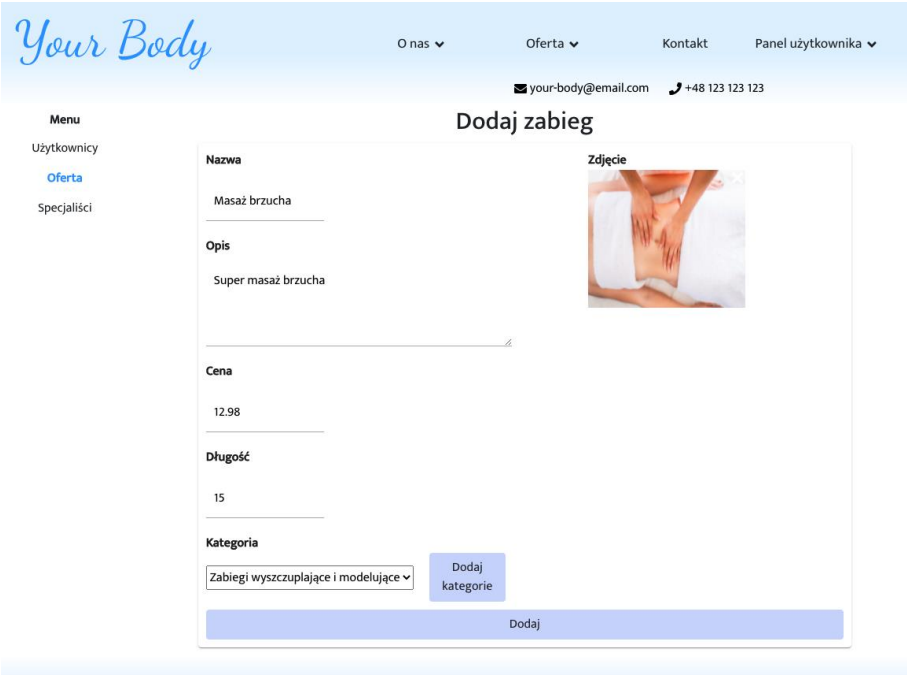


Rysunek 40 Widok karty do zarządzania użytkownikami w panelu administratora, źródło: opracowanie własne

Strona ma formę tabeli zawierającej konta wszystkich pacjentów zarejestrowanych do aplikacji. W sytuacji, gdy konto użytkownika jest zablokowane, administrator ma możliwość jego odblokowania. Aby to uczynić, należy po znalezieniu odpowiedniego wiersza, nacisnąć przycisk „Odblokuj konto”. Po udanej operacji zostanie wyświetlona informacja.

4.2.10. Dodawanie oferty

Mając na celu dodanie zabiegu bądź kategorii, administrator powinien rozpocząć od zalogowania się na własne konto. Kolejno winien przejść do panelu użytkownika poprzez odpowiednią zakładkę. Będąc w panelu administratora, należy zmienić wyświetlaną kartę na „Oferta". Wówczas wyświetlony zostanie formularz pozwalający na dodawanie nowych zabiegów. Po wypełnieniu każdego pola oraz wybraniu zdjęcia, możliwe stanie się jego zatwierdzenie jak pokazano na rysunku 41.



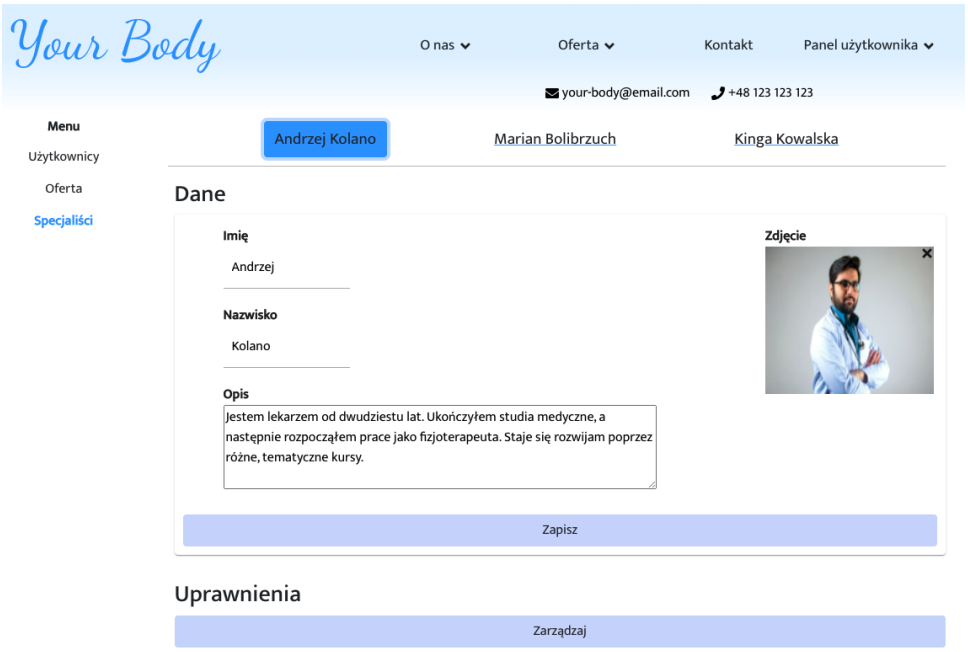
Rysunek 41 Widok karty do dodawania oferty lub kategorii w panelu administratora, źródło: opracowanie własne

W ramach oferty można również dodać nową kategorię zabiegów. Należy w opisany wyżej sposób przejść do karty „Oferta" w panelu użytkownika. Następnie poprzez przycisk „Dodaj kategorie" otworzyć okno zawierające pole do wpisania nazwy oraz przycisk do potwierdzenia dodania pokazany w rysunku 42.

Rysunek 42 Widok okna do dodawania nowej kategorii, źródło: opracowanie własne

4.2.11. Zarządzanie lekarzami

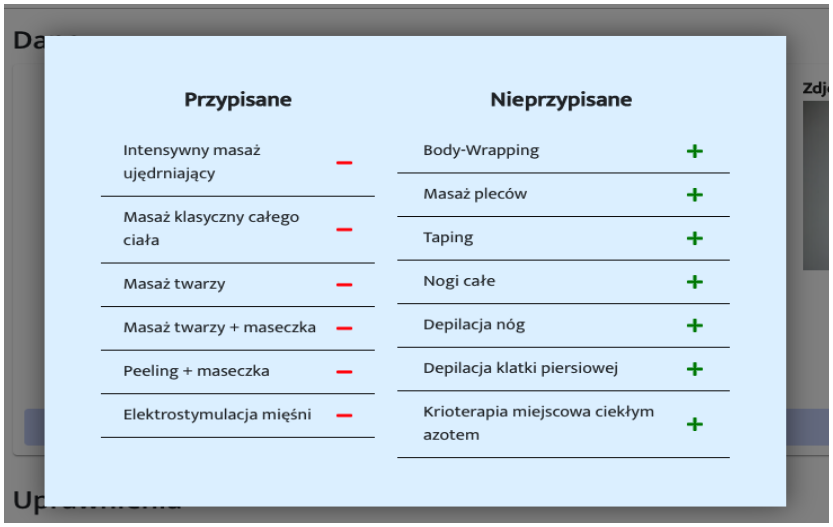
System dodatkowo zapewnia administratorowi możliwość zarządzania lekarzami. W tym celu należy zalogować się na jego konto, przejść do panelu użytkownika, a następnie w menu po lewej stronie wybrać kartę „Specjaliści". Wyświetlony zostanie interfejs widoczny na rysunku 43.



Rysunek 43 Widok karty do zarządzania lekarzami w panelu administratora, źródło: opracowanie własne

W tej karcie możliwy jest wybór lekarza, którego dane administrator chciałby zaktualizować. W tym celu powinien on zaznaczyć interesującą go osobę, a następnie zmienić interesujące go dane. Po naciśnięciu „Zapisz" zostaną zapisane. Dodatkowo system umożliwia zmianę uprawnień lekarza, tj. zmianę przypisanych do niego zabiegów. Służy do tego część podpisana „Uprawnienia". Po naciśnięciu „Zarządzaj", wyświetlone zostanie okno pokazane na rysunku 44.

Przedstawia ono zabiegi przypisane oraz nieprzypisane do danego specjalisty w postaci dwóch kolumn. W celu dodania bądź usunięcia przypisania należy nacisnąć znak „+" lub „-" znajdujący się przy zabiegu.



Rysunek 44 Widok okna do zarządzania zabiegami lekarza, źródło: opracowanie własne

5. Testy

Testy są częścią kodu (przeważnie metodą), które wywołują inną część kodu oraz sprawdzają, czy końcowy wynik jest zgodny z założeniami. Jeżeli nie, test kończy się niepowodzeniem. [15]

Mając na celu zapewnienie niezawodności funkcjonowania systemu, napisane zostały testy sprawdzające poszczególne elementy. Pośród nich wyróżnić możemy:

- testy jednostkowe - mające na celu sprawdzenie konkretnych, pojedynczych części systemu np. jedna metoda lub klasa,
- testy integracyjne - ich zadaniem jest sprawdzenie części składających się z kilku zależnych od siebie elementów np. proces zapisu do bazy danych.

5.1. Narzędzia

5.1.1. JUnit

JUnit jest frameworkiem, który zapewnia szkielet służący do pisania testów jednostkowych. Na jego charakterystyczne cechy składa się wiele sposobów uruchamiania testów, oddzielenie kodu testów od kodu aplikacji, możliwość tworzenia raportów. Dodatkowo w przypadku języka Java, dodaje on adnotacje, które z góry ustalają, jak dany element powinien się zachować. Zaliczyć do nich możemy np.:

- @Test - która wskazuje, że dana metoda jest metodą testową,
- @BeforeEach - wskazuję, że metoda zostanie uruchomiona przed każdym testem,
- @AfterEach – wskazuję, że metoda zostanie uruchomiona po każdym teście..

W celu weryfikacji wyniku, JUnit wykorzystuje asercje. Za pomocą m.in. *assertEquals*, programista jest w stanie porównać spodziewany wynik z wynikiem testu. W przypadku niepowodzenia jest on odpowiednio informowany o zaistniałym błędzie.

5.1.2. Mockito

Mockito jest to framework ułatwiający pisanie testów. Sprawia on, że stają się one czytelniejsze, a powstałe błędy są łatwiejsze do zrozumienia. [11] Głównymi jego zaletami jest możliwość zamiany poszczególnych elementów na

atrapy, których zachowanie można przewidzieć. Sprawia to, że dane obiekty stają się wobec siebie niezależne.

W tym celu framework daje możliwość korzystania z adnotacji:

- @Mock - inicjalizuje mockowany obiekt,
- @Runwith - deklaruje, że dana klasa będzie klasą mockującą,
- @Before - wskazuje metodę, która zostanie uruchomiona przed każdym testem.

5.2. Implementacja

Do napisania testów wykorzystany został abstract zapewniający podstawową konfigurację danej grupy testów.

```
public abstract class AbstractTest {
   protected MockMvc mvc;
   @Autowired
   WebApplicationContext webApplicationContext;

   protected void setUp() {
       mvc = MockMvcBuilders
             .webAppContextSetup(webApplicationContext)
             .addFilter(new SecurityFilterMock()).build();
   }
   protected String mapToJson(Object obj) throws JsonProcessingException {
       ObjectMapper objectMapper = new ObjectMapper();
       return objectMapper.writeValueAsString(obj);
   }
   protected <T> T mapFromJson(String json, Class<T> clazz)
           throws JsonParseException, JsonMappingException, IOException {

       ObjectMapper objectMapper = new ObjectMapper();
       return objectMapper.readValue(json, clazz);
   }
}
```

Listing 1 Kod klasy AbstractTest

Zadaniem metody setUp jest przygotowanie makiety całej aplikacji, tak aby testowane mogły być poszczególne zapytania. Dodatkowo abstract posiada dwie metody: mapToJson oraz mapFromJson, których zadaniem jest konwersja danych pomiędzy formatem JSON-em a obiektem Javy.

5.2.1. Testy jednostkowe

Pierwszym testem jednostkowym jest kontrola poprawność działania funkcji o nazwie *checkIfTreatmentHasErrors* sprawdzającej posiadanie wszystkich wymaganych atrybutów w obiekcie „NewTreatmentDto". Schemat tego obiektu zaprezentowano na listingu 2.

```
public class NewTreatmentDto {
   private String name;
   private String description;
   private String image;
   private Integer price;
   private Integer treatmentDuration;
   private Long categoryId;
}
```

Listing 2 Kod klasy NewTreatmentDto

Wymagane jest, aby:

- name - nie może być puste,
- price - nie może być mniejsze od 1,
- treatmentDuration - nie może być mniejsze od 1,
- categoryId - nie może być puste.

W tym celu napisana została klasa testująca różne możliwe stany obiektu NewTreatmentDto. W tabelach 5, 6, 7 oraz 8 przedstawione zostały testy sprawdzające kolejne przypadki:

Tabela 5 Test funkcji sprawdzającej nowy zabieg przypadek 1

Stan obiektu	poprawny
Spodziewany wynik	brak błędów - false
Kod	

```
@Test
public void shouldFindCorrectObjectAndReturnFalse() {
   NewTreatmentDto newTreatmentDto = new NewTreatmentDto()

.toBuilder().name("Name").treatmentDuration(10)
                                    .categoryId(1L).price(15).build();
   Boolean result =
utilityFunctions.checkIfTreatmentHasErrors(newTreatmentDto);

   Assertions.assertFalse(result);
}
```

Listing 3 Kod testu sprawdzającego obiekt nowy zabieg - 1

Tabela 6 Test funkcji sprawdzającej nowy zabieg przypadek 2

Stan obiektu	name jest puste
Spodziewany wynik	posiada błąd - true
Kod	

```
@Test
public void shouldFindBlankNameAndReturnTrue() {
   NewTreatmentDto newTreatmentDto = new NewTreatmentDto().toBuilder()
                                     .name("").treatmentDuration(10)
                                     .categoryId(1L).price(15).build();

   Boolean result = utilityFunctions
                    .checkIfTreatmentHasErrors(newTreatmentDto);

   Assertions.assertTrue(result);
}
```

Listing 4 Kod testu sprawdzającego obiekt nowy zabieg – 2

Tabela 7 Test funkcji sprawdzającej nowy zabieg przypadek 3

Stan obiektu	price jest ujemne
Spodziewany wynik	posiada błąd - true
Kod	

```
@Test
public void shouldFindNegativePriceAndReturnTrue() {
   NewTreatmentDto newTreatmentDto = new NewTreatmentDto().toBuilder()
                                     .name("Name").treatmentDuration(10)
                                     .categoryId(1L).price(-20)
                                     .build();

   Boolean result = utilityFunctions.
                           checkIfTreatmentHasErrors(newTreatmentDto);

   Assertions.assertTrue(result);
}
```

Listing 5 Kod testu sprawdzającego obiekt nowy zabieg – 3

Tabela 8 Test funkcji sprawdzającej nowy zabieg przypadek 4

Stan obiektu	brak treatmentDuration
Spodziewany wynik	posiada błąd - true
Kod	

```
@Test
public void shouldFindNullDurationAndReturnTrue() {
   NewTreatmentDto newTreatmentDto = new NewTreatmentDto().toBuilder()
                                          .name("Name").categoryId(1L)
                                          .price(15).build();

   Boolean result = utilityFunctions
                          .checkIfTreatmentHasErrors(newTreatmentDto);

   Assertions.assertTrue(result);
}
```

Listing 6 Kod testu sprawdzającego obiekt nowy zabieg – 4

Tabela 9 Test funkcji sprawdzającej nowy zabieg przypadek 5

Stan obiektu	brak categoryId
Spodziewany wynik	posiada błąd - true
Kod	

```
@Test
public void shouldFindNullCategoryAndReturnTrue() {
   NewTreatmentDto newTreatmentDto = new NewTreatmentDto().toBuilder()
                                     .name("Name").treatmentDuration(10)
                                     .price(15).build();

   Boolean result = utilityFunctions
                          .checkIfTreatmentHasErrors(newTreatmentDto);

   Assertions.assertTrue(result);
}
```

Listing 7 Kod testu sprawdzającego obiekt nowy zabieg - 5

Do przeprowadzenia kolejnego testu jednostkowego, konieczne było przygotowanie atrapy serwisu, z którego testowany komponent korzysta. W tym celu napisany został profil konfiguracyjny przedstawiony w listingu 8.

```
@Profile("reservations")
@Configuration
public class ReservationTestConfiguration {
   @Bean
   @Primary
   public ReservationService reservationService(){
       return Mockito.mock(ReservationService.class);
   }
}
```

Listing 8 Kod profilu konfiguracyjnego

Zamiast oryginalnej implementacji serwisu wstrzykiwana jest atrapa, do której w trakcie wykonywania testu dołączony jest spodziewany wynik.

Z racji, iż testowana klasa jest kontrolerem, test sprawdzał będzie poprawność zapytania jednego z endpointów - pobieranie informacji o lekarzu. Spodziewanym wynikiem jest w tym przypadku status 200 oraz obiekt klasy DoctorDto.

```
@ActiveProfiles("reservations")
public class ReservationsControllerTests extends AbstractTest {

   @Autowired
   private ReservationService reservationService;
   @Override
   @Before
   public void setUp() {
       super.setUp();
   }

   @Test
   public void shouldFindDoctorsDetails() throws Exception {
       DoctorDto doctorDto = new DoctorDto().toBuilder()
               .description("Description").firstName("Doctor")
               .firstName("Good").image("www.firebase-storage.com")
               .id(2L).build();
       when(reservationService.getDoctorById(2L))
               .thenReturn(doctorDto);
       String uri = "/public/doctor/2";
       MvcResult mvcResult = mvc
               .perform(MockMvcRequestBuilders.get(uri)
               .contentType(MediaType.APPLICATION_JSON_VALUE))
               .andReturn();
       int status = mvcResult.getResponse().getStatus();
       Assertions.assertEquals(200, status);
```

```
        String content = mvcResult.getResponse().getContentAsString();
        DoctorDto responseObject
=mapFromJson(content,DoctorDto.class);
        Assertions.assertEquals(responseObject, doctorDto);
    }
}
```

Listing 9 Kod testu kontrolera Reservations

Kod przedstawiony w listingu 9 jest to klasa testująca wraz z funkcją testową *shouldFindDoctorDetails*. Dzięki @Autowired wstrzykiwana jest wcześniej utworzona atrapa serwisu. Następnie przed każdym testem wywoływana jest funkcja setUp, która korzysta z implementacji przygotowanej przez klasę, po której dziedziczy, w tym przypadku po AbstractTest. Adnotacją @Test rozpoczyna się metoda testowa. Z racji, iż korzysta ona z atrapy serwisu, używając *when*, ma ona możliwość określić co jego funkcja *getDoctorById* powinna zwrócić. Następnie po określeniu adresu, następuje wysłanie zapytania, którego odpowiedź zapisywana jest w zmiennej *mvcResult*. Następnie używając metod *assertionEquals*, weryfikowane jest, czy otrzymany rezultat jest zgodny z założeniami.

5.2.2. Testy integracyjne

Do przeprowadzenia testów integracyjnych konieczna była zmiana implementacji klasy odpowiedzialnej za filtrowanie zapytań - SecurityFilter. Z tego względu przygotowano mock, który rozpoznaje użytkownika jako pacjenta oraz zwraca jego dane. Jego kod został przedstawiony w listingu 10.

```
public class SecurityFilterMock extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain)
            throws ServletException, IOException {
        verifyToken(request);
        filterChain.doFilter(request, response);
    }

    private void verifyToken(HttpServletRequest request) {
        User user = getUser();
        if (user != null) {
            UsernamePasswordAuthenticationToken authentication = new
UsernamePasswordAuthenticationToken(user, "Credentials" , null);
            authentication.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authentication);
```

```
        }
    }

    public User getUser() {
        User user = new User();
        user.setId(1L);
        user.setUid("Uid");
        user.setEmail("Template@Email.com");
        user.setEmailVerified(true);
        user.setId(2L);
        user.setAccountType(AccountType.PATIENT);
        user.setFirstName("Leon");
        user.setLastName("Zawodowiec");
        user.setPhone("123123123");
        return user;
    }
}
```

Listing 10 Kod mock'a klasy SecurityFilter

W klasie *SecurityFilterMock* sprawdzany jest token autoryzacyjny użytkownika. Na potrzeby testów przedstawiona implementacja została zmieniona na taką, która zawsze rozpoznaje użytkownika jako pacjenta. Następuje to w funkcji *verifyToken*, która to po otrzymaniu obiektu User autoryzuje dostęp. Ów filtr zostaje dodany w trakcie przygotowania środowiska testowego aplikacji.

Pierwsza grupa testów integracyjnych przygotowana została w celu sprawdzenia poprawności działania kontrolera dostępnego jedynie dla administratora. Jej pierwszy test ma za zadanie zweryfikować, czy użytkownik mający przypisaną rolę Pacjent, może pobrać informacje o wszystkich Pacjentach w systemie.

```
@Test
public void shouldAccessBeForbidden() throws Exception {
    String uri = "/private/users/all";
    MvcResult mvcResult =
              mvc.perform(MockMvcRequestBuilders.get(uri)
                 .accept(MediaType.APPLICATION_JSON_VALUE)
                 .queryParam("accountType","PATIENT")).andReturn();
    int status = mvcResult.getResponse().getStatus();
    Assertions.assertEquals(403,status);
}
```

Listing 11 Kod testu sprawdzającego role administrator

Test przedstawiony w listingu 11, rozpoczyna się przygotowaniem adresu. Następnie zostaje wykonane zapytanie, którego odpowiedź zapisywana jest do zmiennej *mvcResult*. Ostatnim krokiem jest sprawdzenie statusu odpowiedzi. W przypadku odmowy dostępu zwracany powinien być kod 403.

Kolejne testy przeprowadzone na tym kontrolerze, sprawdzają poprawność dostępu administratora do zablokowanych zasobów oraz wywołanie całej procedury dodania kategorii zabiegów do systemu. W tym celu rola użytkownika zostaje zmieniona na Admin, a następnie zostają wywołane funkcje testowe.

```
@Test
public void shouldCreateAndReturnCategory() throws Exception {
   String uri = "/private/category";
   String category = "testCategory";
   MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.post(uri)
           .contentType(MediaType.APPLICATION_JSON_VALUE)
           .param("category", category)).andReturn();
   int status = mvcResult.getResponse().getStatus();
   Assertions.assertEquals(200, status);
   String content = mvcResult.getResponse().getContentAsString();
   TreatmentCategoryDto tDto =
mapFromJson(content,TreatmentCategoryDto.class);
   Assertions.assertEquals(category,tDto.getName());
}
```

Listing 12 Kod testu sprawdzającego tworzenie kategorii

Pierwszy test przedstawiony w listingu 12, zakłada dodanie nowej kategorii oraz jej zwrócenie. Rozpoczyna się od przygotowania adresu oraz jej nazwy. Następnie wywoływane jest zapytanie podając jako parametr utworzoną zmienną. Ostatnim etapem jest porównywanie rezultatu z założeniami - w tym wypadku zarówno statusu, jak i zwracanej nazwy.

Drugi test dotyczący kategorii sprawdza, czy w przypadku nieuzyskania nazwy, system przerwie proces oraz zwróci odpowiedni status.

```
@Test
public void shouldNotCreateCategoryNoName() throws Exception {
   String uri = "/private/category";
   String category = "";
   mvc.perform(MockMvcRequestBuilders.post(uri)
                   .contentType(MediaType.APPLICATION_JSON_VALUE)
                   .param("category",category))
                   .andExpect(status().isBadRequest())
                   .andExpect(result ->
                      Assertions.assertEquals("No category name",
                         result.getResponse().getContentAsString()));
}
```

Listing 13 Kod testu sprawdzającego błąd przy tworzeniu kategorii

Funkcja testowa ma ten sam schemat co poprzednia, z tą różnicą, iż nazwa kategorii pozostaje pusta. Test oczekuję zwrócenia statusu 400 oznaczającego *bad request* oraz tekstu „No category name”.

Następnym testowanym elementem jest kontroler odpowiedzialny za obsługę rejestracji oraz zwracanie informacji o użytkowniku po zalogowaniu. W tym celu klasa testowa została przygotowana jak poprzednie.

Pierwszym testowanym przypadkiem jest rejestracja. Weryfikowane jest, czy po wysłaniu zapytania odpowiedź będzie miała poprawny status oraz czy zwrócony zostanie użytkownik o tym samym e-mailu, jaki został podany przy rejestracji.

```
@Test
public void shouldRegisterTheUser() throws Exception {
   String uri = "/public/user/register";
   NewUserDto newUser = new NewUserDto().toBuilder()
        .accountType(AccountType.PATIENT).allowedNotifications(false)
        .userId("123321").firstName("name")lastName("lastName")
        .email("email@test.com").build();
   MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.post(uri)
           .contentType(MediaType.APPLICATION_JSON_VALUE)
           .content(mapToJson(newUser))).andReturn();
   int status = mvcResult.getResponse().getStatus();

   Assertions.assertEquals(200, status);

   String content = mvcResult.getResponse().getContentAsString();
   AccountDetailsDto accountDetailsDto =
                super.mapFromJson(content, AccountDetailsDto.class);

   Assertions.assertNotNull(accountDetailsDto);
   Assertions.assertEquals(accountDetailsDto.getEmail(),
                               newUser.getEmail());
}
```

Listing 14 Kod testu sprawdzającego rejestrowanie

Test rozpoczyna się od przygotowania adresu oraz obiektu nowego użytkownika. Następnie wysyłane jest zapytanie wraz z obiektem z danymi utworzonymi dzięki metodzie *mapToJson*. Z zapisanej w zmiennej odpowiedzi pobierany jest jej status oraz string, który następnie dzięki funkcji *mapFromJson* zamieniany jest na obiekt. Ostatnim etapem jest porównywanie wyników z założeniami.

Z racji, iż pobranie informacji o użytkowniku wymaga autoryzacji, w kolejnym teście wykorzystany został mock filtra, przygotowany podczas testowania kontrolera administracji. Zaprezentowany w listingu 15 kod, przedstawia funkcje testującą ten przypadek.

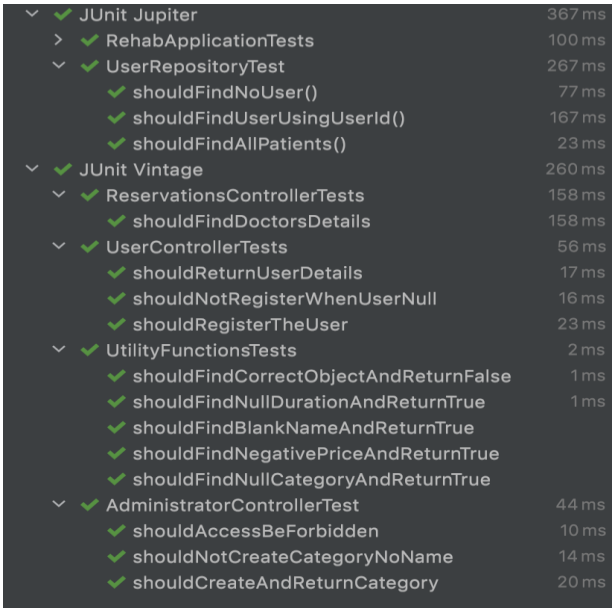
```java
@Test
public void shouldReturnUserDetails() throws Exception {
   String uri = "/private/user";
   MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.get(uri)

.contentType(MediaType.APPLICATION_JSON_VALUE)).andReturn();
   int status = mvcResult.getResponse().getStatus();

   Assertions.assertEquals(200, status);

   String content = mvcResult.getResponse().getContentAsString();
   User user = super.mapFromJson(content,User.class);

   Assertions.assertNotNull(user);
   Assertions.assertEquals(user.getFirstName(), "Leon");
   Assertions.assertEquals(user.getLastName(), "Zawodowiec");
}
```

Listing 15 Kod testu sprawdzającego zwracanie informacji o użytkowniku

Test rozpoczyna się od przygotowania adresu. Następnie wywoływane jest zapytanie. Po otrzymaniu odpowiedzi, jej status oraz informacje o użytkowniku, zostają porównane z założeniami.

5.2.3. Wyniki

Zarówno testy jednostkowe, jak i testy integracyjne zakończyły się tak, jak przewidziano, co zaprezentowano na rysunku 45. Potwierdzają one poprawność sposobu zaprojektowania oraz implementacji systemu. Dodatkowo stanowią zabezpieczenie przed uszkodzeniami w przyszłości.



Rysunek 45 Wyniki przeprowadzonych testów, źródło: opracowanie własne

6. Podsumowanie

Celem pracy było zaprojektowanie aplikacji umożliwiającej rezerwacje wizyt w gabinecie fizjoterapii. Założenia przedstawione we wstępie pracy zostały zrealizowane. Wykonane zostało:

- zaprojektowana aplikacja w postaci strony internetowej, umożliwiająca tworzenie rezerwacji online,
- serwer do obsługi zapytań wysyłanych przez stronę gabinetu,
- relacyjna baza danych przechowująca wszystkie potrzebne do prawidłowego działania systemu informacje,
- system umożliwiający autoryzację użytkowników,
- wprowadzono 4 role, które różnią się uprawnieniami.

W trakcie tworzenia aplikacji użyte zostały dodatkowe narzędzia, jakim jest między innymi Firebase. Posłużył on do zarządzania procesem rejestracji oraz logowania. Dodatkowo wykorzystany został do przechowywania plików multimedialnych wyświetlanych w aplikacji.

System posiada wiele ścieżek rozwoju. Jednym z ważniejszych jest dodanie obsługi płatności czy też analityki. Pierwsza z nich sprawiłaby, iż zwiększyłby się komfort użytkowania aplikacji - potencjalny pacjent spędziłby jeszcze mniej czasu w placówce. Druga zaś pozwoliłaby dowiedzieć się więcej na temat odwiedzających stronę. Kontrola m.in. najchętniej sprawdzanych zabiegów, najczęściej odwiedzanych stron umożliwiłaby dostosowywanie treści tak, aby bardziej zaspokajała potrzeby klientów.

Podczas implementacji zauważono, iż projektowany system mógłby w znacznym stopniu ułatwić organizację dnia zarówno przez gabinet, jak i pacjentów. Ponadto zminimalizowałby ilość czasu spędzanego w poczekalni w oczekiwaniu na przyjęcie. Tym samym zmalałaby długość kolejek do rejestracji, czy też do poszczególnych sal zabiegowych.

Proces projektowania oraz implementacji aplikacji do rezerwacji wizyt w gabinecie fizjoterapii pozwolił na dokładniejsze zrozumienie sposobu komunikacji między aplikacją, a serwerem. Dodatkowo umożliwił poznanie standardów programowania panujących w branży IT.

7. Bibliografia

[1] „**Architektura klient-serwer**” [Online] https://pl.wikipedia.org/wiki/Klient-serwer. [dostęp: 10 Styczeń 2022].

[2] „**Architektura oprogramowania**” [Online] http://architektura-oprogramowania.blogspot.com/p/architektura-klient-serwer.html [dostęp: 10 Styczeń 2022]

[3] „**Bootstrap**” [Online]. https://pl.wikipedia.org/wiki/Bootstrap_(framework). [dostęp: 11 Styczeń 2022].

[4] „**Dokumentacja techniczna Firebase Analytics**” [Online]. https://firebase.google.com/docs/analytics. [dostęp: 12 Styczeń 2022].

[5] „**Dokumentacja techniczna Firebase Auth**” [Online]. https://firebase.google.com/docs/auth. [dostęp: 12 Styczeń 2022].

[6] „**Dokumentacja techniczna Spring Security**” [Online]. https://docs.spring.io/spring-security/site/docs/3.0.x/reference/security-filter-chain.html. [dostęp: 12 Styczeń 2022].

[7] „**Interfejsy API REST**” [Online]. https://www.ibm.com/pl-pl/cloud/learn/rest-apis. [dostępu: 11 Styczeń 2022].

[8] „**JavaScript i jQuery**”, D. Borycki, Helion, 2014.

[9] „**Java dokumentacja**” [Online]. https://java.com/pl/about/. [dostęp: 12 Styczeń 2022].

[10] „**Java w pigułce Wydanie VI**”, D. F. Benjamin J Evans, Helion, 2015.

[11] „**Mockito Dokumentacja techniczna**” [Online]. https://site.mockito.org/. [dostęp: 10 Luty 2022].

[12] „**Model Relacyjny**” [Online]. https://pl.wikipedia.org/wiki/Model_relacyjny. [dostęp: 12 Styczeń 2022].

[13] „**React i Redux Praktyczne tworzenie aplikacji WWW. Wydanie II**”, K. Chinnathambi, Helion, 2019.

[14] „**Spring w akcji Kompendium wiedzy na temat Spring Framework. Wydanie IV**”, C. Walls,, Helios.

[15] „**The Art of Unit Testing**”, R. Osherove, Manning, 2009.

[16] „**Wykres popularności narzędzi JavaScript do tworzenia stron WWW**” [Online] https://www.npmtrends.com/react-vs-vue-vs-@angular/core. [dostęp: 10 Luty 2022].

[17] „**Wprowadzenie do Spring Data Jpa**” [Online]. https://nullpointerexception.pl/spring-data-jpa-wprowadzenie/. [dostępu: 11 Styczeń 2022].

8. Spis rysunków

9. Spis tabel

10. Spis listingów