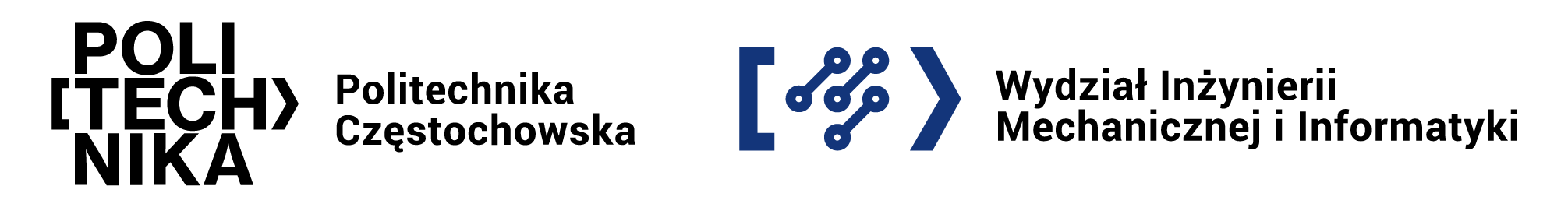
****

PRACA INŻYNIERSKA

**Projekt i implementacja systemu informatycznego wspomagającego zarządzanie klientami i usługami w sieci klubów fitness**

*Project and implementation of an IT system supporting the management of clients and services in the network of fitness clubs.*

Patryk Skwara

Nr albumu: 129766

Kierunek: Informatyka

Forma studiów: stacjonarne

Poziom studiów : I

Promotor pracy: Dr inż. Mariusz Ciesielski

Praca przyjęta dnia:

Podpis promotora:

*Częstochowa, …….*

# Projekt i implementacja aplikacji wspomagającej zarządzanie klientami i usługami w sieci klubów fitness.

Aplikacja stworzona na potrzeby tej pracy powinna być oprogramowaniem umożliwiającym użytkownikowi czyli w domyśle recepcjoniście klubu fitness pełne zarządzanie obiektem. Użytkownik powinien mieć możliwość:

* Zalogowania się do aplikacji przy użyciu swojego indywidualnego konta stworzonego przez administratora,
* Rejestracji klienta,
* Szybkiego wyszukania klienta,
* Edycji danych osobowych klienta,
* Zmiany statusu klienta,
* Wglądu do terminu ważności karnetu,
* Aktualizacji karnetu uprawniającego do korzystania z klubu,
* Dodawania wpisów konkretnemu klientowi odnośnie jego wizyt w klubie tzn. symulacja działania karty magnetycznej polegająca na odczytaniu danych klienta przy użyciu czytnika, który znajduje się przy drzwiach wejściowych do klubu w celu zarejestrowania momentu wejścia do klubu oraz wyjścia z klubu.
* Przeglądania historii wizyt klienta,
* Przeglądania sprzętu dostępnego dla klientów,
* Dodawania nowej pozycji do danej kategorii sprzętu,
* Edycji konkretnej pozycji w danej kategorii sprzętu,
* Przeglądania pracowników zatrudnionych na etat w klubie,
* Przeglądania wydarzeń przydzielonych do konkretnego pracownika,
* Przeglądania kalendarza wszystkich wydarzeń,
* Dodawania wydarzenia,
* Edycji wydarzenia,
* Podglądu szczegółowych informacji na temat konkretnego wydarzenia,
* Usuwania, archiwizowania wydarzenia,
* Przeglądania asortymentu produktów przeznaczonych do sprzedaży klientom,
* Dodawania nowej pozycji produktu,
* Aktualizowania stanu produktu,
* Przeglądania swoich danych osobowych,

Zmiany zdjęcia profilowego,

* Zmiany hasła,
* Zmiany adresu e-mail

Oprócz konta użytkownika istnieje również konto administratora aplikacji, który posiada wszystkie uprawnienia użytkownika oraz jego funkcjonalność jest rozszerzona o:

* Przeglądanie użytkowników,
* Rejestracje użytkownika,
* Usuwanie konta użytkownika

## 1.1 Wymagania funkcjonalne:

Aplikacja jest podzielona na poszczególne zakładki, kolejno: klienci, sprzęt, pracownicy, wydarzenia, asortyment, użytkownicy (w przypadku konta administratora) oraz ustawienia.

Na poniższym diagramie przypadków użycia widnieje dwóch aktorów – Administrator i użytkownik czyli konta aplikacji przydzielone do następujących funkcjonalności:

## Kolumna - Klienci:

* Rejestracja klienta – Administrator, użytkownik.
* Edycja danych osobowych – Administrator, użytkownik.
* Przeglądanie klientów – Administrator, użytkownik.
* Wyszukiwanie klientów – Administrator, użytkownik.
* Aktualizacja karnetu – Administrator, użytkownik.
* Rejestracja wizyt - Administrator, użytkownik.
* Przegląd wizyt - Administrator, użytkownik.
* Zmiana statusu - Administrator, użytkownik.

## Kolumna – Sprzęt:

* Przeglądanie sprzętu - Administrator, użytkownik,
* Dodanie nowej pozycji - Administrator, użytkownik,
* Edycja pozycji - Administrator, użytkownik,

## Kolumna – Pracownicy:

* Przeglądanie pracowników - Administrator, użytkownik,
* Podgląd wydarzeń - Administrator, użytkownik,

## Kolumna – Wydarzenia:

* Przeglądanie wydarzeń - Administrator, użytkownik,
* Dodanie wydarzenia - Administrator, użytkownik,
* Edycja wydarzenia - Administrator, użytkownik,
* Podgląd szczegółowych informacji - Administrator, użytkownik,
* Usuwanie wydarzenia - Administrator, użytkownik,

## Kolumna – Asortyment:

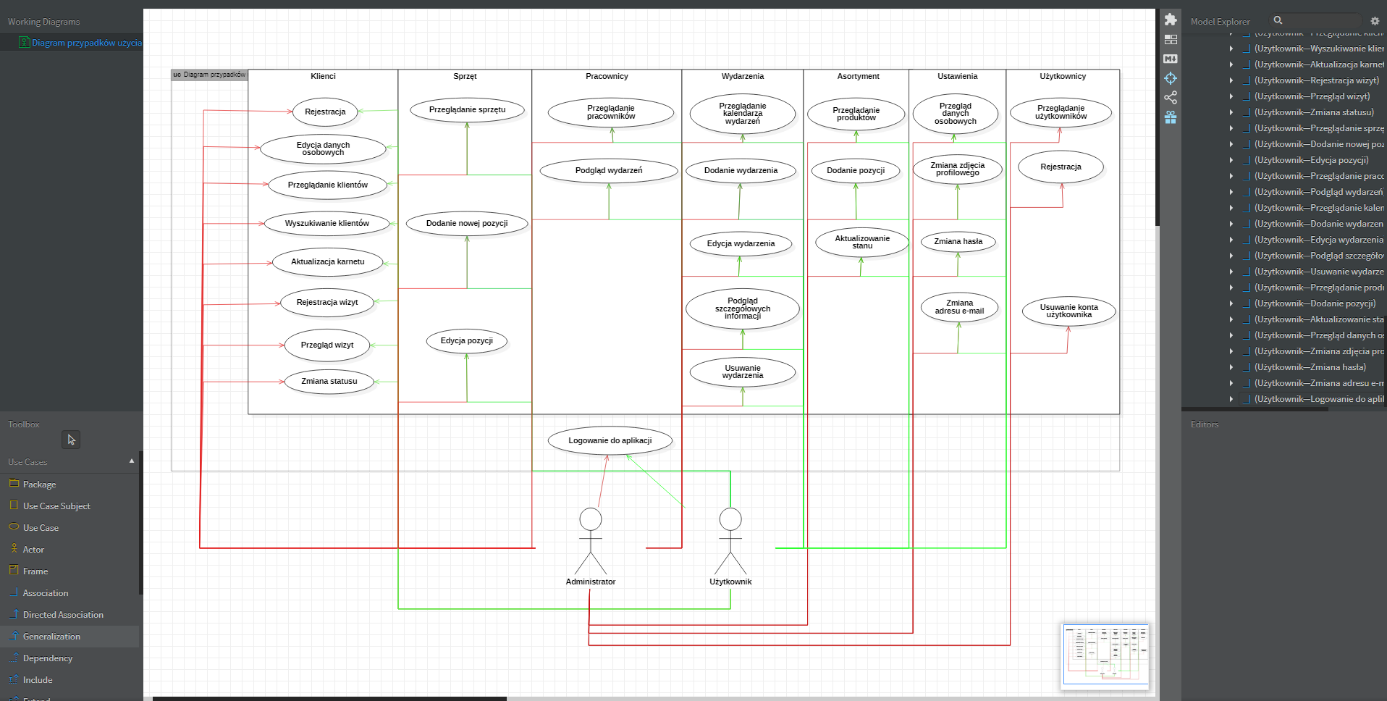
* Przeglądanie produktów - Administrator, użytkownik,
* Dodanie pozycji - Administrator, użytkownik,
* Aktualizowanie stanu - Administrator, użytkownik,

## Kolumna – Ustawienia:

* Przegląd danych osobowych - Administrator, użytkownik,
* Zmiana zdjęcia profilowego - Administrator, użytkownik,
* Zmiana hasła - Administrator, użytkownik,
* Zmiana adresu e-mail - Administrator, użytkownik,

## Kolumna – Użytkownicy:

* Przeglądanie użytkowników – Administrator,
* Rejestracja - Administrator,
* Usuwanie konta użytkownika - Administrator



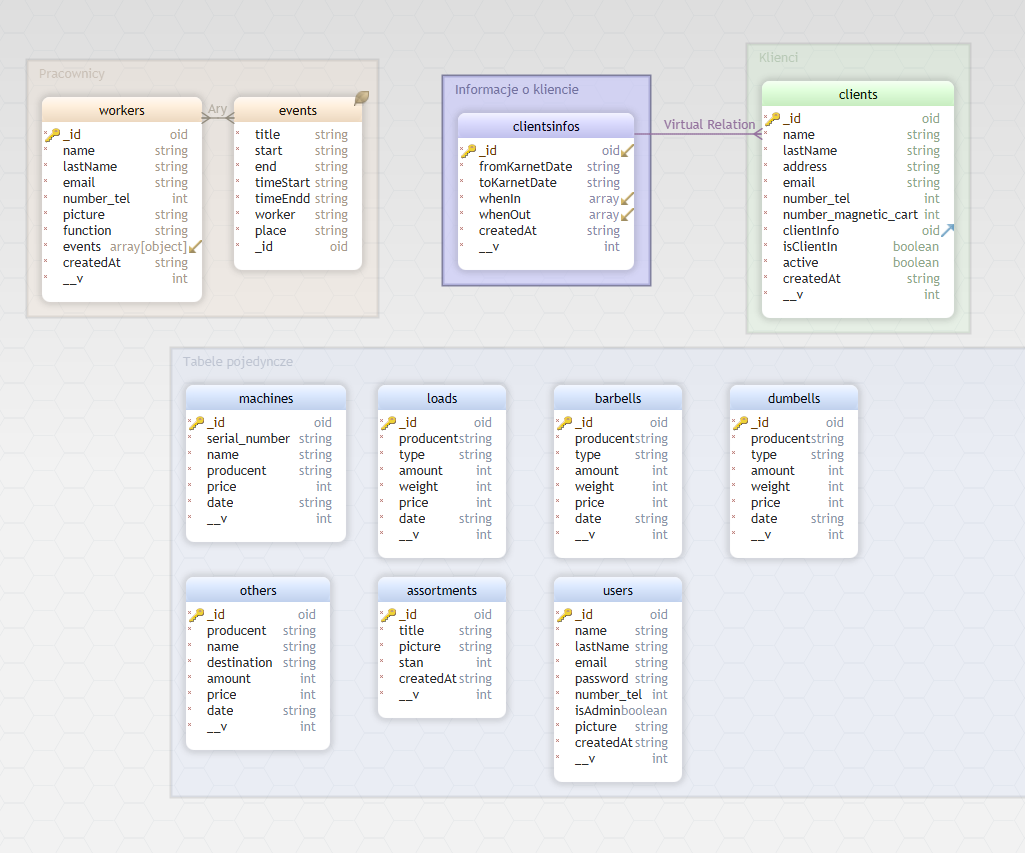
Ilustr.1: Diagram przypadków użycia

## 1.2 Wymagania niefunkcjonalne:

* Użytkownik aplikacji nie powinien musieć przeładowywać strony po dokonaniu jakichkolwiek zmian. W związku z tym w aplikacji została wykorzystana technologia Single Page Application przy użyciu biblioteki React.js.
* W momencie uruchomienia aplikacji wszelkie potrzebne dane zostają pobrane za pośrednictwem aplikacji serwerowej z bazy danych,
* Użytkownik podczas logowania powinien mieć automatycznie generowany token dostępowy.
* Użytkownik w momencie logowania do aplikacji jest zweryfikowany przez serwer w taki sposób, aby mógł zalogować się tylko za pośrednictwem swojego konta widniejącego w bazie danych.
* Hasła wszystkich użytkowników powinny zostać zaszyfrowane.
* Aplikacja jest elastyczna, przejrzysta i prosta w obsłudze. Trzeba pamiętać, że z aplikacji będą korzystać osoby niezwiązane z dziedziną informatyki.
* Wszelkie zmiany w aplikacji są odnotowane i zapisane w bazie danych tak, aby po ponownym zalogowaniu się do aplikacji nie utracić wprowadzonych zmian.
* Szczególne pola formularzy są objęte walidacją aby uchronić aplikację przed niechcianymi danymi.
* Podczas wylogowania się z aplikacji token uwierzytelniający jest usuwany z pamięci podręcznej, dzięki czemu użytkownik nie może powrócić do aplikacji wykorzystując przycisk „wstecz”.
* Tylko jeden użytkownik może być zalogowany w aplikacji w czasie rzeczywistym.

# 2. Projekt bazy danych

Aplikacja korzysta z otwartego, nierelacyjnego systemu zarządzania bazą danych. Charakteryzuje się brakiem ściśle zdefiniowanej struktury obsługiwanych baz danych. Zamiast tego dane składowane są jako dokumenty w stylu JSON. (<https://pl.wikipedia.org/wiki/MongoDB>)



Ilustr. 2: Schemat bazy danych

Ilustracja 2 przedstawia schemat struktury bazy danych zawierającej 11 tabel z czego cztery z nich są połączone wirtualnymi relacjami. Poniżej omówione zostaną kolejne tabele oraz pola i relację między tabelami.

## Tabela „workers”

Tabela „workers” zawiera w sobie pracowników wewnętrznych, a konkretnie ich dane osobowe, zdjęcie profilowe oraz tabele obiektów przechowującą przypisane do poszczególnego pracownika wydarzenia.

* **\_id** – Pole zawierające unikalny ciąg znaków w celu identyfikacji konkretnej pozycji.
* **name** – Pole określające imię pracownika.
* **lastName** – Pole określające nazwisko pracownika.
* **email** – Pole określające adres e-mail pracownika.
* **number\_tel** – Pole określające numer telefonu pracownika.
* **picture** – Pole przechowujące zdjęcie poglądowe pracownika.
* **function** – Pole określające funkcję jaką pracownik sprawuje w klubie fitness.
* **events** – Pole przechowujące wydarzenia przydzielone pracownikowi.
* **createdAt** – Pole przechowujące date utworzenia pracownika.

## Tabela „events”

Tabela „events” jest powiązania relacją jeden do wielu z tabelą „workers” tzn. jeden pracownik może posiadać kilka wydarzeń. Zawiera w sobie szczegółowe informacje na temat konkretnego wydarzenia.

* **\_id** – Pole zawierające unikalny ciąg znaków w celu identyfikacji konkretnego wydarzenia.
* **title** – Pole określające tytuł wydarzenia.
* **start** – Pole określające datę rozpoczęcia wydarzenia.
* **end** – Pole określające datę końca wydarzenia.
* **timeStart** – Pole określające godzinę rozpoczęcia wydarzenia.
* **timeEndd** – Pole określające godzinę zakończenia wydarzenia.
* **worker** – Pole informujące o użytkowniku rejestrującym wydarzenie.
* **place** – Pole określające miejsce wydarzenia.
* **organizer** – Pole określające pracownika organizującego wydarzenie.

## Tabela „clients”

Tabela „clients” przechowuje dane osobowe klientów, numer karty magnetycznej oraz referencję do tabeli „clientsInfo”, z którą jest powiązana relacją jeden do jednego.

* **\_id** – Pole zawierające unikatowy ciąg znaków w celu identyfikacji konkretnego klienta.
* **name** – Pole określające imię klienta.
* **lastName** – Pole określające nazwisko klienta.
* **address** – Pole określające adres klienta.
* **email** – Pole określające adres e-mail klienta.
* **number\_tel** – Pole określające numer telefonu klienta.
* **number-magnetic\_card** – Pole określające unikatowy numer karty magnetycznej.
* **clientInfo** – Pole zawierające referencję do tabeli „clientsInfo”, czyli ciąg znaków określający pozycje w tabeli przypisaną do konkretnego klienta.
* **isClientIn** – Pole zawierające tzw. Flagę w celu określenia czy klient jest aktualnie w klubie.
* **active** – Pole określające czy klient jest aktywnym konsumentem klubu.
* **createdAt** – Pole określające datę rejestracji klienta przez użytkownika.

## Tabela „clientsInfo”

Tabela „clientsInfo” przechowuje szczegółowe informacje na temat klienta, czyli datę ważności karnetu oraz daty i godziny jego wizyt w klubie.

* **\_id** – Unikatowy ciąg znaków w celu identyfikacji informacji o konkretnym kliencie.
* **fromKarnetDate** – Pole zawierające datę rozpoczęcia karnetu.
* **toKarnetDate** – Pole zawierające datę zakończenia karnetu.
* **whenIn** – Pole określające datę i godzinę wizyty klienta w klubie.
* **whenOut** – Pole określające datę i godzinę opuszczenia klubu przez klienta.
* **createdAt** – Pole określające datę utworzenia.

## Tabela „machines”

Tabela „machines” przechowuje informację dotyczące zakupionych maszyn.

* **\_id** – Pole zawierające unikatowy ciąg znaków w celu identyfikacji konkretnej pozycji.
* **serial\_number** – Pole przechowujące numer seryjny sprzętu.
* **name** – Pole określające nazwę maszyny.
* **producent** – Pole określające nazwę producenta maszyny.
* **price** – Pole określające cenę.
* **date** – Pole określające datę zakupu maszyny.

## Tabela „loads”

Tabela „loads” przechowuje informację dotyczące zakupionych obciążeń.

* **\_id** – Pole zawierające unikatowy ciąg znaków w celu identyfikacji konkretnej pozycji.
* **producent** – Pole określające nazwę producenta obciążenia.
* **type** – Pole przechowujące typ obciążenia.
* **amount** – Pole określające ilość talerzy obciążenia na stanie.
* **weight** – Pole określające wagę obciążenia.
* **price** – Pole przechowujące cenę zakupu.
* **date** – Pole określające datę zakupu.

## Tabela „barbells” i „dumbells”

Tabele „barbells” i „dumbells” przechowują informację dotyczące zakupionych gryfów prostych, łamanych lub wypełnionych obciążeniem sztang lub hantli.

* **\_id** – Pole zawierające unikatowy ciąg znaków w celu identyfikacji konkretnej pozycji w tabeli.
* **producent** – Pole zawierające nazwę producenta.
* **type** – Pole przechowujące typ sztangi.
* **amount** – Pole określające ilość danej pozycji.
* **weight** – Pole określające wagę sztangi.
* **price** – Pole określające cenę zakupu.
* **date** – Pole określające datę zakupu.

## Tabela „others”

Tabela „others” przechowuje informację dotyczące pozostałych akcesoriów dostępnych klientowi.

* \_**id** – Unikatowe pole identyfikujące daną pozycję w tabeli.
* **producent** – Pole zawierające nazwę producenta.
* **name** – Pole określające nazwę sprzętu.
* **destination** – Pole określające przeznaczenie sprzętu.
* **amount** – Pole zawierające ilość sprzętu.
* **price** – Pole zawierające cenę sprzętu.
* **date** – Pole określające datę kupna sprzętu.

## Tabela „assortments”

Tabela „assortments” przechowuję informację dotyczące produktu w magazynie.

* \_**id –** Unikatowe pole identyfikujące daną pozycję produktu w tabeli.
* **title –** Pole określające nazwę produktu.
* **picture –** Pole zawierające zdjęcie produktu.
* **stan –** Pole określające ilość produktu.
* **createdAt –** Pole określające datę dodania produktu.

## Tabela „users”

Tabela „users” przechowuję informację na temat danych osobowych użytkowników aplikacji.

* \_**id** – Unikatowe pole identyfikujące danego użytkownika.
* **name** – Pole określające imię użytkownika.
* **lastName** – Pole określające nazwisko użytkownika**.**
* **email** – Pole określające adres e-mail użytkownika.
* **password** – Pole zawierające zakodowane hasło użytkownika.
* **number\_tel** – Pole określające numer telefonu użytkownika.
* **isAdmin** – Pole zawierające wartość logiczną określające uprawnienia użytkownika.
* **Picture** – Pole zawierające zdjęcie profilowe użytkownika.
* **createdAt** – Pole określające datę utworzenia użytkownika.

# Proces implementacji

## Wprowadzenie

Aplikacja została zbudowana w stylu SPA(Single Page Application), który oferuje biblioteka języka JavaScript – React.js. Dzięki technologii SPA aplikacja nie potrzebuje konieczności przeładowywania strony aby zaktualizować jakiekolwiek zmiany. Taka technologia jest bardzo przyjazna dla użytkownika korzystającego z aplikacji, ponieważ charakteryzuję się dużą elastycznością oraz przyjaznym interfejsem. Przykładem takiej aplikacji jest popularna platforma Netflix. Więcej informacji na temat biblioteki React.js można znaleźć w następującej książce[1].

## Środowisko i narzędzia programistyczne

Środowisko wykorzystane podczas implementacji całej aplikacji z uwzględnieniem aplikacji serwerowej to IDE WebStorm w wersji 2021.3. Wspiera ono w pełni proces implementacji aplikacji webowych. Wspomniane wcześniej środowisko pozwala na jednoczesne uruchomienie aplikacji klienckiej oraz serwerowej przy pomocy dwóch oddzielnych terminali.

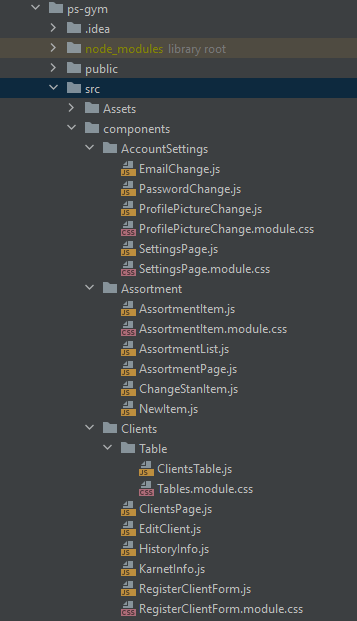
Baza danych wykorzystana w projekcie to MongoDB w wersji 5.0.5 – bez relacyjna, która przechowuje rekordy w tabelach jako dokumenty w stylu JSON. Zamiast jawnych relacji między tabelami nadal można powiązać je ze sobą. Proces ten polega na przechowywaniu referencji między jedną a drugą tabelą, wówczas takie wiązanie nazywa się wirtualną relacją.

Platforma serwerowa gdzie został napisany szkielet aplikacji(interfejs API) pośredniczący między aplikacją a bazą danych to Express.js

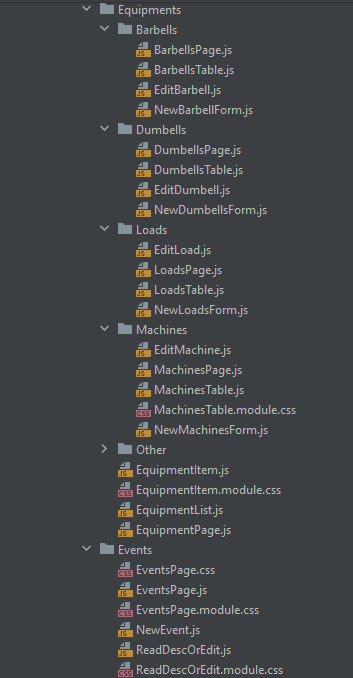
Modele produktów zostały zrealizowane przy pomocy aplikacji przeznaczonej do modelowania grafiki 3D – Blender. Do zrealizowania diagramu przypadków użycia został wykorzystany program StarUML.

## Komponenty, dodatkowe biblioteki, narzędzia informatyczne

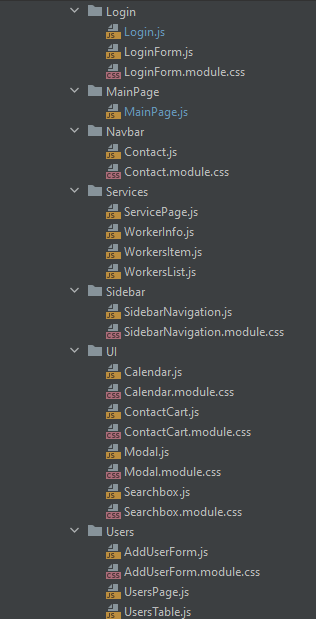
Aplikacja składa się z szeregu komponentów odpowiadających za prawidłowe funkcjonowanie sytemu. Ich zadaniem jest generowanie interfejsu użytkownika, przetwarzanie danych oraz w zależności od przypadku, ponowne wygenerowanie komponentu na przykład w momencie dodania nowego klienta do tabeli. Niewątpliwą zaletą komponentów jest możliwość komunikacji między nimi w zależności od wymagań aplikacji.



Ilustr.3: Struktura komponentów.



Ilustr.4: Struktura komponentów, ciąg dalszy.



Ilustr.5: Struktura komponentów, ciąg dalszy.

Dodatkowe biblioteki wykorzystane w projekcie:

* React hooks: useState, useEffect, useContext, useNavigate.
* React-dom(Document Object Model): React Router, React Portals, React Links.
* Bootstrap.
* Css.
* React Material.

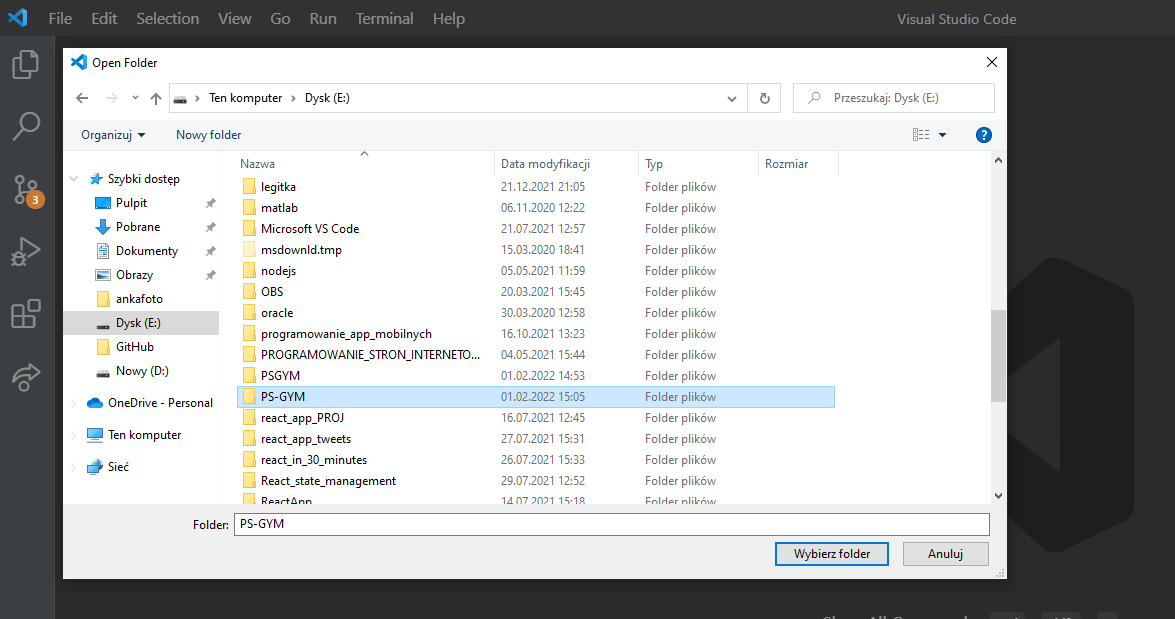
Postępy jak i cała aplikacja była przechowywana za pośrednictwem serwisu GitHub.

# Prezentacja aplikacji.

## 3.1 Uruchomienie aplikacji klienckiej

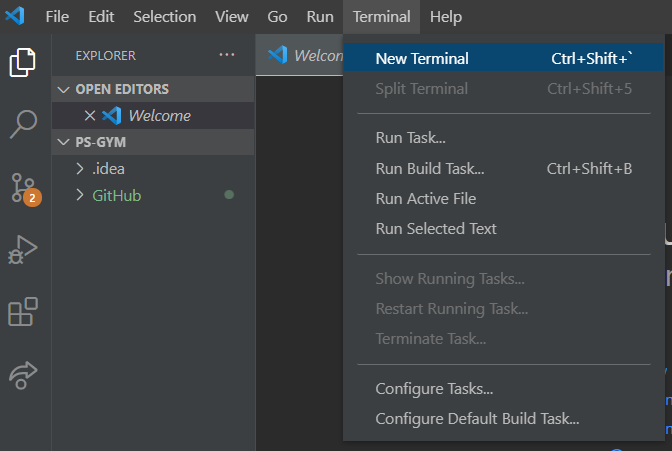
Aby uruchomić aplikację trzeba uwzględnić podział na aplikację kliencką oraz aplikację pośredniczącą między serwerem a bazą danych.

W pierwszym kroku należy uruchomić środowisko programistyczne(np. Visual Studio Code), a następnie wybrać folder PS-GYM.



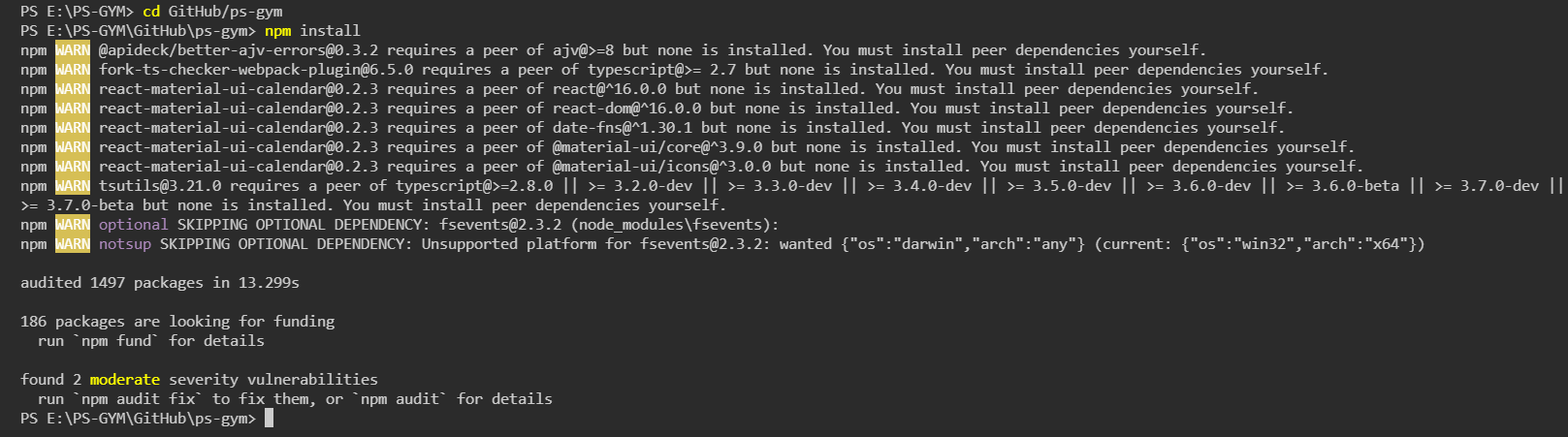
Ilustr.6: Wybór folderu zawierającego aplikację.

Następnie należy skorzystać z zakładki „Terminal” i wybrać opcję „New Terminal”.



Ilustr.7: Uruchomienie nowego terminala.

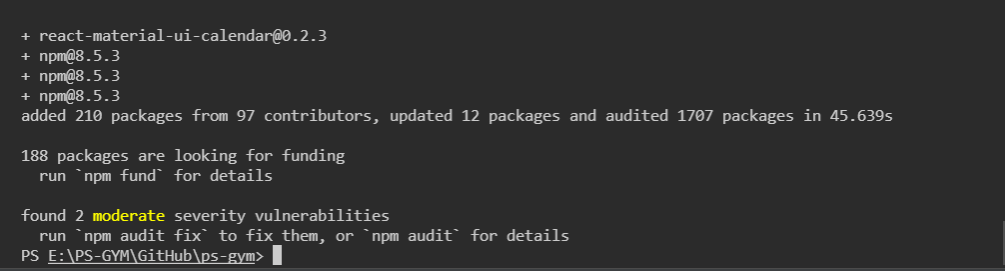
Teraz trzeba dostać się do folderu z kodem aplikacji. Uzyskamy to za pomocą polecenia: cd GitHub/ps-gym. Następnie należy zainstalować szereg pakietów biblioteki React przy pomocy polecenia: npm install.



Ilustr.8: Instalacja pakietów biblioteki React.js.

W następnym kroku wymagane jest zainstalowanie biblioteki stylów React Material przy pomocy poleceń:

* npm install @mui/material,
* npm install @emotion/react,
* npm install @emotion/styled,
* npm install react-material-ui-calendar.



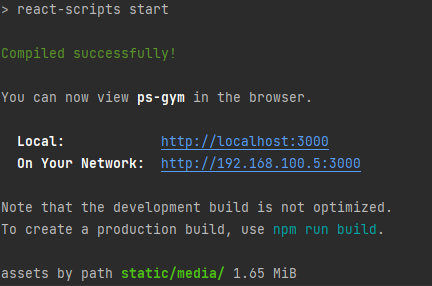
Ilustr.9: Instalacja biblioteki React Material.

Do uruchomienia aplikacji klienckiej brakuje już tylko biblioteki stylów bootstrap. W celu instalacji należy wydać polecenie: npm install react-bootstrap [bootstrap@5.1.3](mailto:bootstrap@5.1.3).



Ilustr.10: Instalacja biblioteki arkuszy stylów css bootstrap.

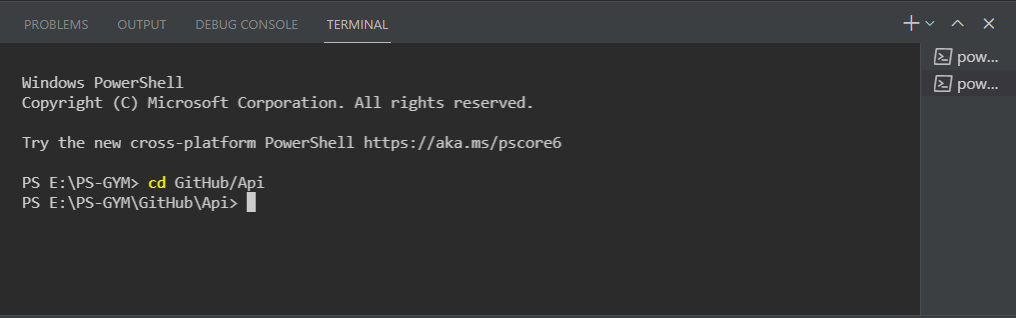
Po zainstalowaniu wszelkich potrzebnych bibliotek należy wydać polecenie npm start w celu uruchomienia aplikacji.



Ilustr.11: Uruchomienie aplikacji klienckiej - rezultat.

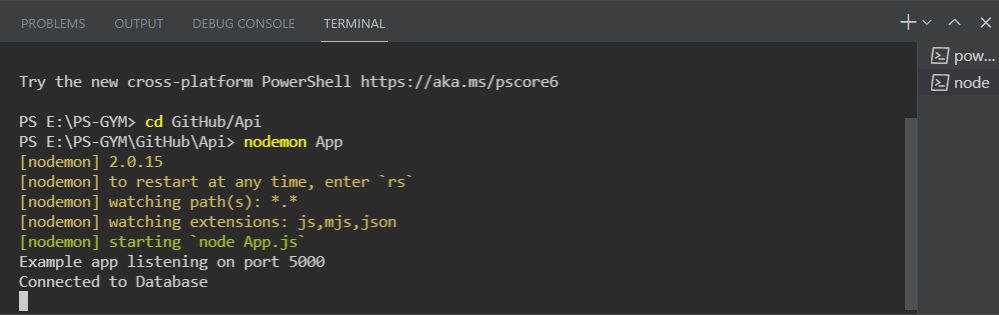
## 3.2 Uruchomienie aplikacji serwerowej

Pierwszym krokiem w celu uruchomienia aplikacji jest pobranie środowiska uruchomieniowego Node.js ze strony <https://nodejs.org/en/>. Następnie należy otworzyć nowy terminal w już uruchomionym wcześniej środowisku programistycznym, po czym przejść do katalogu z serwerem. Aby to zrobić należy wydać polecenie: cd GitHub/Api.



Ilustr.12: Przejście do katalogu z serwerem.

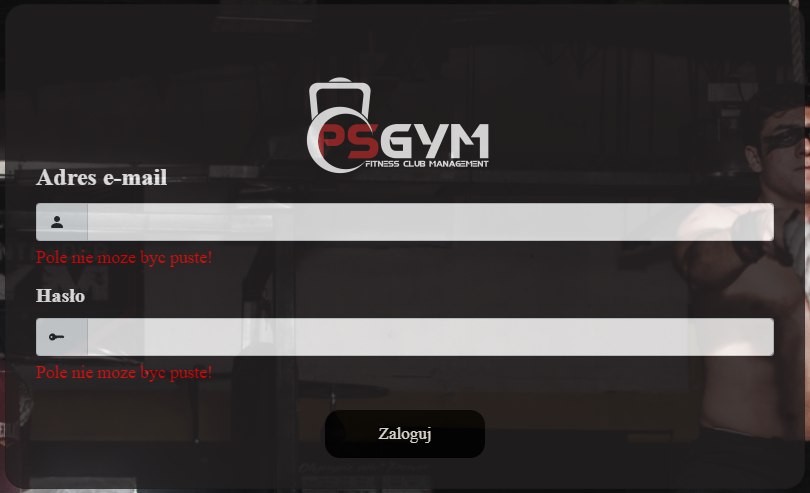
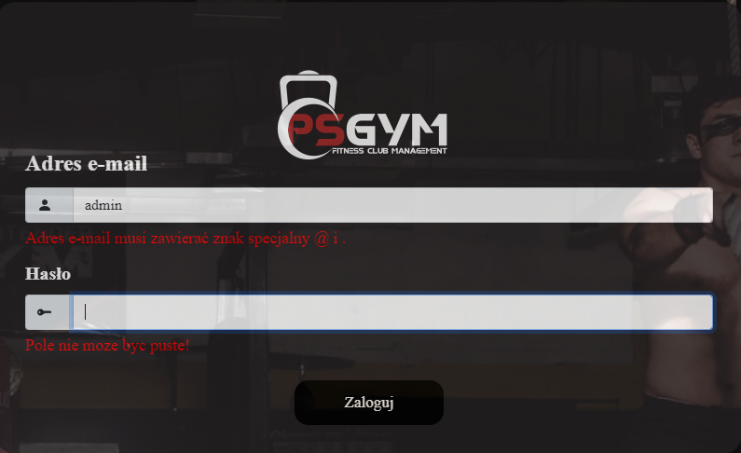
Następnie należy globalnie zainstalować narzędzie uruchomieniowe dzięki poleceniu npm install -g nodemon, a następnie wpisać polecenie nodemon App. Terminal powinien wyświetlić następujący rezultat.



Ilustr.13: Uruchomienie aplikacji serwerowej - rezultat.

## 3.3 Logowanie do aplikacji

Panel logowania do aplikacji składa się z dwóch pól tekstowych. Wymaga wprowadzenia adresu e-mail oraz hasła. Po zatwierdzeniu formularza przyciskiem „Zaloguj” dane użytkownika są przesyłane na serwer w celu zweryfikowania poprawności danych. Jeżeli dane są poprawne to następuję przekierowanie do aplikacji. Oczywiście formularz logowania jest zabezpieczony przed zatwierdzeniem chociażby pustego formularza. Adres e-mail jest sprawdzany po stronie klienckiej pod względem wymagań składniowych, takich jak znak specjalny „@” oraz „.”. Natomiast hasło użytkownika wymaga wprowadzenia przynajmniej jednego znaku specjalnego, dużej litery oraz co najmniej 6 znaków.

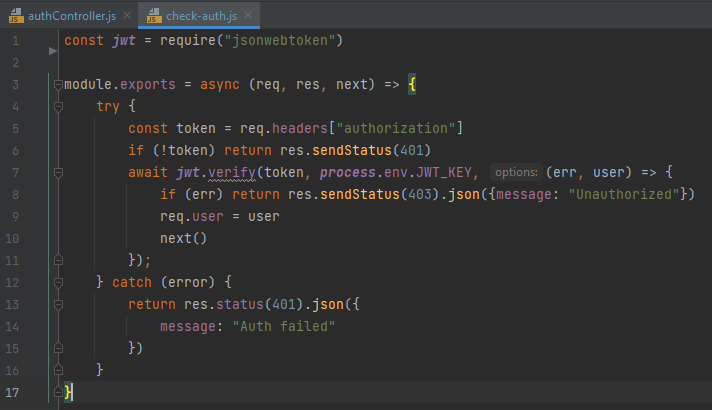


Ilustr.14: Walidacja pustych pól formularza Ilustr.15: Walidacja niepoprawnego adresu e-mail



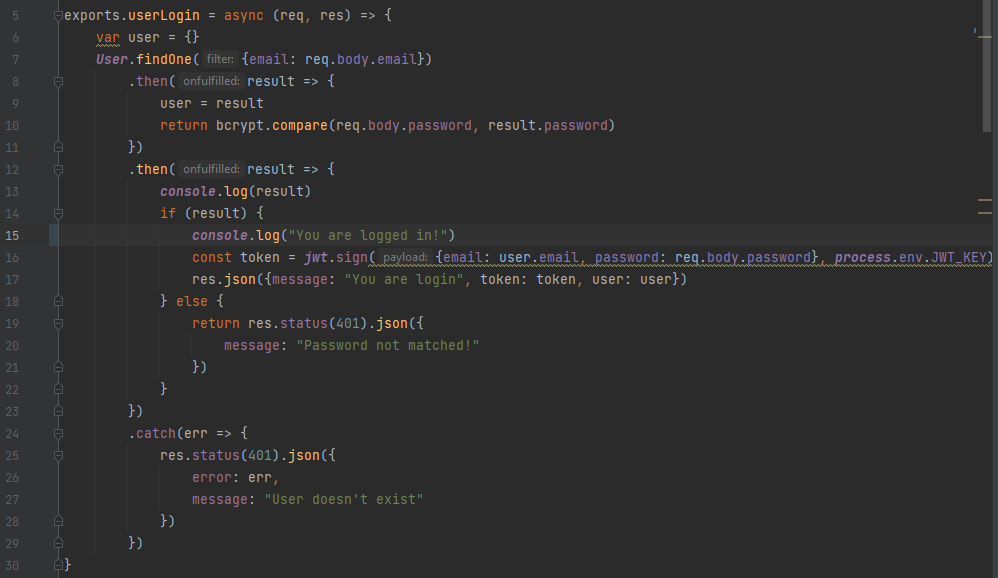
Ilustr.16: Panel logowania do aplikacji po wprowadzeniu poprawnych danych.

W momencie zatwierdzenia formularza następuje wcześniej już wspomniana weryfikacja przez serwer. Implementacja tego mechanizmu prezentuje się następująco:



Ilustr.17: Kod modułu wykonywanego przed wykonaniem akcji kontrolera.

* Linijka 1: Deklaracja stałej niemodyfikowalnej zmiennej, która korzysta z biblioteki „jsonwebtoken”.
* Linijki 3-17: Deklaracja i jednoczesny eksport modułu, który ma za zadanie weryfikację zgodności tokenu. Jeżeli weryfikacja przebiegnie w sposób pomyślny, wywoływana jest akcja kontrolera za pośrednictwem metody next().

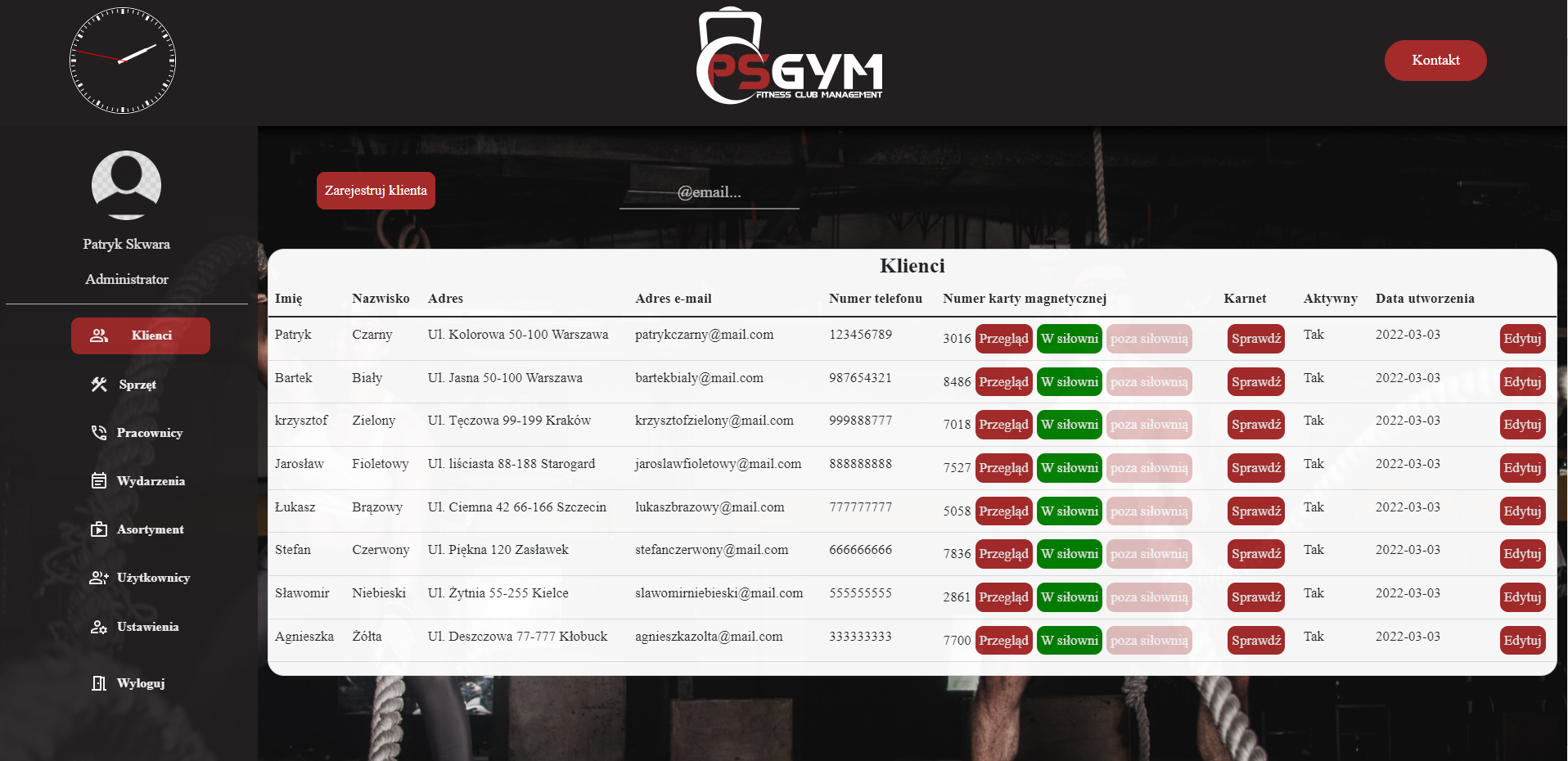


Ilustr.18: Prezentacja kodu akcji kontrolera.

Asynchroniczna akcja kontrolera „userLogin” jest odpowiedzialna za porównanie hasła wprowadzonego przez użytkownika podczas logowania(Ilustr.16) z rzeczywistym hasłem widniejącym w bazie danych. Warto wspomnieć, że mechanizm biblioteki „bcrypt” porównuje już zaszyfrowane hasła(linijka 10). Jeżeli w rezultacie hasła okażą się dokładnie takie same, wyświetlany jest komunikat dla programisty(linijka 15), a następnie generowany jest token dostępowy. W odpowiedzi z serwera w aplikacji klienckiej otrzymywany jest komunikat, token oraz dane aktualnie zalogowanego użytkownika w celu dalszego zarządzania tymi danymi(linijka 17). W przeciwnym razie otrzymamy kod statusu błędu oraz stosowny komunikat.

## 3.4 Interfejs użytkownika

Po zalogowaniu do aplikacji, wyświetlany jest użytkownikowi widok startowy, który prezentuje się następująco:



Ilustr.19: Widok startowy aplikacji.

Koncepcja widoku aplikacji składa się z trzech nadrzędnych komponentów:

1. **Navbar** – Panel poziomy górny przechowujący zegar analogowy, logo aplikacji oraz przycisk, który po kliknięciu wywołuje okno modalne wyświetlające dane kontaktowe twórcy aplikacji.

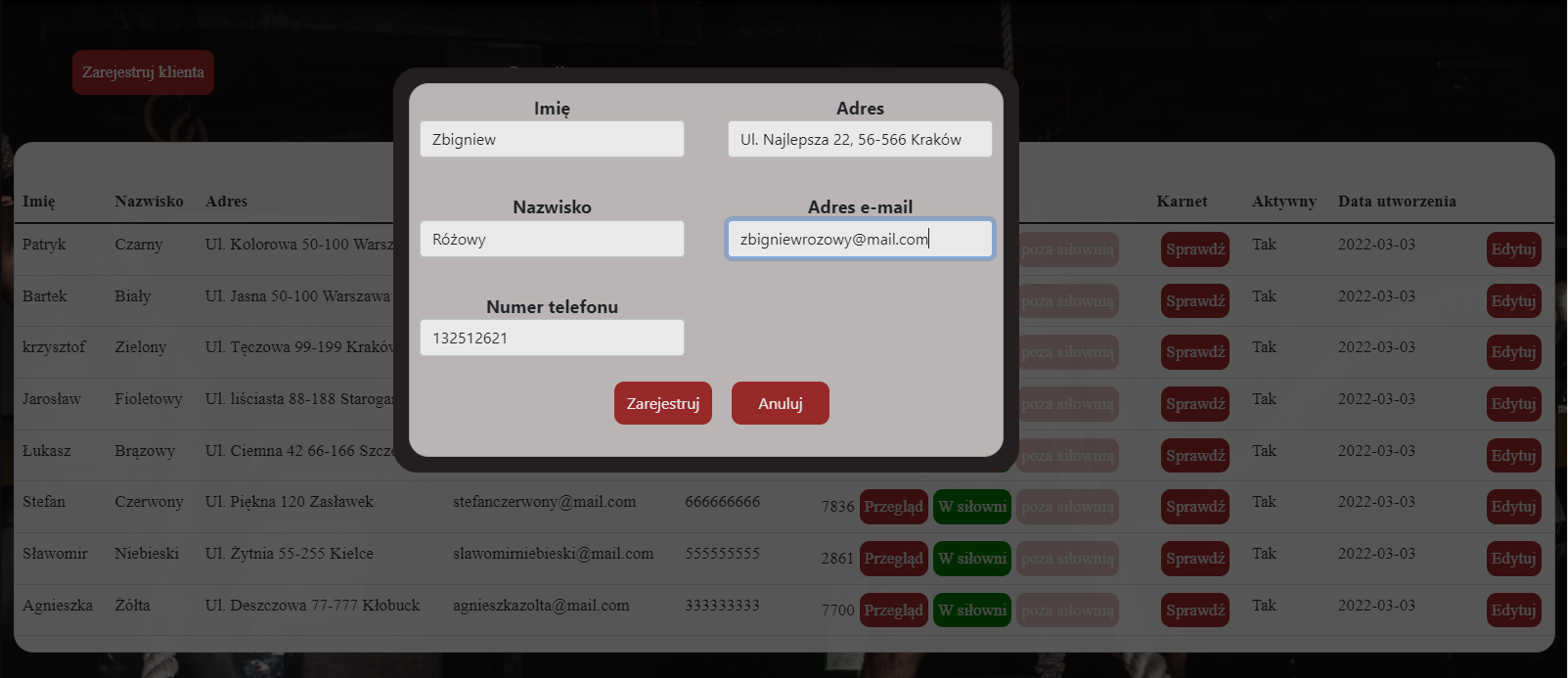


Ilustr.20: Okno modalne „Kontakt”

1. **Sidebar** – Panel pionowy przytwierdzony do lewej strony widoku aplikacji zawierający w sobie zdjęcie profilowe, imię i nazwisko oraz funkcję użytkownika. Poniżej białej poziomej linii znajdują się odnośniki, które po kliknięciu przekierowują użytkownika do wybranej zakładki.
2. **Main page** – Główna przestrzeń aplikacji znajdująca się poniżej panelu górnego i obok panelu bocznego. Jej zadaniem jest wyświetlanie użytkownikowi interfejsu graficznego wybranej zakładki.

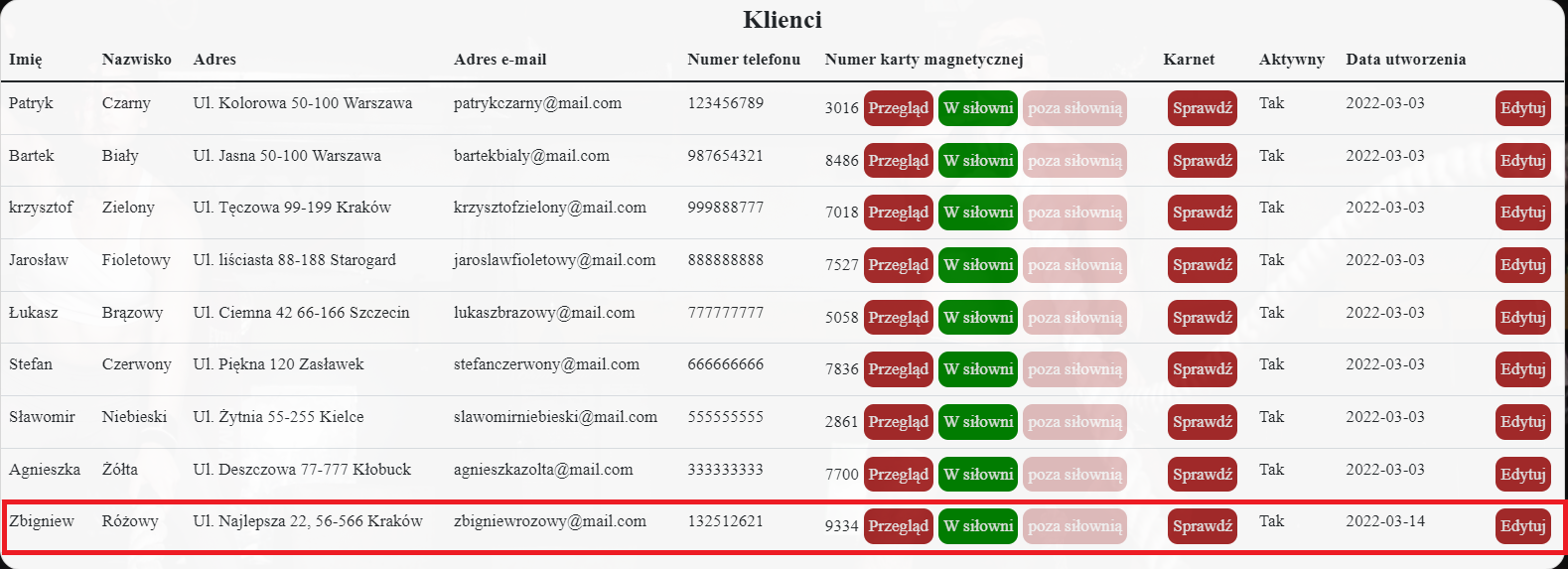
## Zakładka „Klienci”

Zakładka oferuje użytkownikowi aplikacji szereg funkcjonalności w związku z zarządzaniem klientami. Najistotniejszym elementem jest oczywiście ich rejestracja.



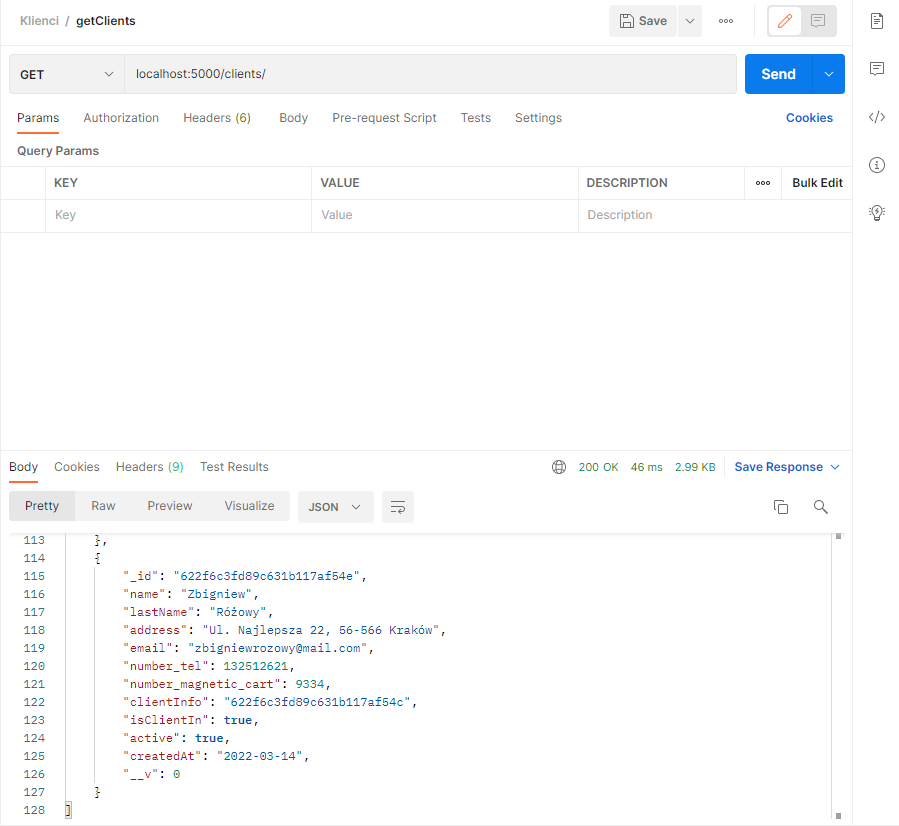
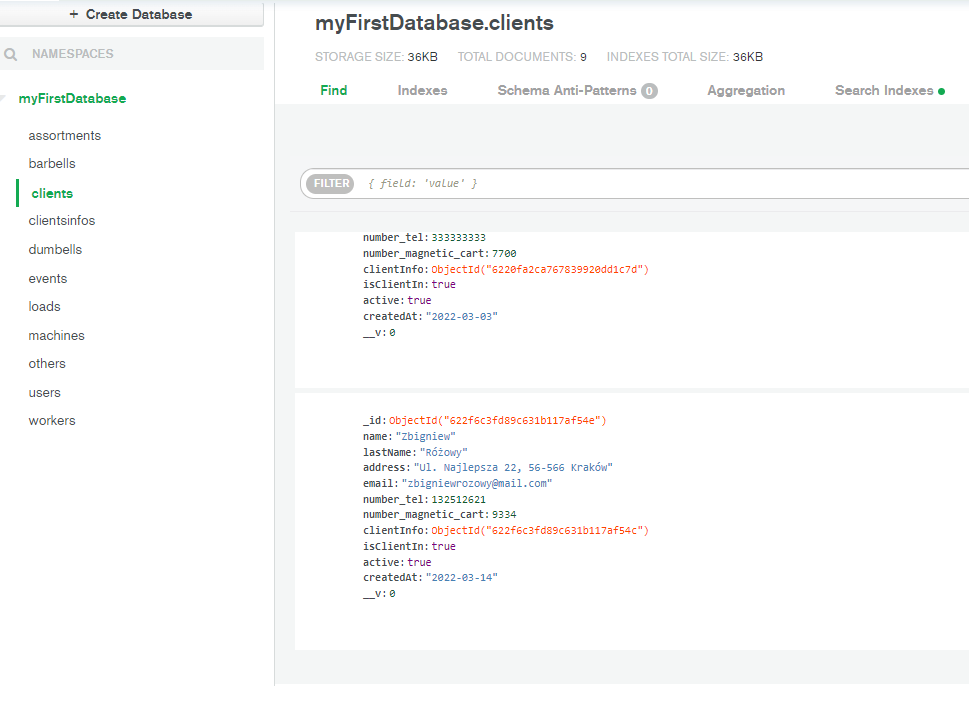
Ilustr.21: Widok okna odpowiedzialnego za rejestrację klienta.

Aby zarejestrować nowego klienta w klubie fitness PSGYM należy wypełnić pola formularza jego podstawowymi danymi. W tym celu należy wywołać okno modalne(ilustr.21) klikając w przycisk „Zarejestruj klienta” znajdujący się nad tabelą z klientami. Po wypełnieniu pól formularza oraz zatwierdzeniu przyciskiem „Zarejestruj” klient zostaje dodany do tabeli oraz do bazy danych klientów.



Ilustr.22: Dodanie nowej pozycji do tabeli Klienci.

Po wykonaniu odpowiedniego zapytania do bazy danych za pośrednictwem aplikacji do testowania interfejsów API oraz aplikacji back-endu – Postman, należy upewnić się czy klient został dodany do bazy danych.



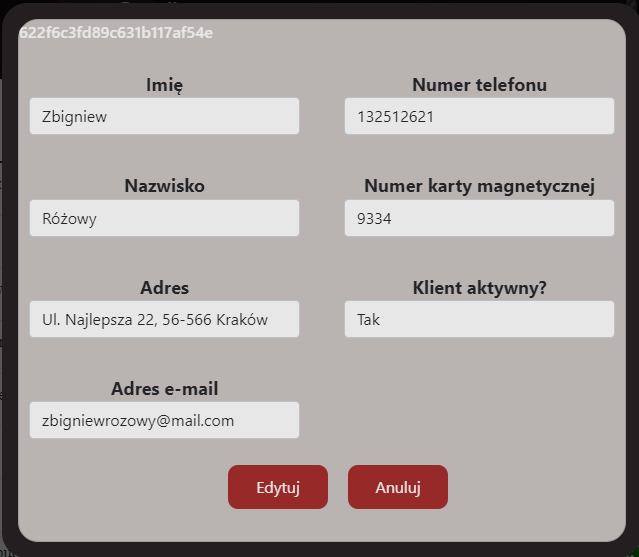
Ilustr.23: Widok nowego klienta w aplikacji Postman. Ilustr.24: Widok nowego klienta w bazie danych.



Ilustr.25: Filtrowanie tabeli klientów.

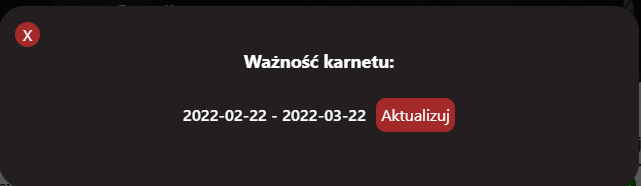
Użytkownik ma możliwość szybkiego wyszukiwania klienta za pomocą wprowadzenia jego adresu e-mail(ilustr.25). Filtrowanie tabeli odbywa się w sposób dynamiczny(wyszukiwarka reaguje na każdy wprowadzony znak i tymczasowo modyfikuje widok tabeli. Element ten został zaimplementowany zgodnie z założeniem, że klientów z czasem będzie wielu, a ręczne szukanie pojedynczej pozycji na przykład w celu aktualizacji karnetu, byłoby męczące i nieoptymalne.

Przycisk „Edytuj” zapewnia możliwość edycji danych osobowych klienta, oraz zmianę jego statusu tj. Aktywny lub nieaktywny. Po jego kliknięciu wyświetlone zostaje poniższe okno.

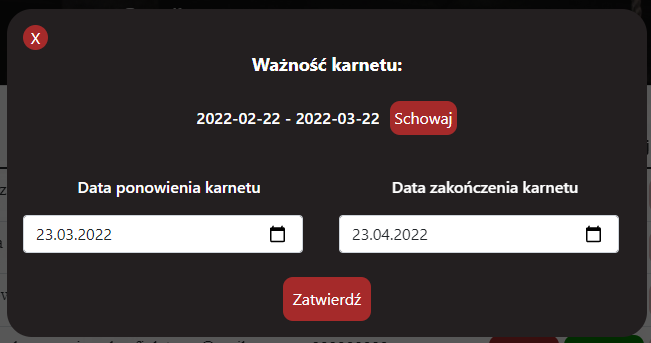


Ilustr.26: Okno edycji klienta.

Istotnym elementem w kwestii zarządzania klientami jest możliwość przeglądu oraz aktualizacji karnetu uprawniającego korzystanie z obiektu PSGYM. Przycisk wywołujący takie okno znajduje się w kolumnie „Karnet”.



Ilustr.27: Widok okna przeglądu karnetu.

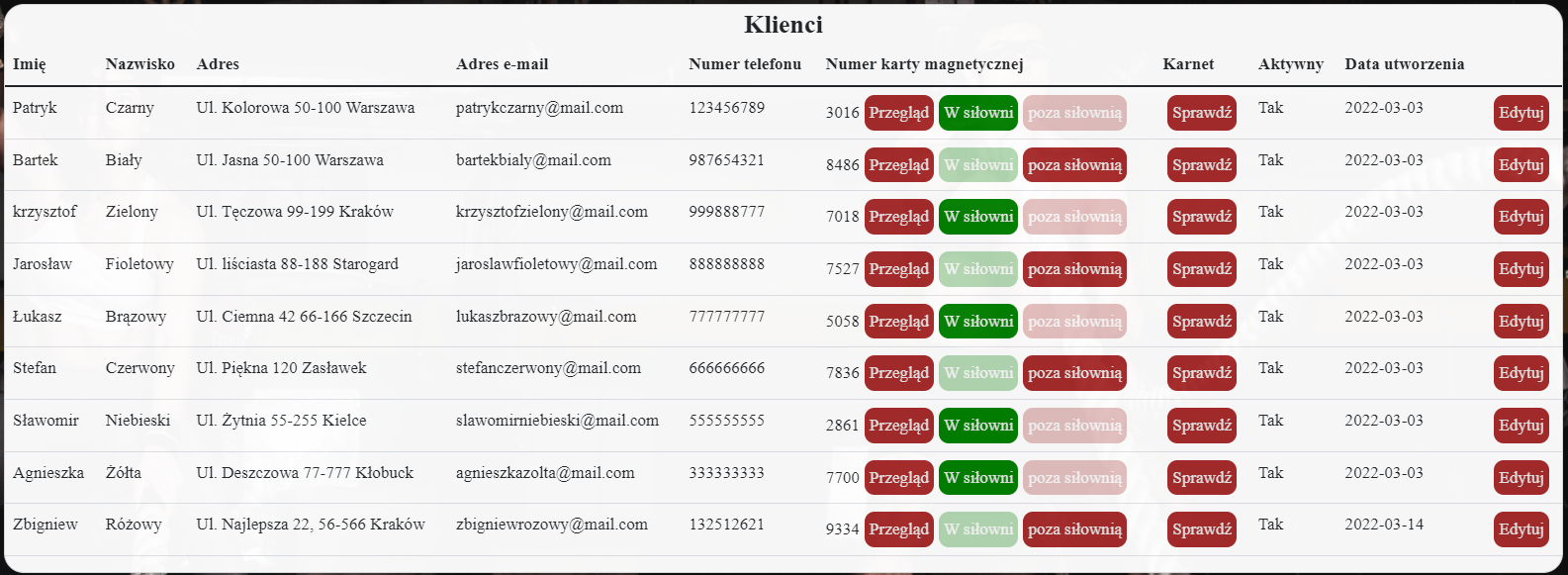


Ilustr.27: Widok okna aktualizacji karnetu.

W związku z faktem posiadania karty magnetycznej przez klientów, została opracowana symulacja działania czytnika do takich kart. Klient używa karty magnetycznej podczas wejścia do obiektu oraz wyjścia z obiektu. W momencie przyłożenia karty do czytnika rejestrowany jest wpis do historii wizyt danego klienta, zawierający aktualną datę oraz godzinę. Taka funkcjonalność została zaimplementowana przede wszystkim w celu zwiększenia bezpieczeństwa, aby w razie strat materialnych takich jak zniszczenia lub kradzieże, mieć możliwość sprawdzenia którzy klienci byli wtedy w obiekcie aby wyegzekwować konsekwencję. W aplikacji, funkcjonalność ta została opracowana w następujący sposób: Kolumna „Numer karty magnetycznej” zawiera trzy przyciski z czego jeden z nich „Przegląd”(ilustr.28) odpowiedzialny jest za wywołanie okna w celu sprawdzenia historii wizyt danego klienta. Natomiast przyciski „W siłowni” oraz „poza siłownią” działają na zasadzie przełączników, czyli drugi z tych przycisków domyślnie jest wyłączony, a uaktywnia się w momencie wciśnięcia pierwszego, a w praktyce po wejściu klienta do obiektu. W takiej sytuacji zostaje dodany nowy wpis do historii wizyt klienta, po czym wciśnięty przycisk zostaje zablokowany(Klient nie może wejść do obiektu gdy już to zrobił), a odblokowany zostaje drugi przycisk oczekujący na wciśnięcie gdy klient opuści obiekt(ilustr.29).



Ilustr.28: Historia wizyt pewnego klienta.



Ilustr.29: Symulacja działania czytnika kart magnetycznych.