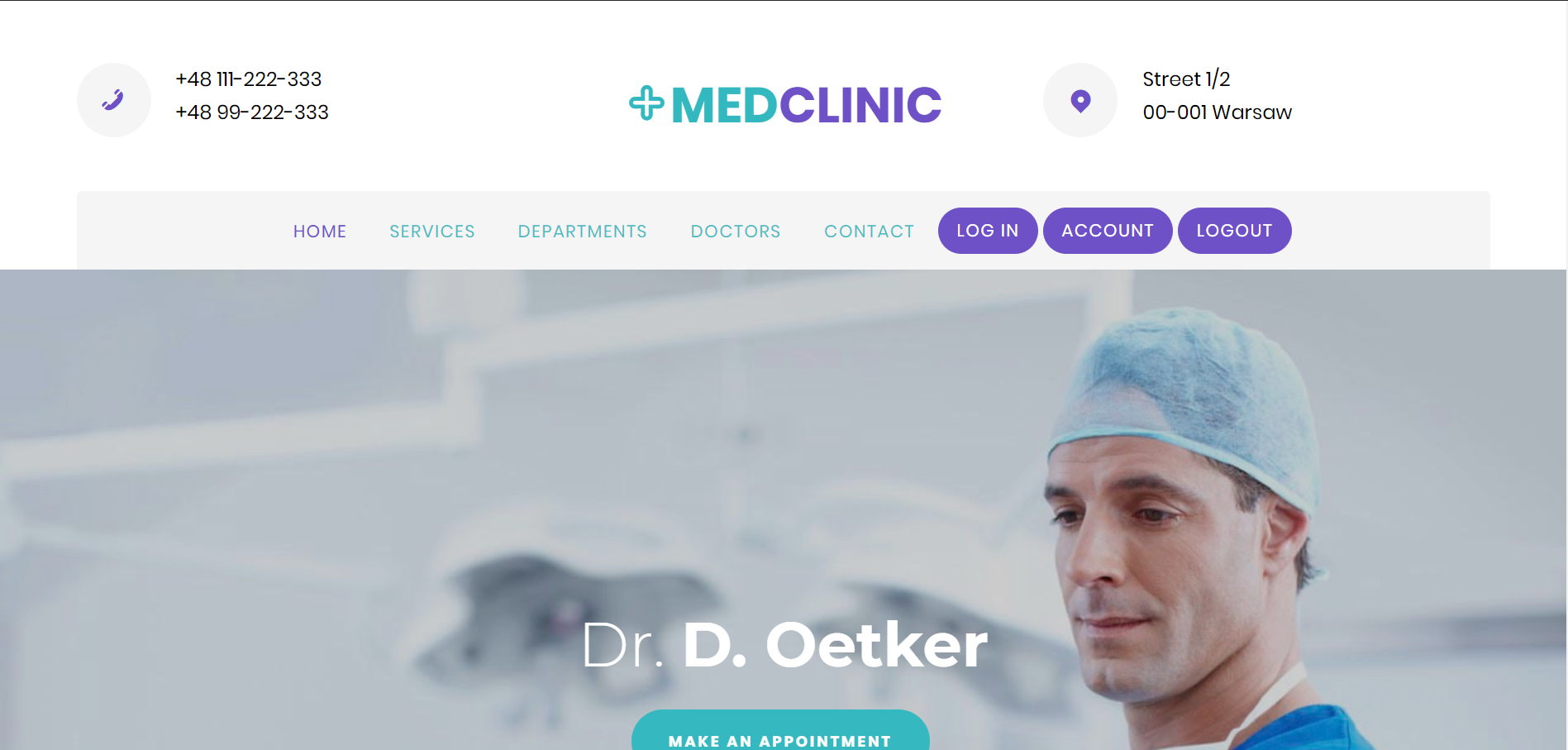
**RAPORT PROJEKTU KLINIKI MEDYCZNEJ**

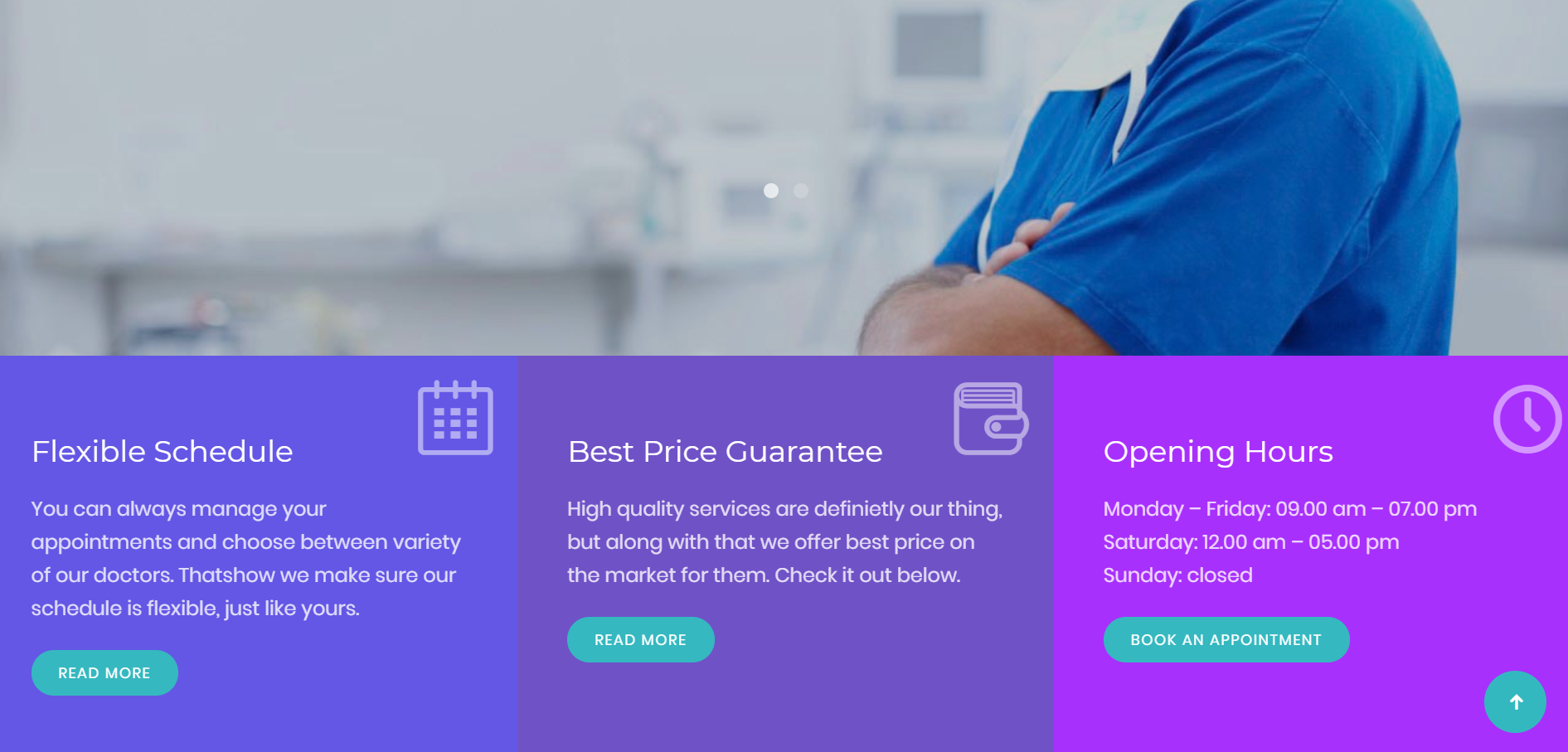
Technologie Informacyjne

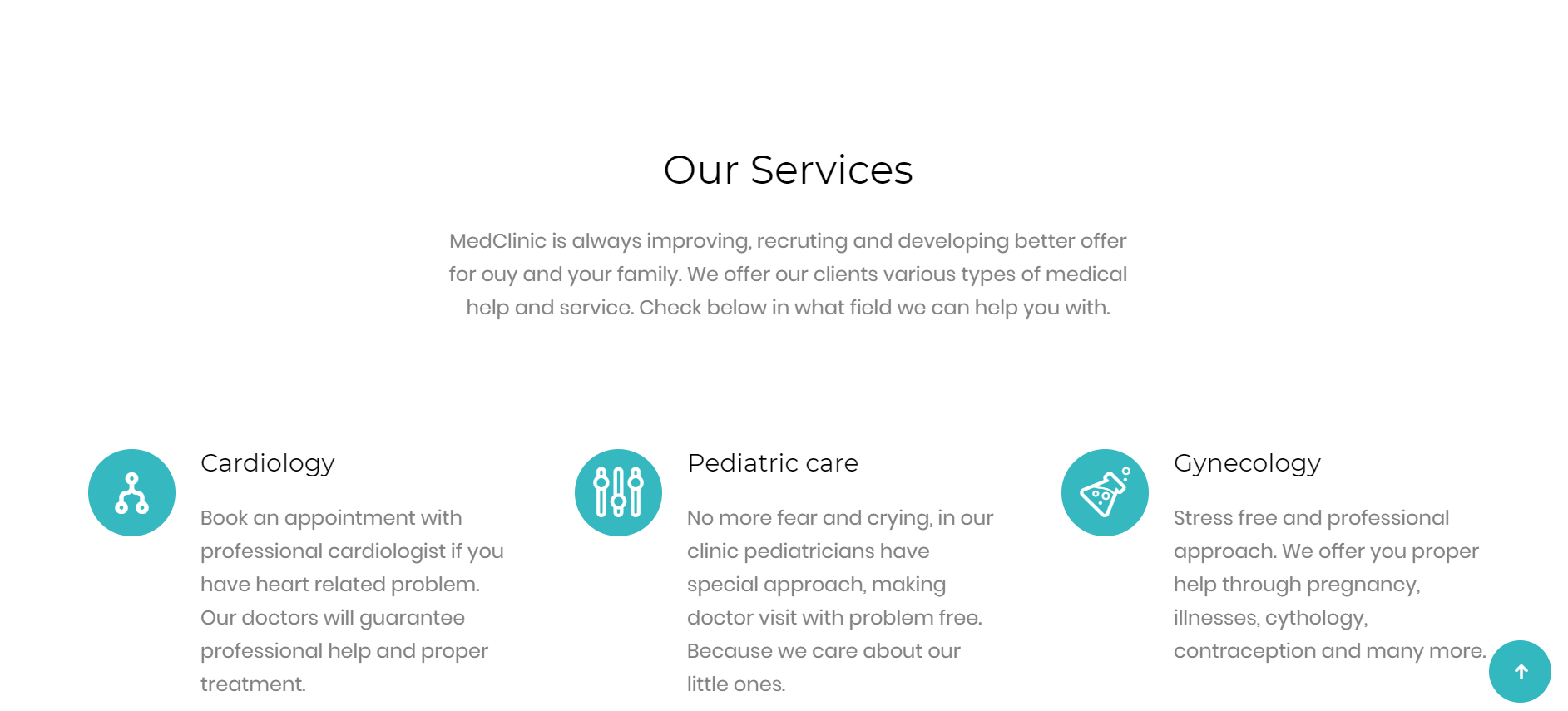
Alicja Karłowicz 244482

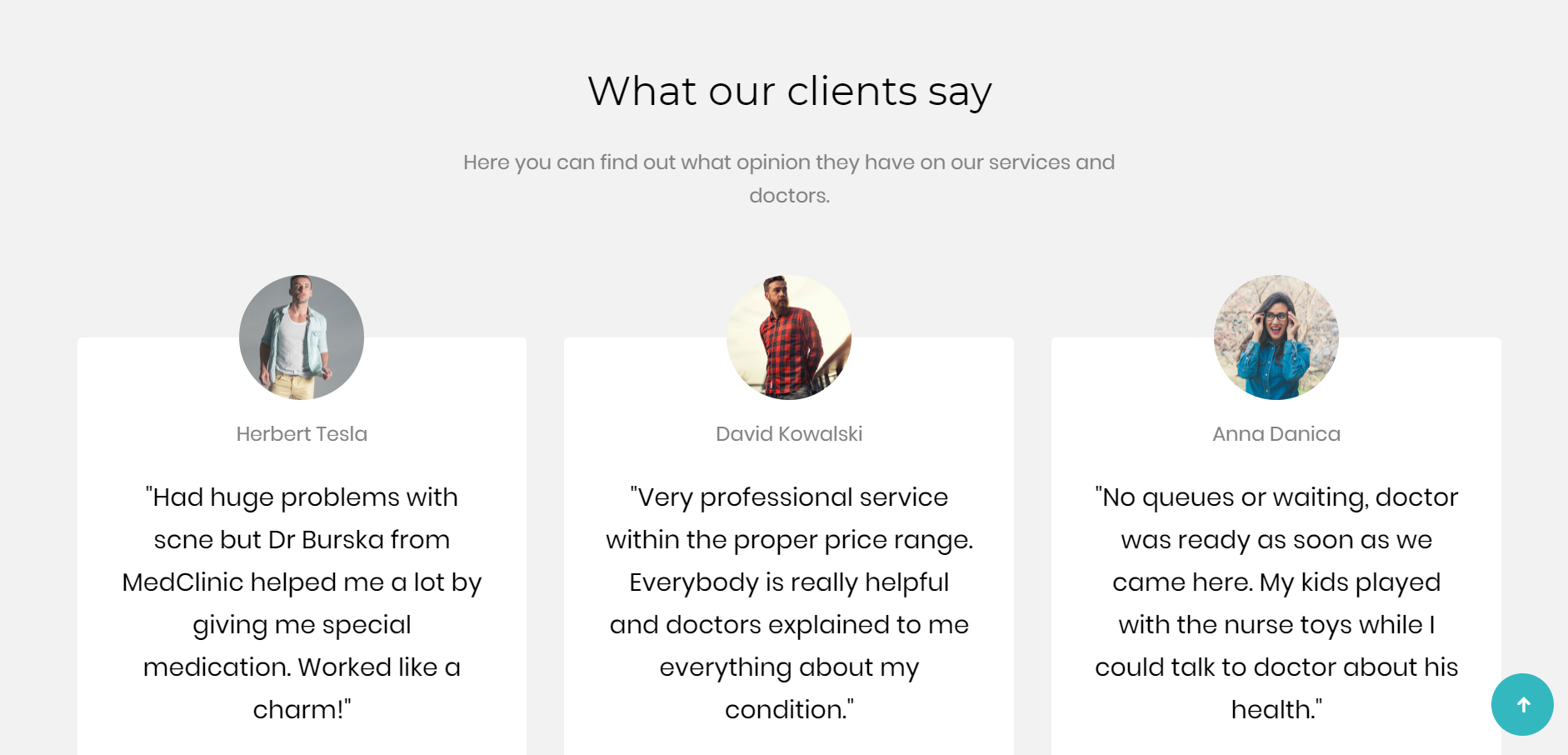
1. **Opis działania projektu z widokami stron (screen)**

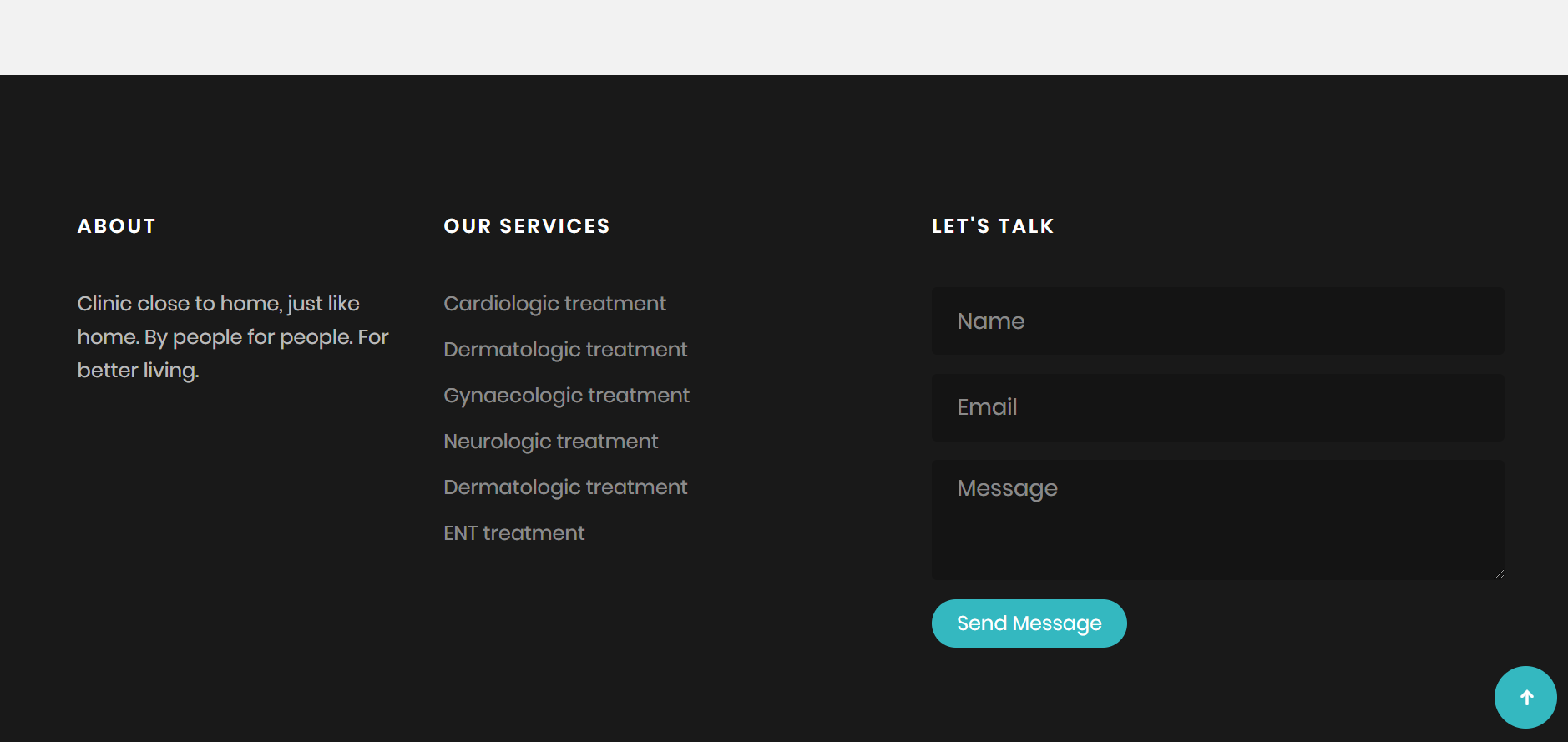
**Strona główna index.html**

****

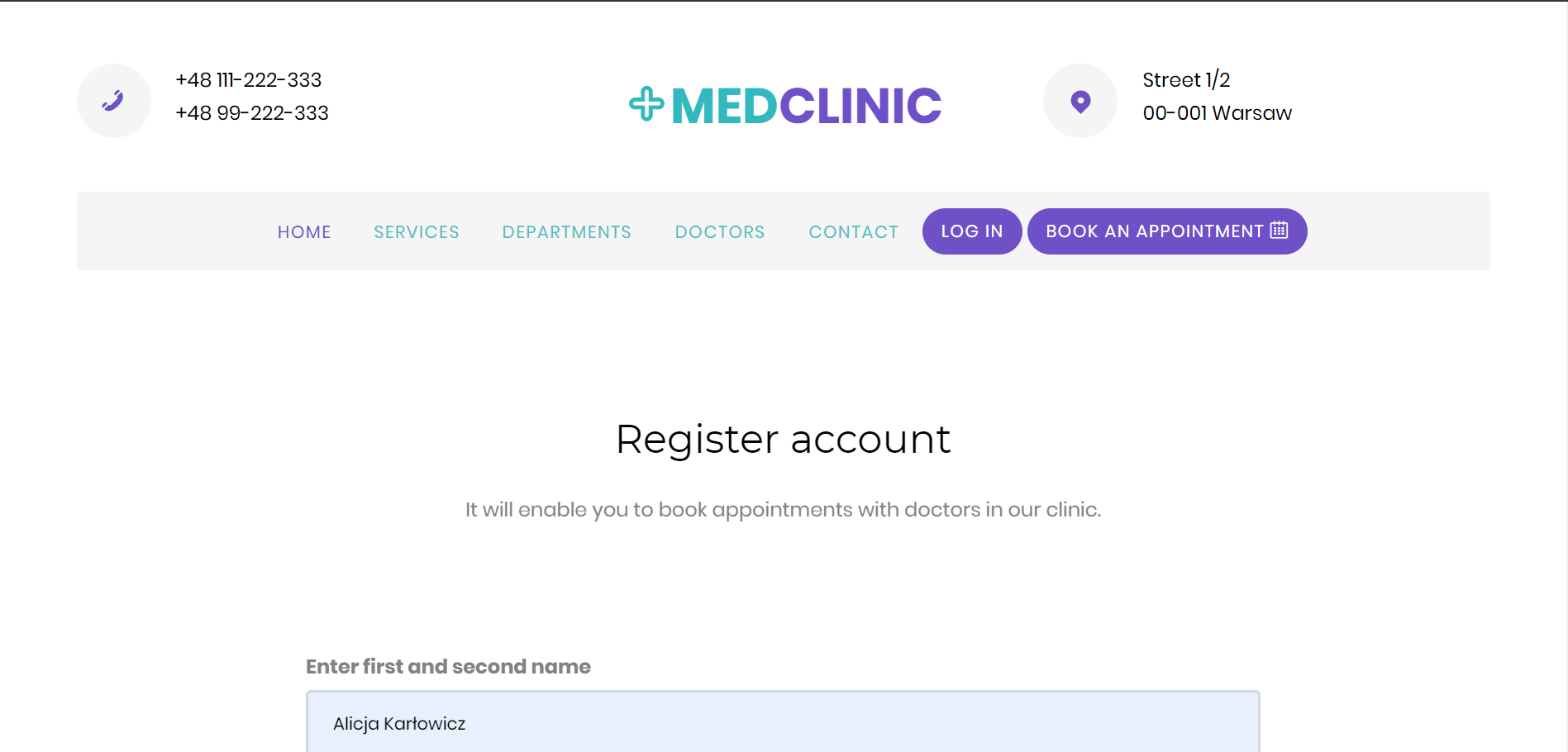
****

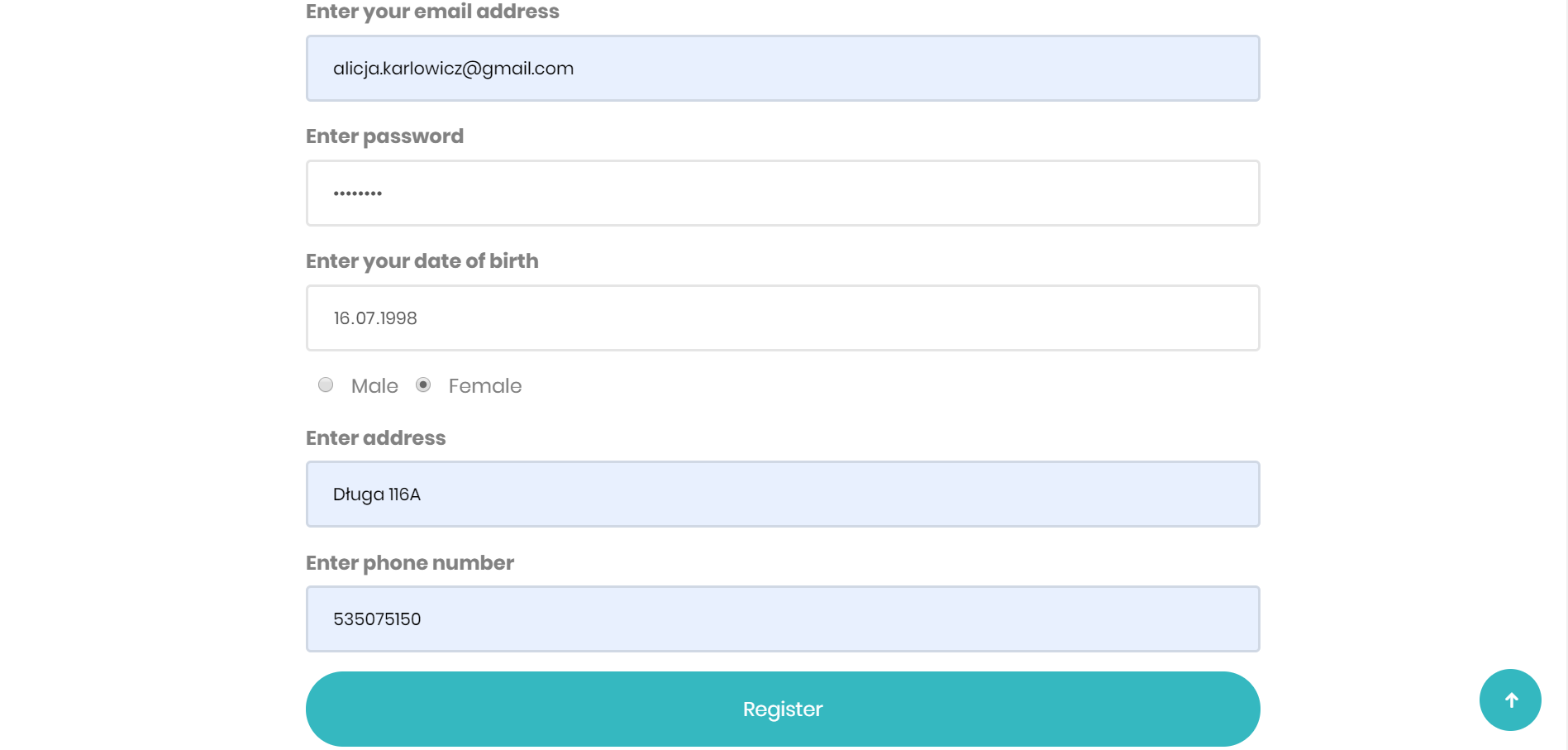
****

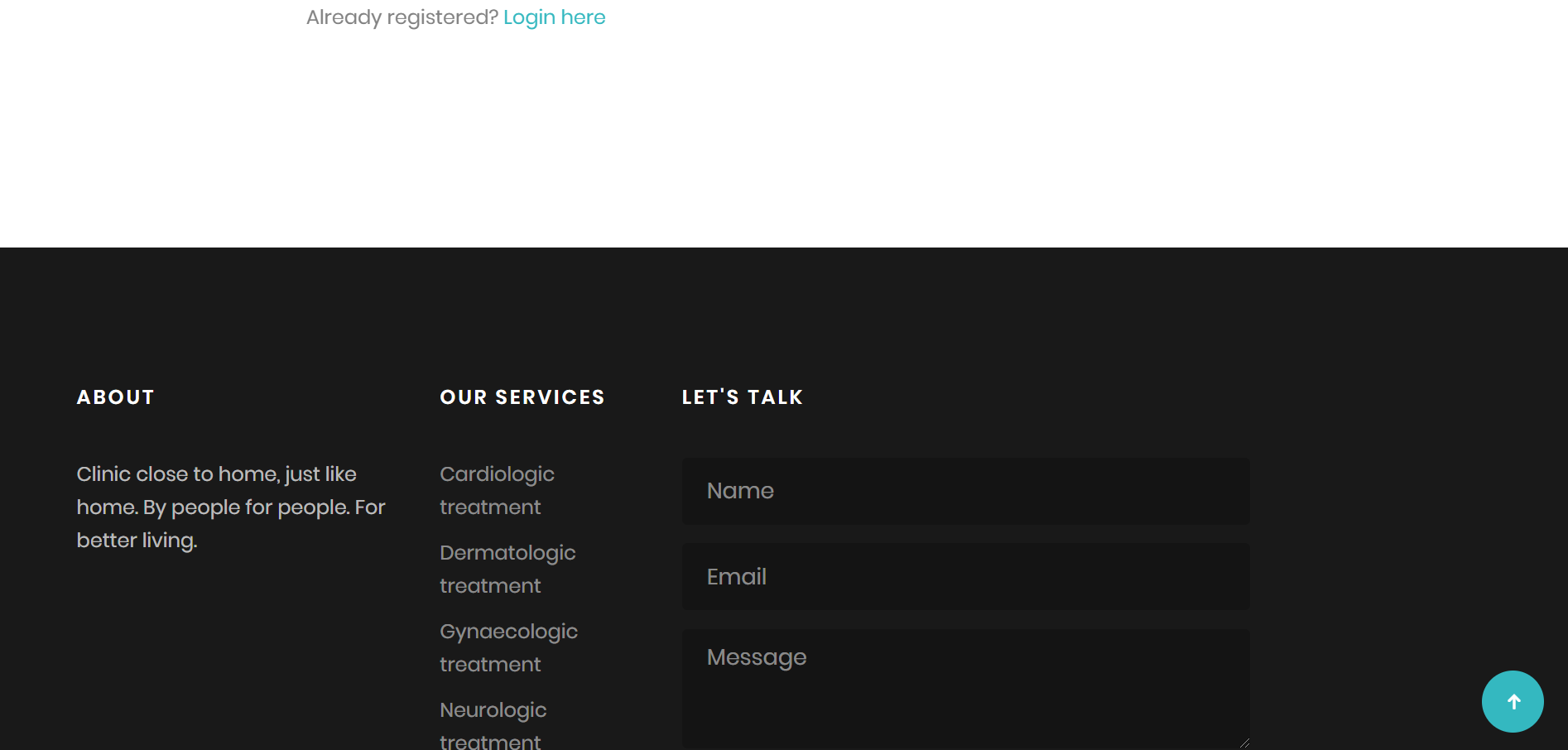
****

****

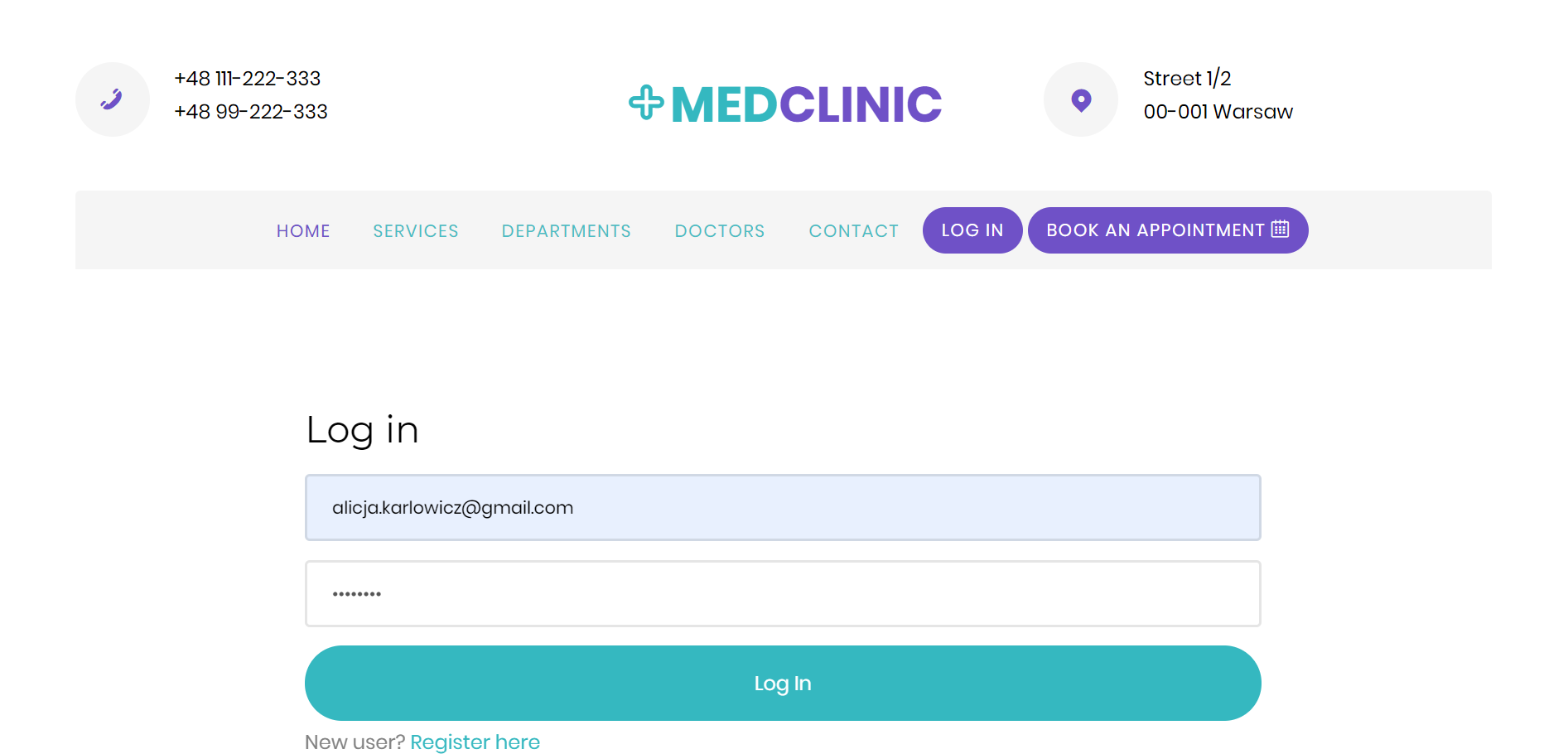
**Okno rejestracji: registration.html. Dostępne po kliknięciu LOG IN na navbarze oraz Register here. W celu zarejestrownaia konta należy wypełnić wszystkie pola i klinąć register. Następnie aplikacja przejdzie do okna logowania, jeśli rejestracja zakończyła się pomyślnie.**

****

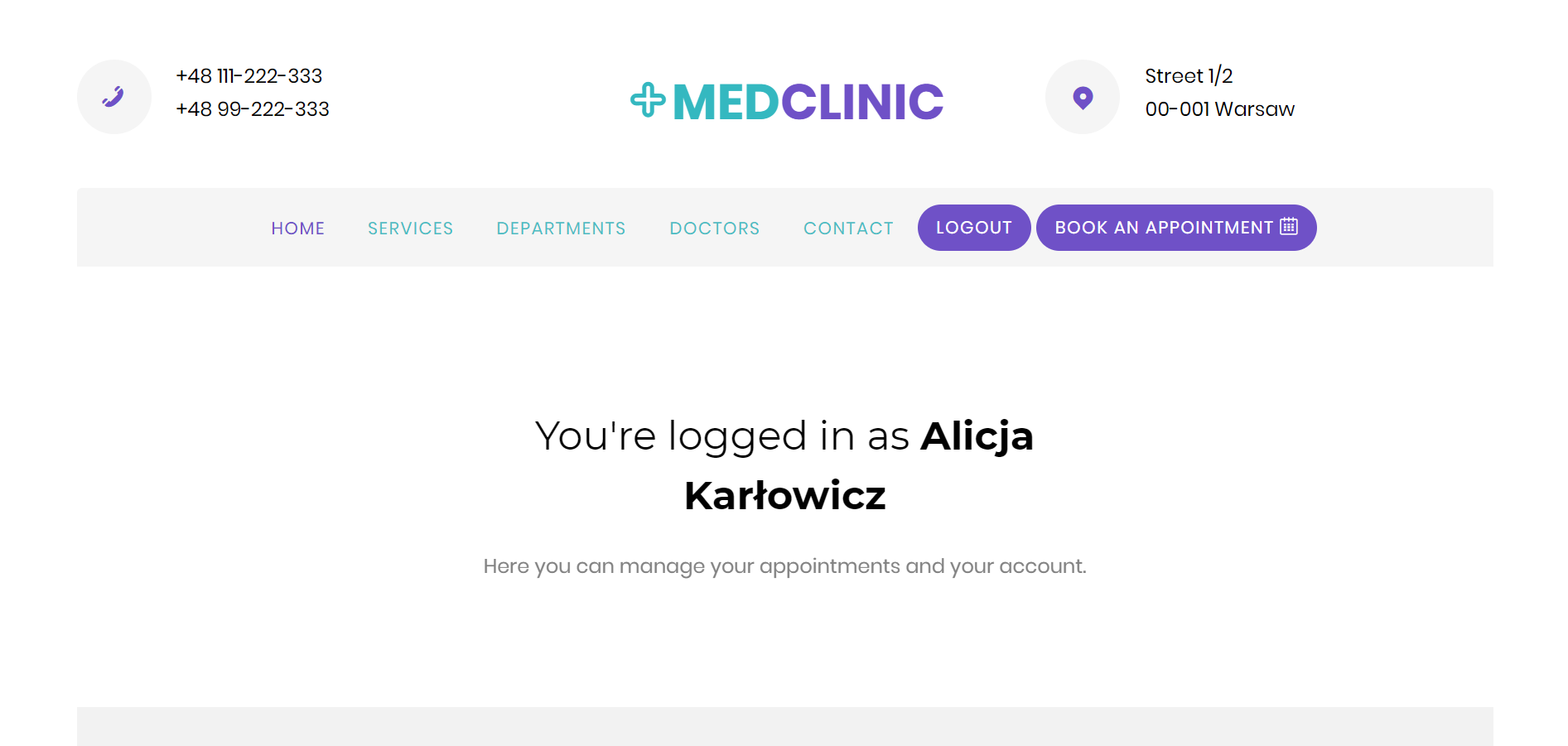
****

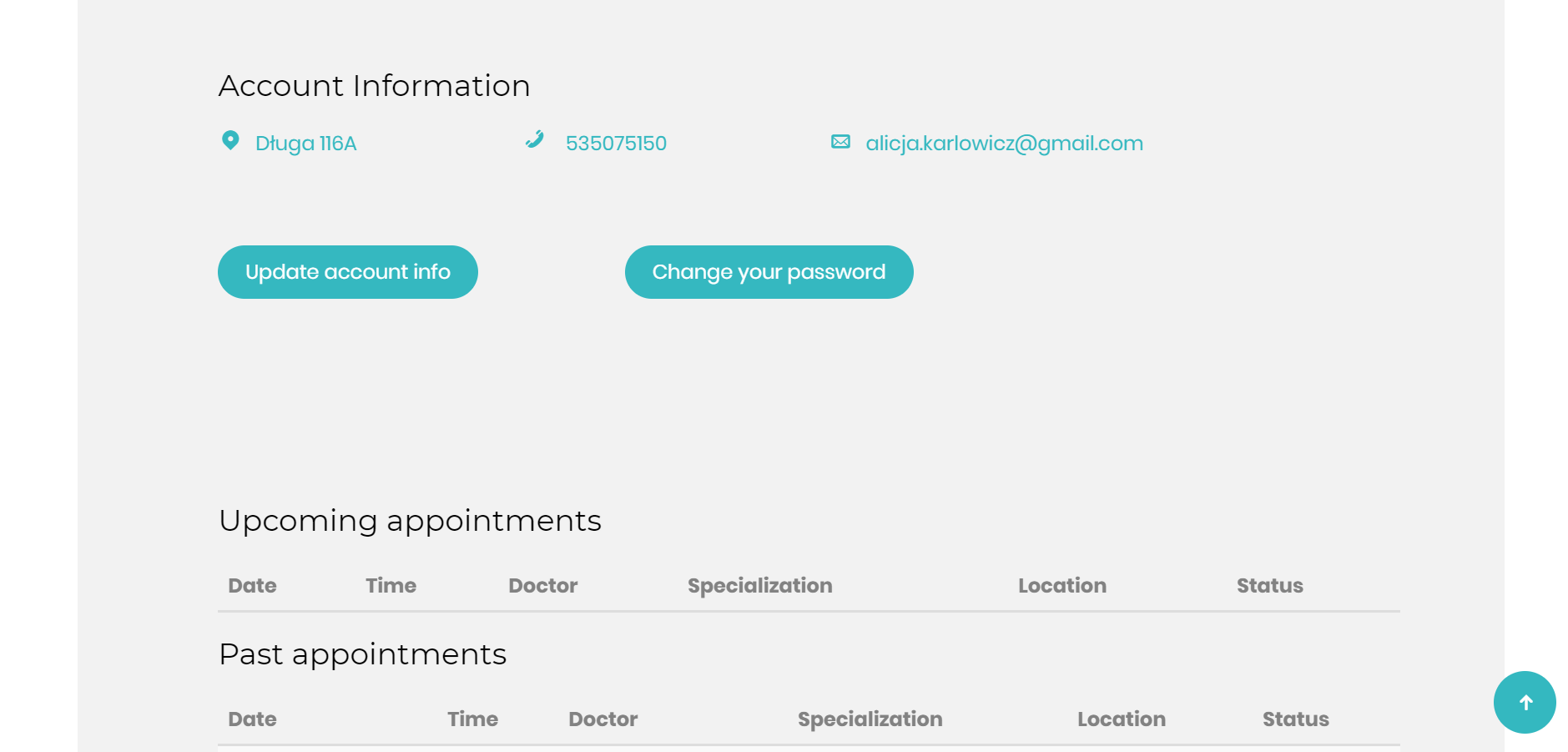
****

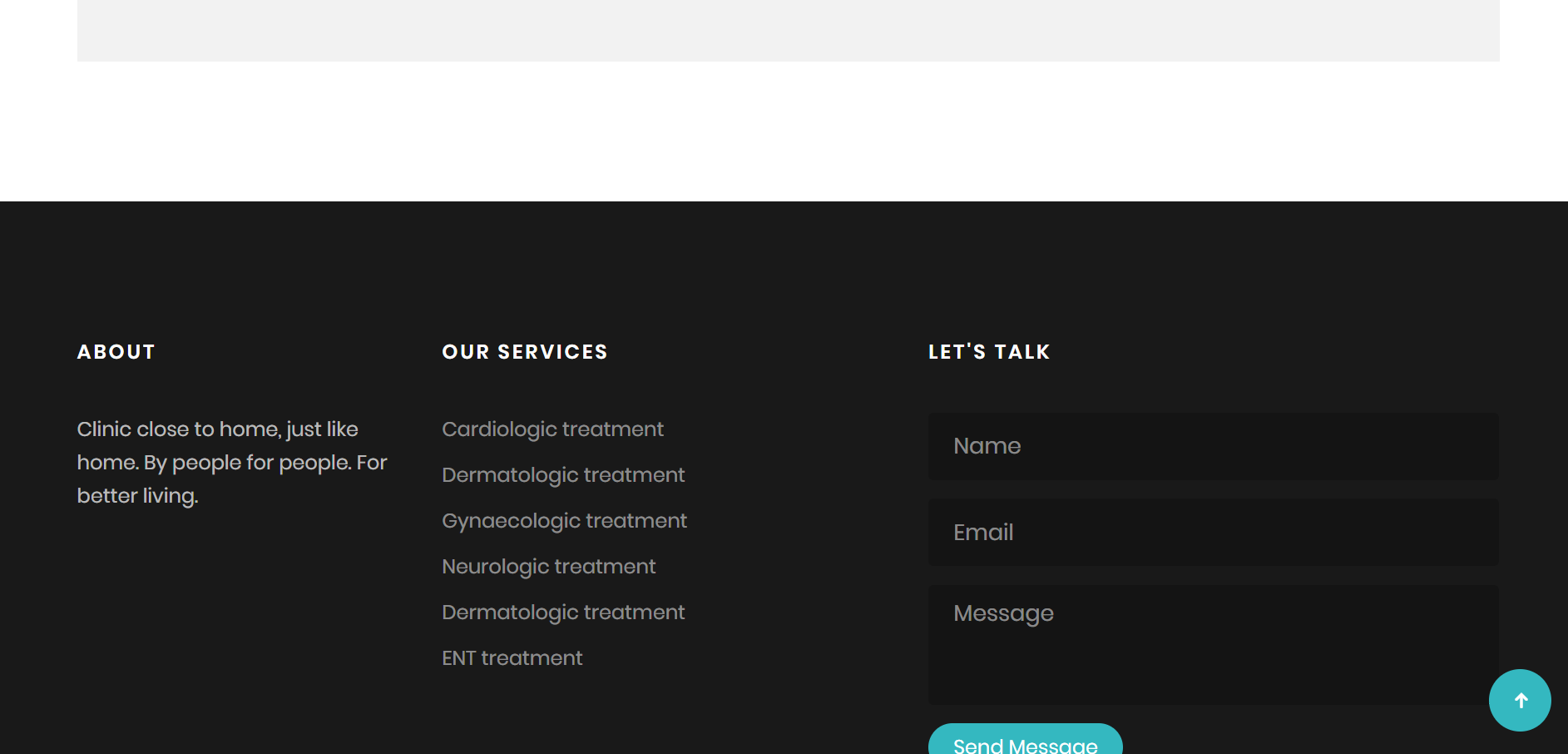
**Okno logowania: login.html. W celu zalogowania na zarejestrowane konto należy podać email rejestracji oraz ustalone hasło i wcisnąć log in.**

****

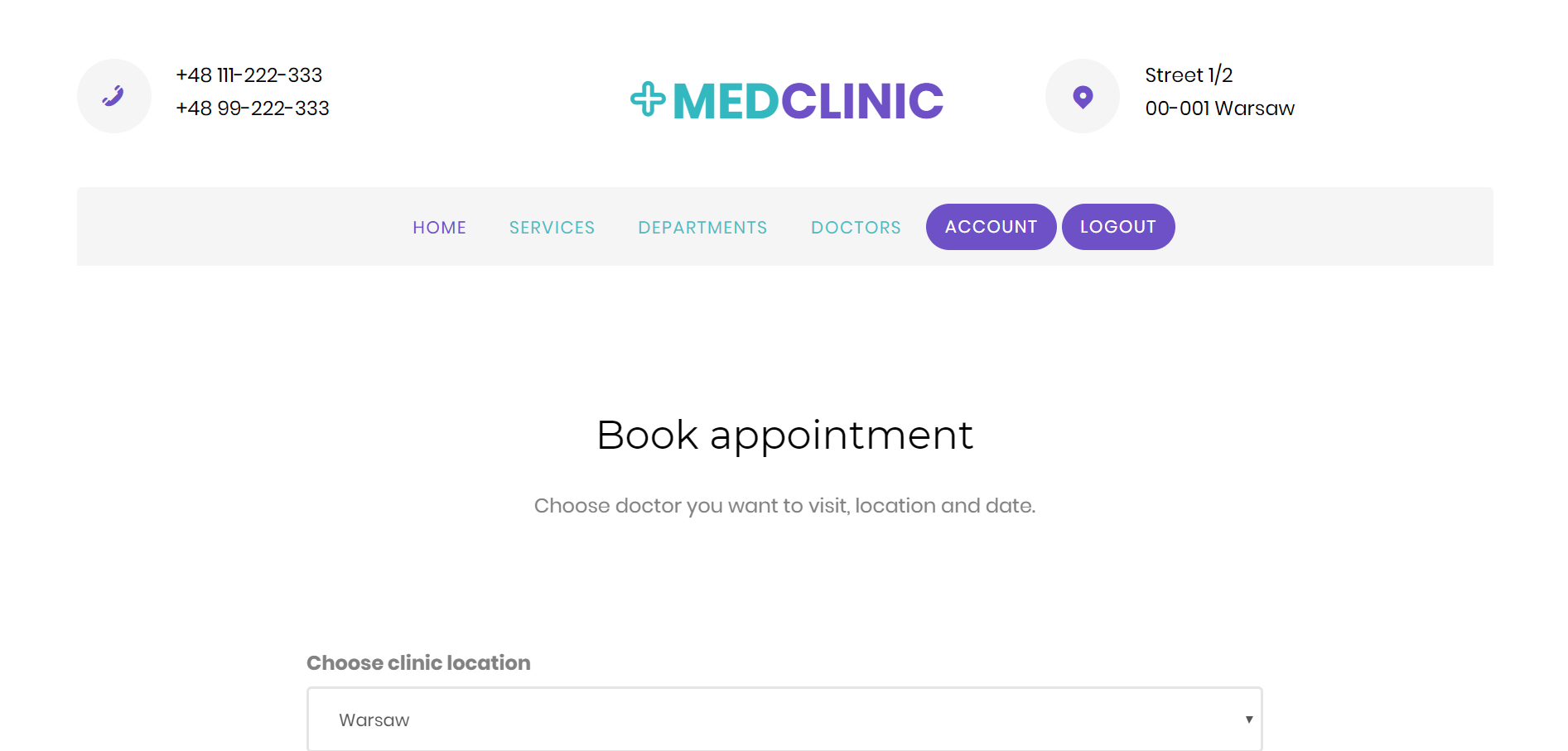
**Okno pacjenta patient-view.html**

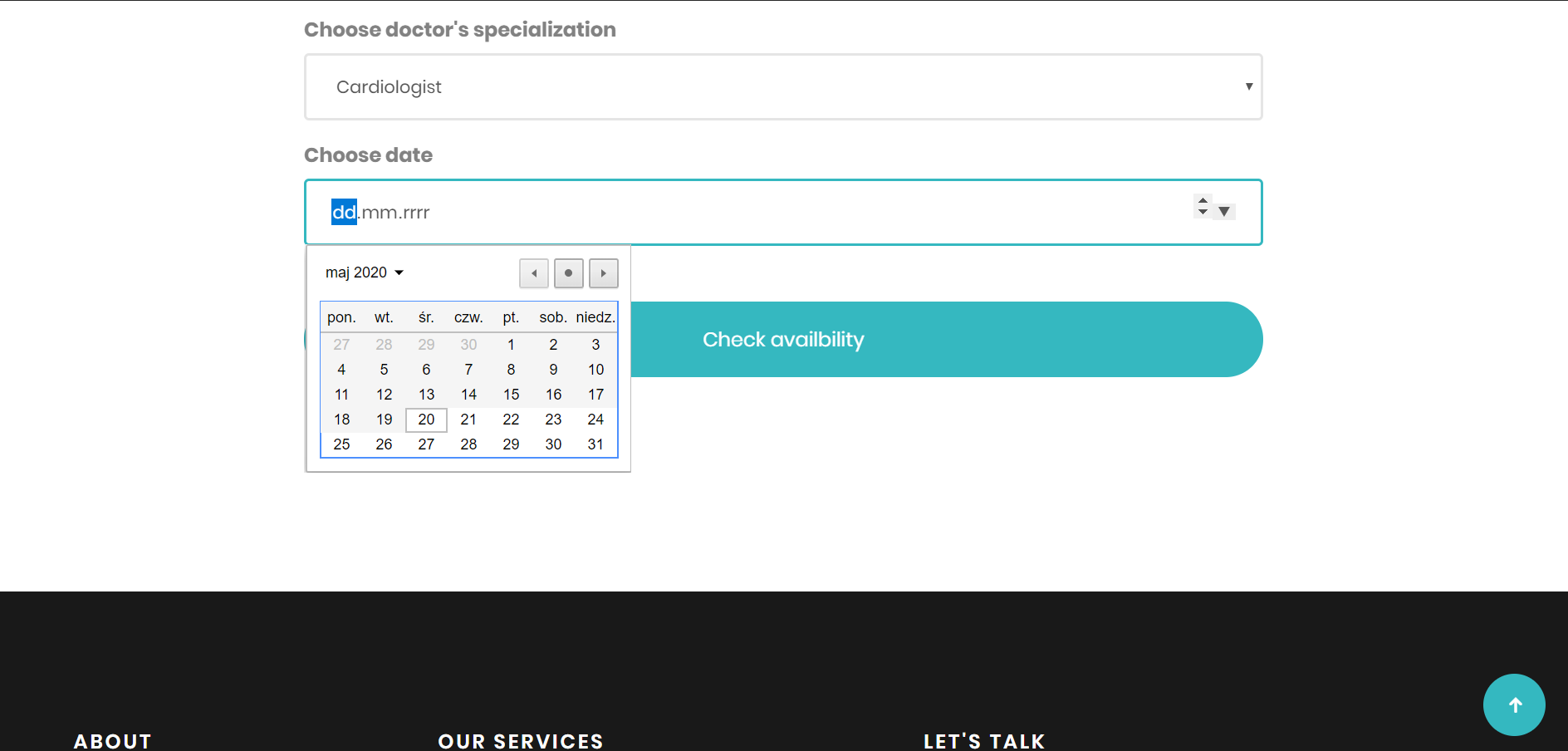
****

****

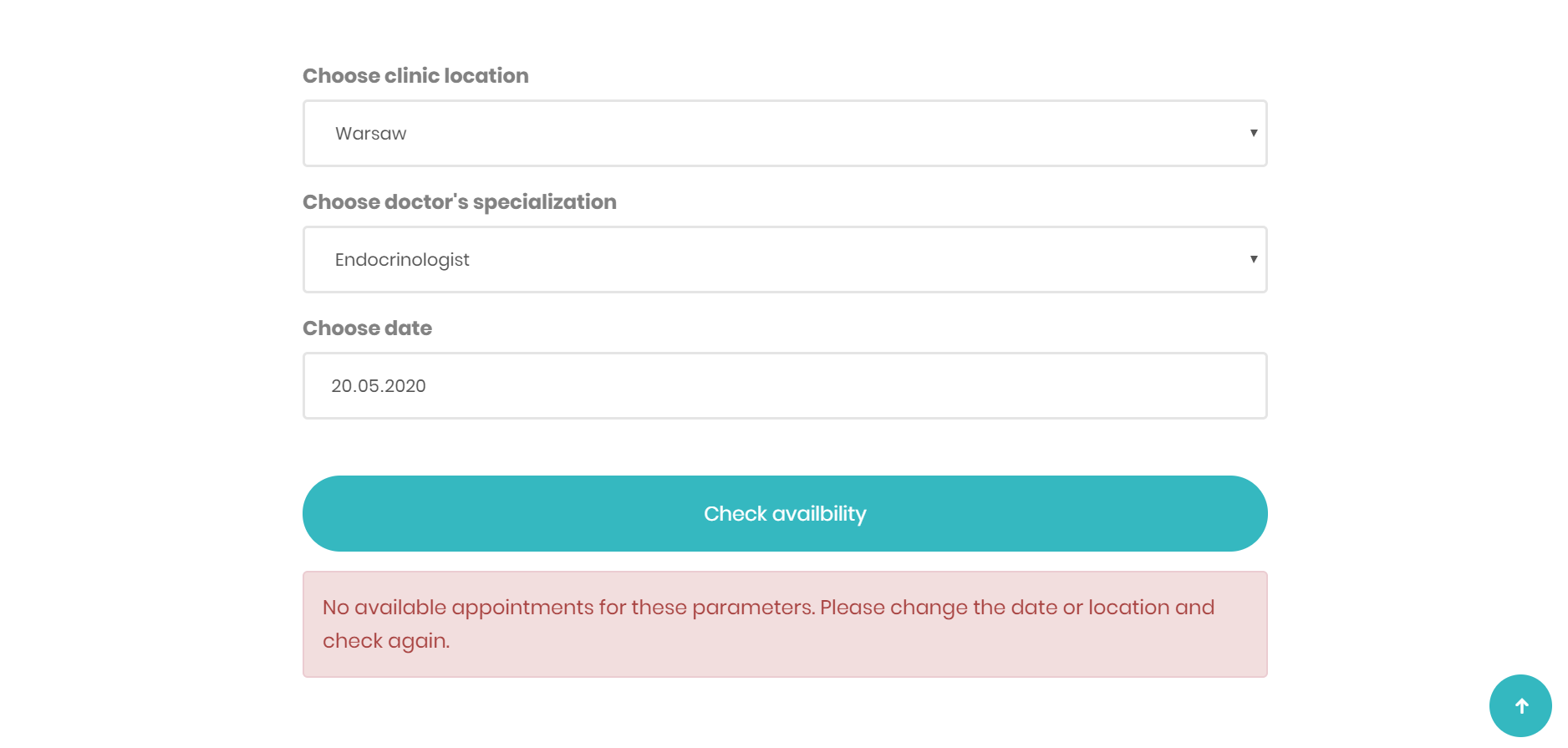
****

**Umów wizytę: book-appointment.html. Przejście poprzez naciśnięcie BOOK AND APPOINTMENT na navbarze. Wybranie miasta, specjalizacji i daty oraz zatwierdzenie przyciskiem Choose avalibility załaduje dostępne godziny lub wyświetli komunikat braku dostępności.**

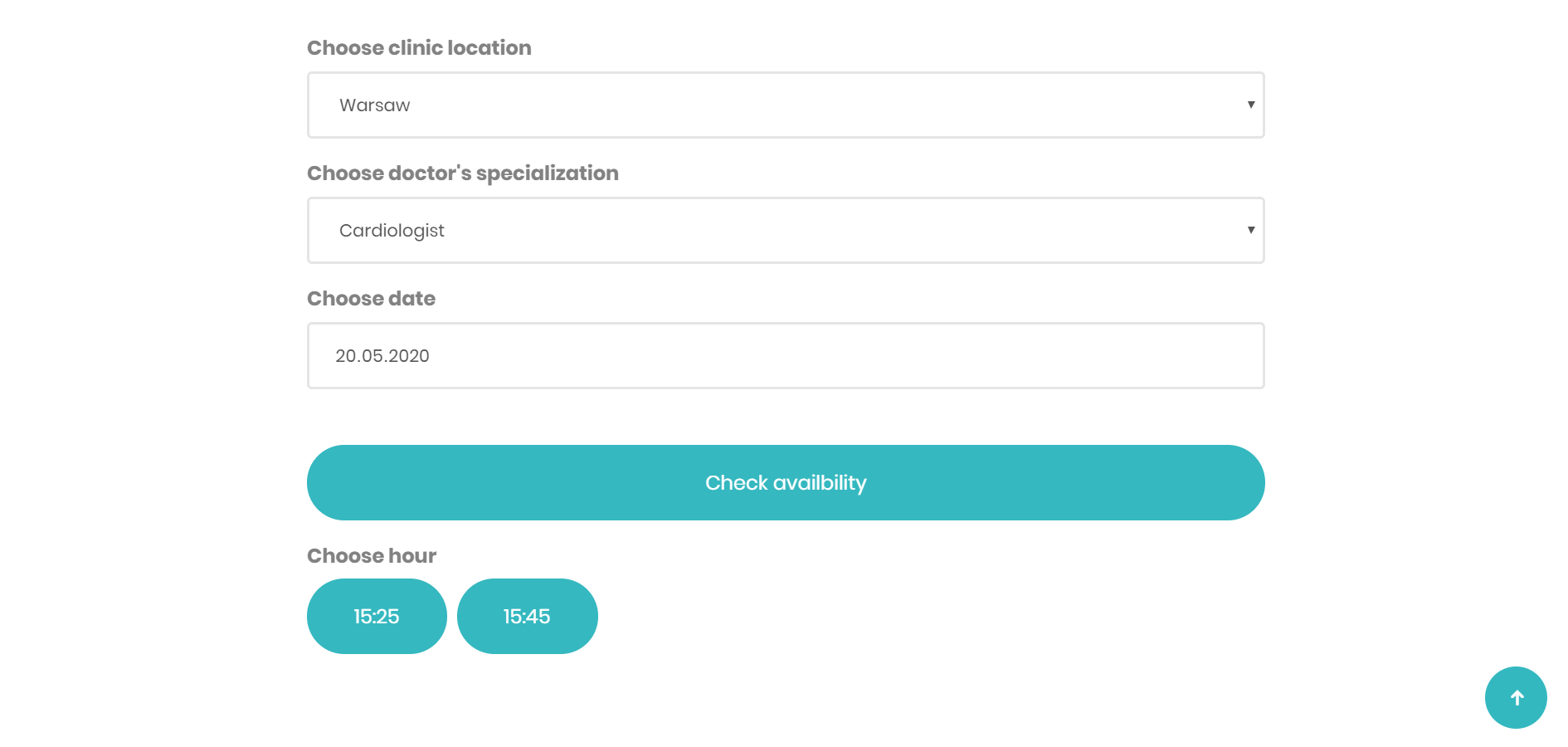
****

****

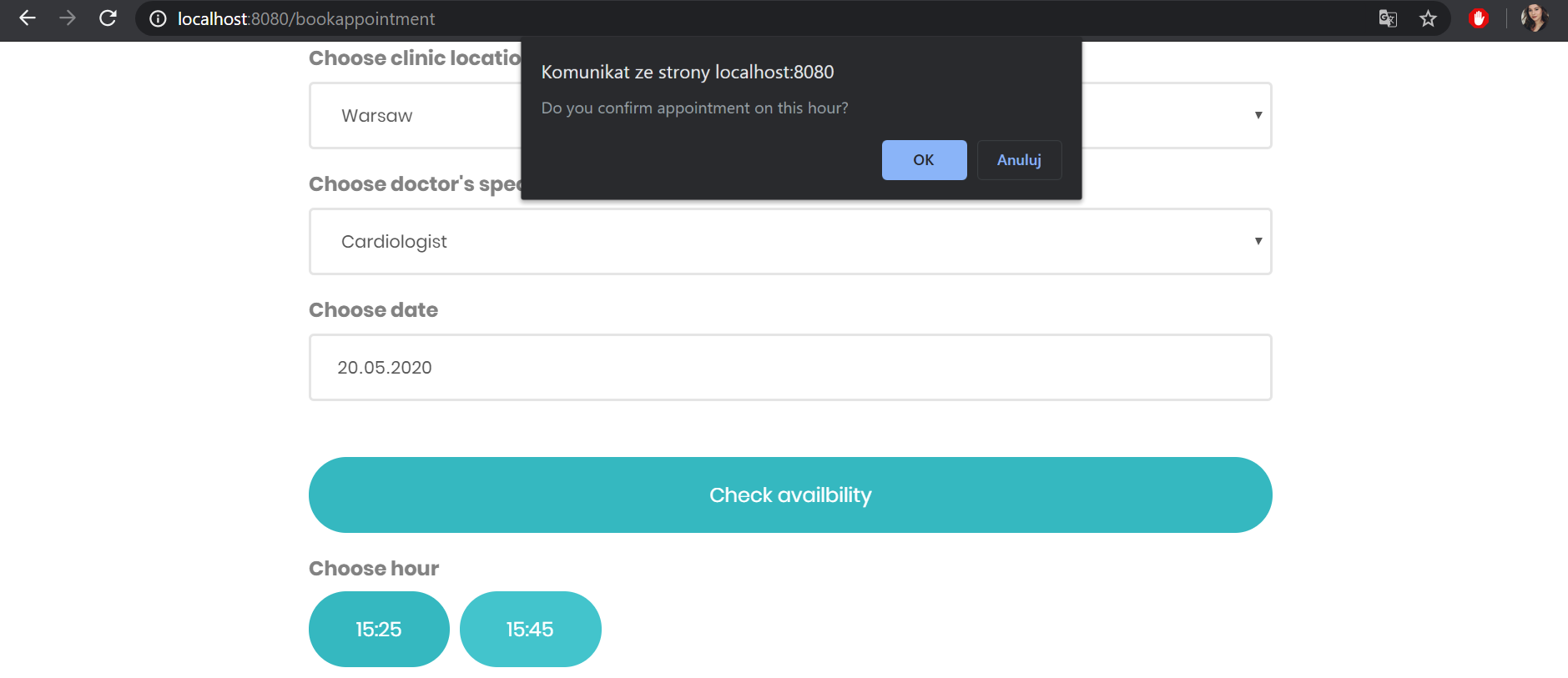
**Przykład braku dostępności.**

****

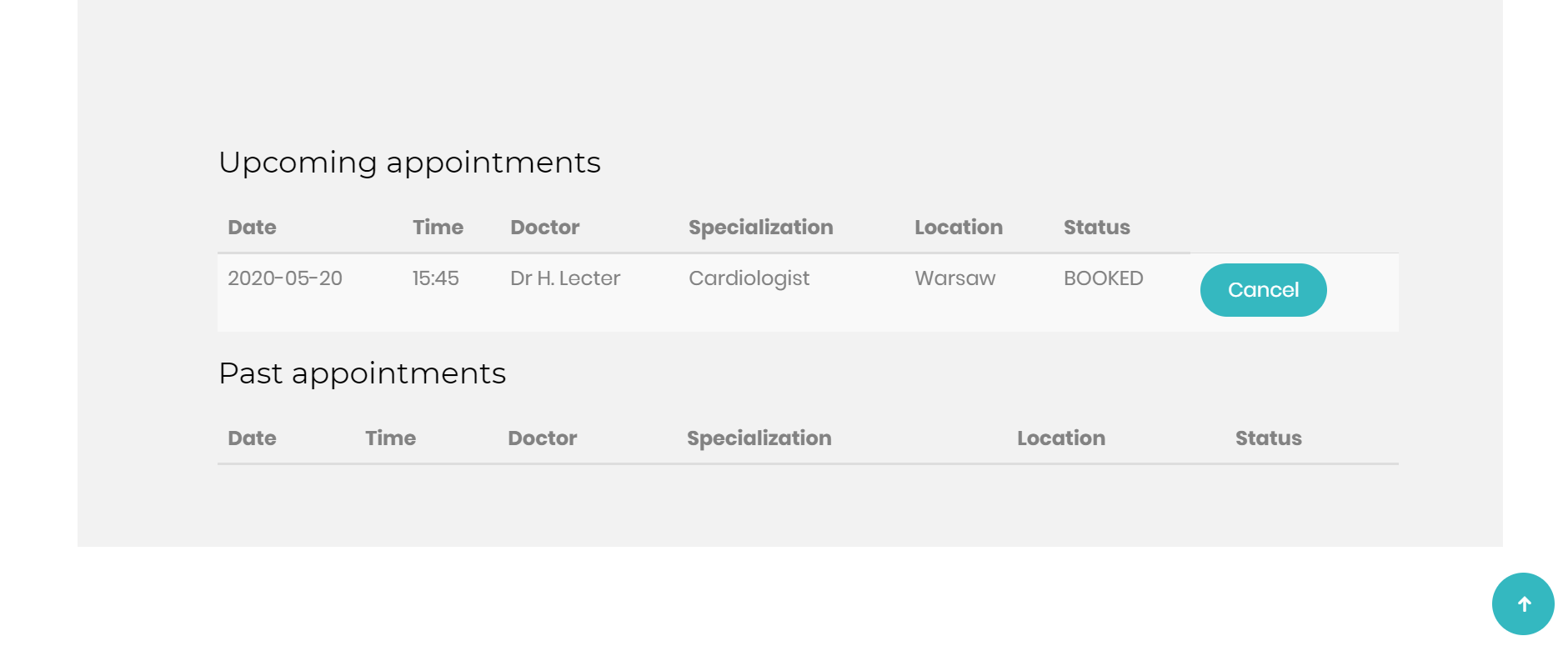
**Przykład dostępnych godzin.**

****

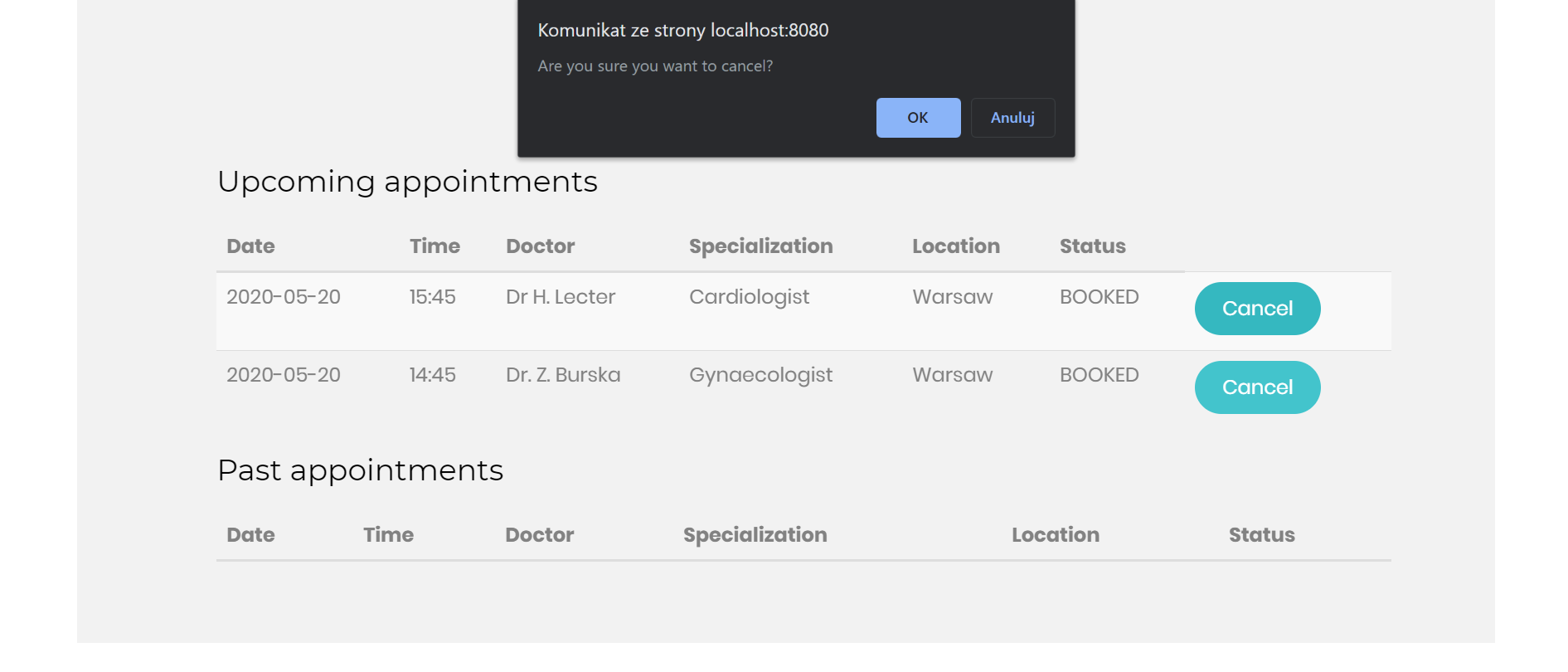
**Po kliknięciu na wybraną godzinę pojawia się komunikat potwierdzenia umówienia wizyty.**

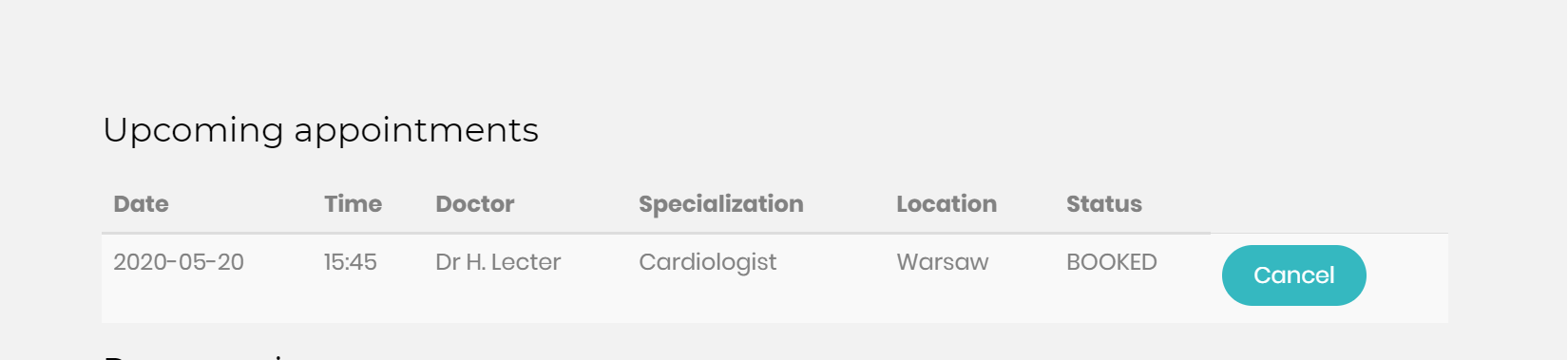
****

**Po zatwierdzeniu przekierowanie do widoku pacjenta.**

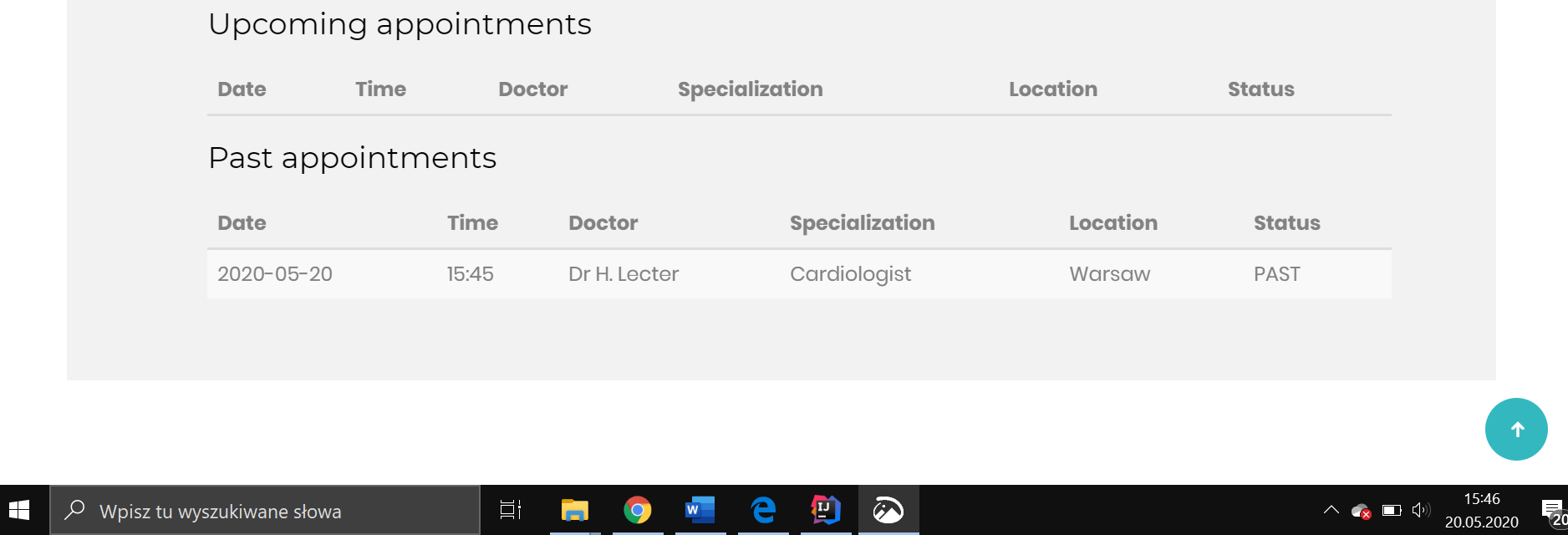
****

**Anulowanie wizyty przyciskiem Cancel. Wizyta znika z konta pacjenta, a zwolniona godzina znów pojawia się jako dostępna. W międzyczasie dodałam kolejną wizytę na 14:45.**

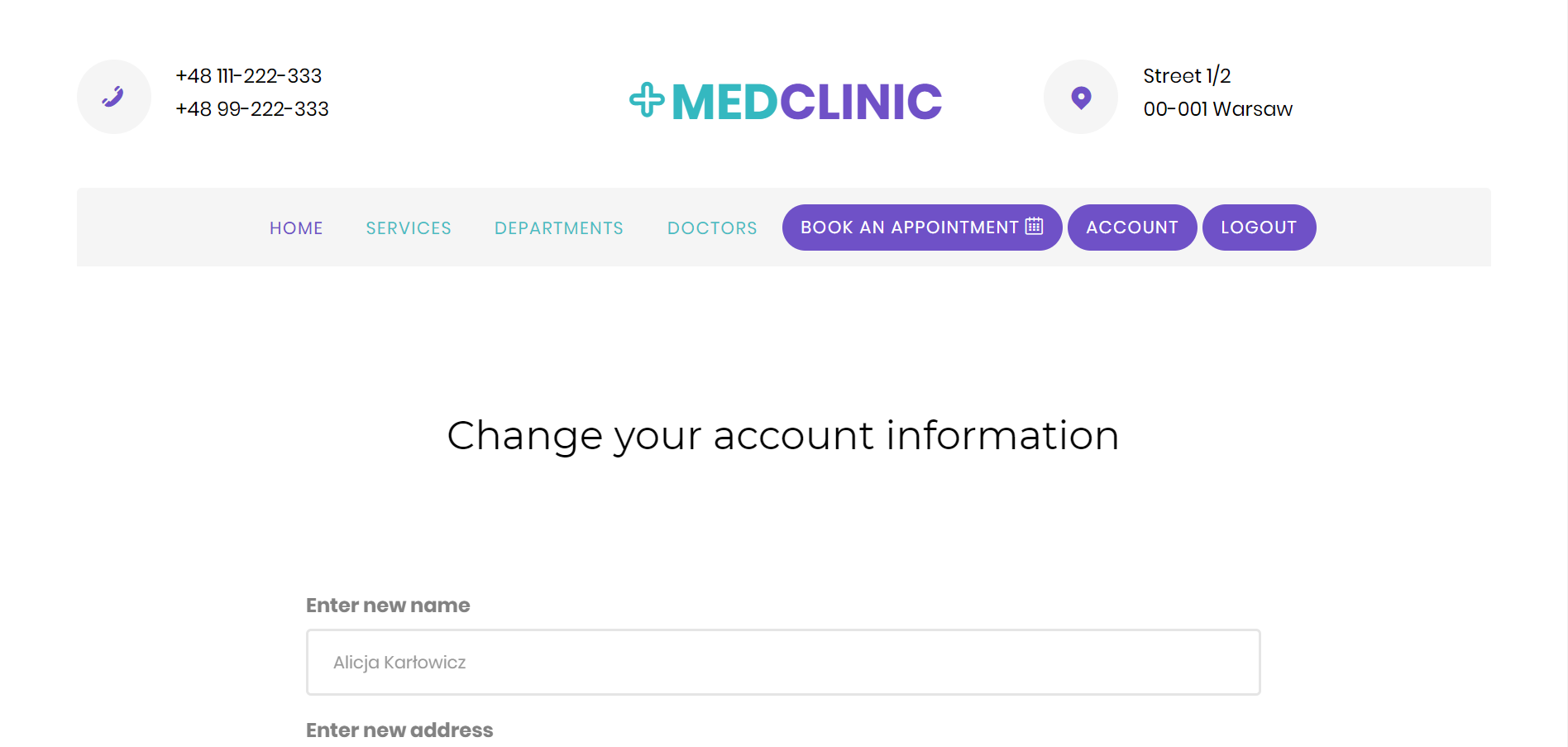
****

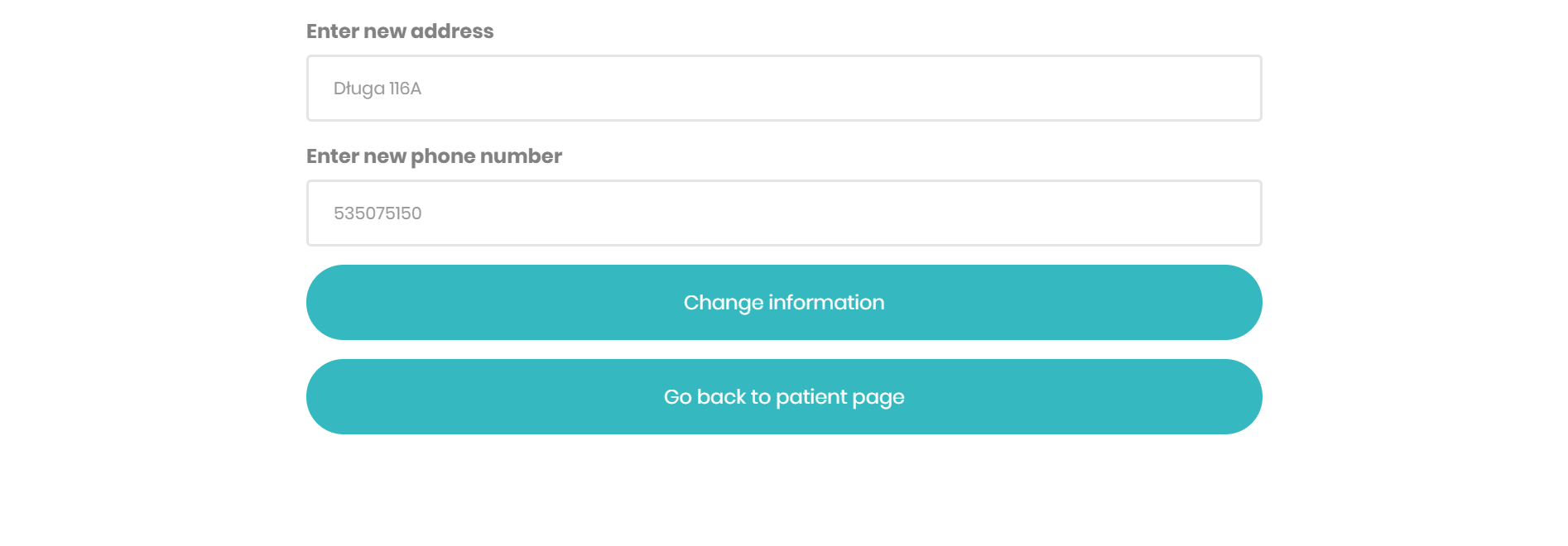
****

**Dawne wizyty: w tabeli past appointments pojawiają się wizyty, które nie były anulowane, a ich godzina już minęła (założenie, że zostały odbyte). Program sprawdza co 5 minut czy wizyty nie przenieść już do Past appointments.**

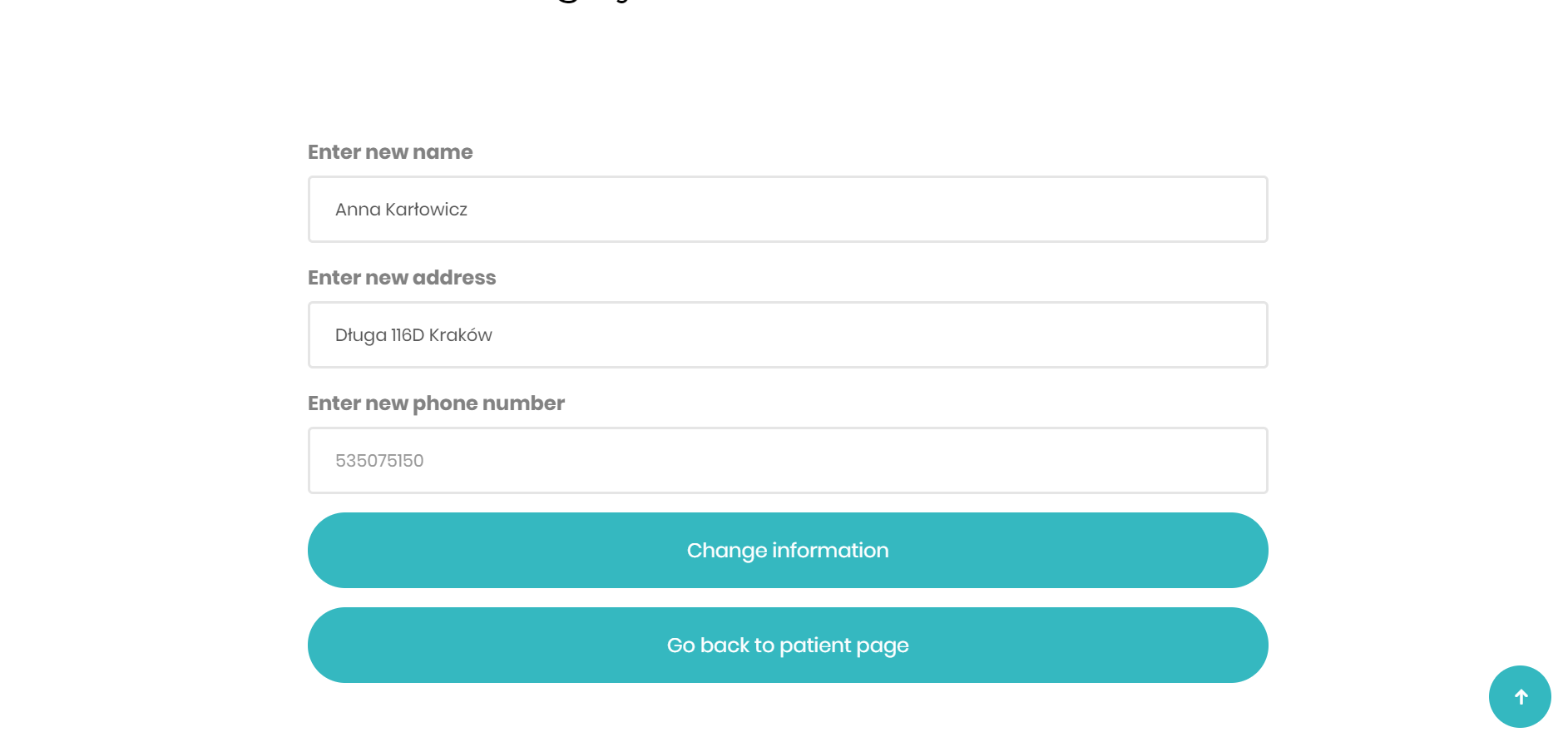
****

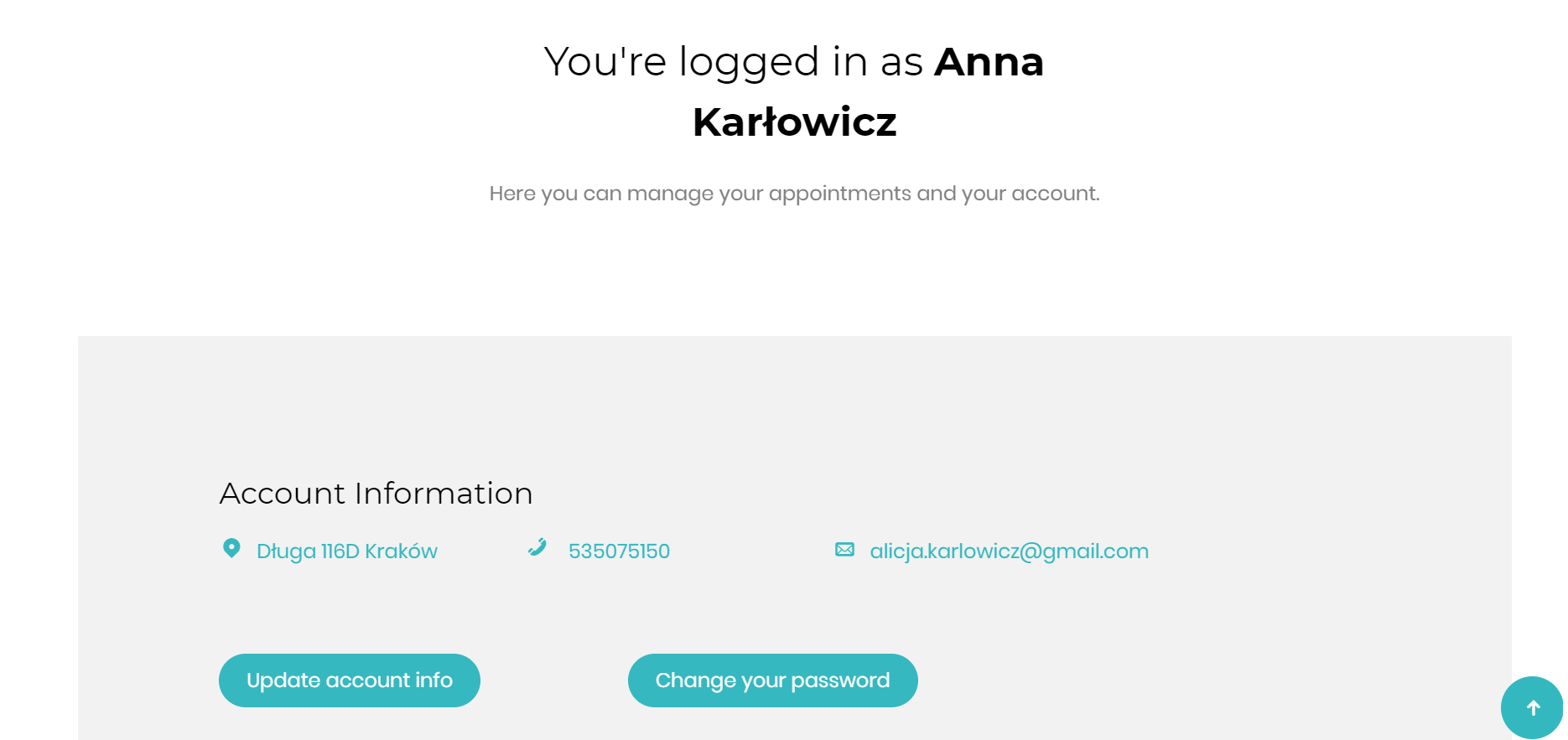
**Zmiana danych pacjenta: change-info.html. Przejście do okna po przyciśnięciu Update account info na panelu pacjenta. Wyświetlane obecne dane jako placeholder.**

****

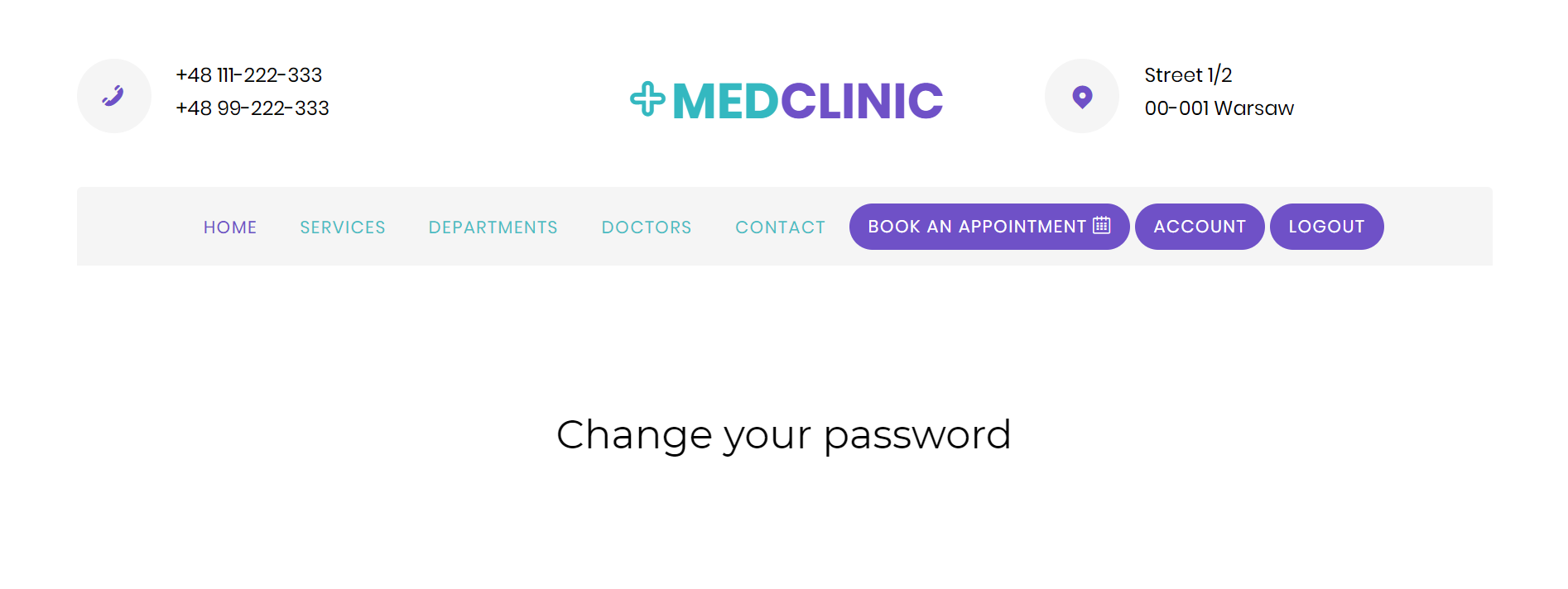
****

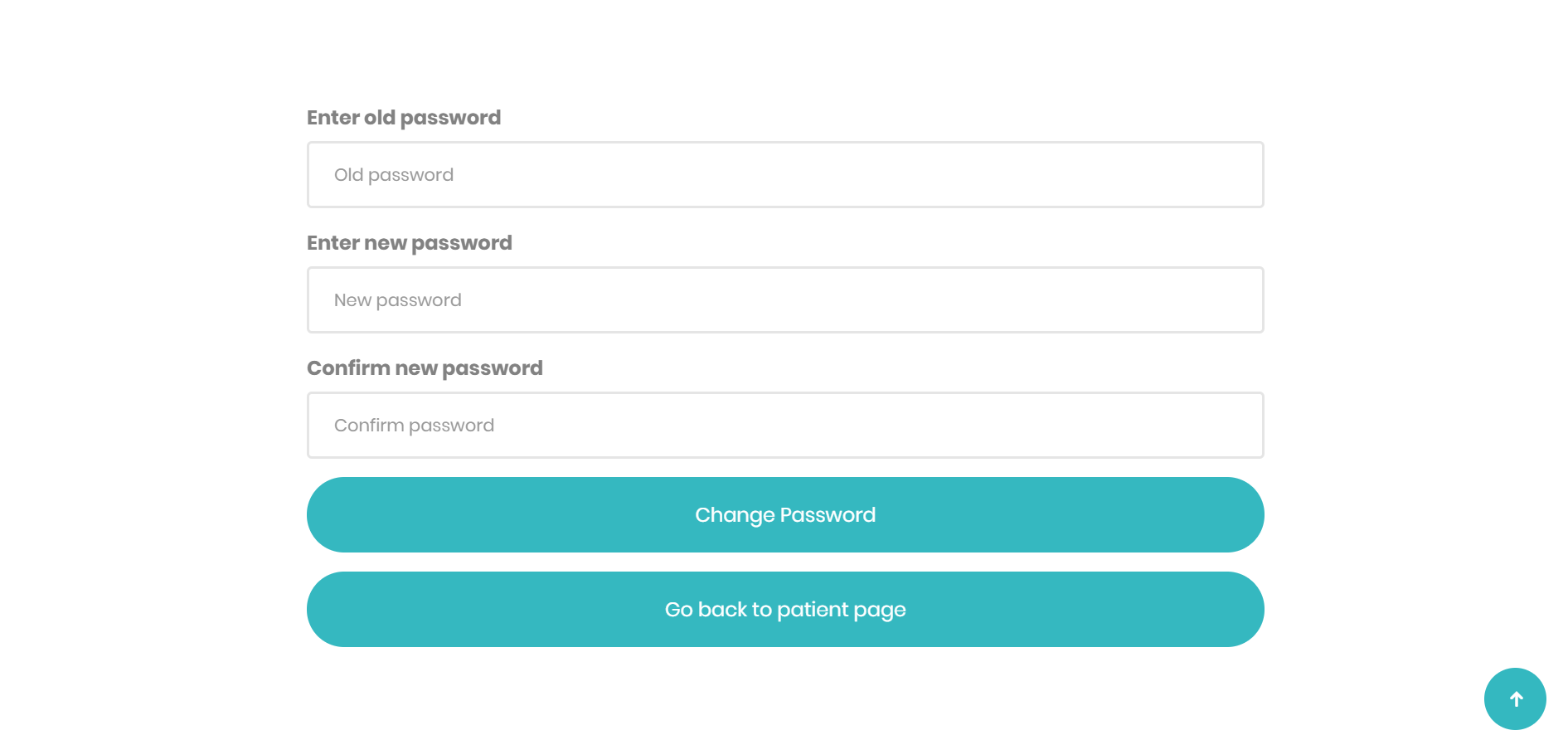
**Zmiana imienia i adresu. Zatwierdzenie przyciskiem Change information, automatycznie przekieruje do panelu pacjenta.**

****

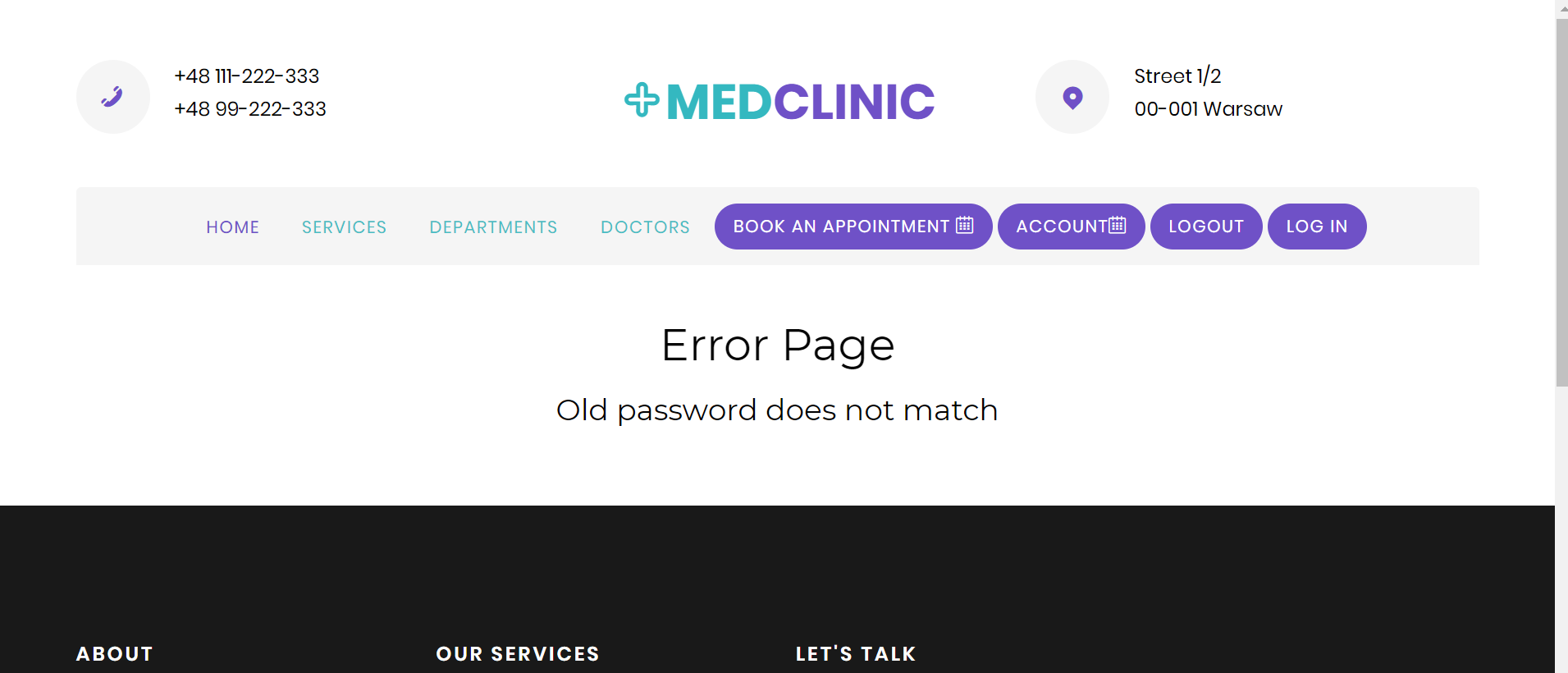
****

**Zmiana hasła: change-password.html. Dostępne po wciśnięciu Change your password w panelu pacjenta.**

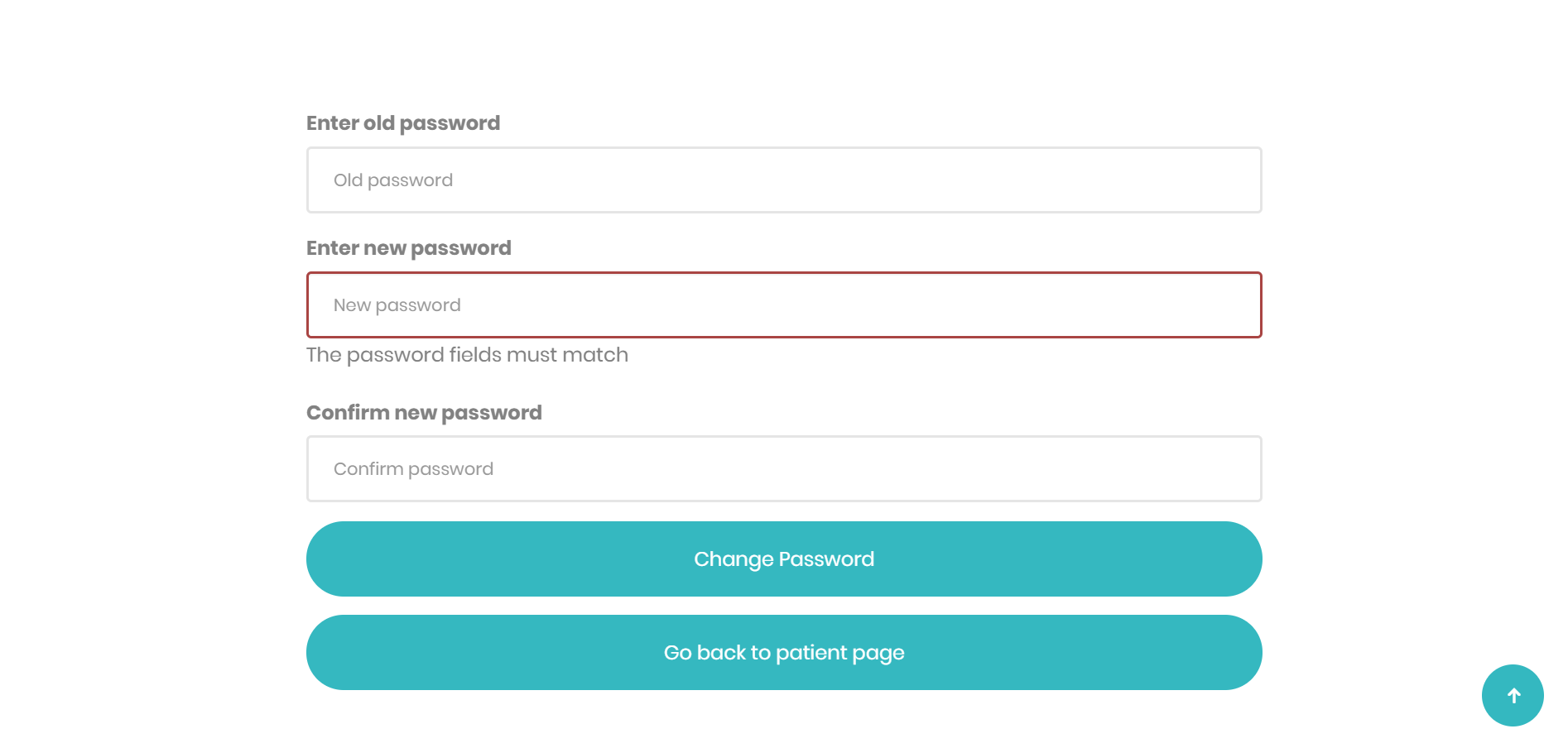
****

****

**Wpisanie złego starego hasła.**

****

**Wpisanie różnych nowych haseł.**

****

**Poprawna zmiana hasła wyloguje nas z sesji i przejdzie do ekranu logowania by zalogować się z nowym hasłem.**

1. **Zwięzły opis kodu**

**Controllers**

W controllerach znajdują się endpointy i logika za nimi, przesyłane są dane z formularz oraz odbierane atrybuty z widoków. Trzy controllery odpowiadając trzem modułom: LoginController to moduł logownia i rejestracji, PatientController to moduł widoku pacjenta, zmiana jego danych, przkazywanie umówionych wizyt oraz ich usuwanie, AppointmentController zajmuje się wyszukiwaniem, umawianiem wizyt oraz sprawdzeniem dostępności.

**Service**

Serwisy odzwierciedlają logikę biznesową aplikacji i operacje na tabelach takie jak znajdowanie elementu w bazie spełniających warunek lub wszystkich, usuwanie, zmienianie atrybutów (update, np. hasła). Wykorzystują domyślne metody dostarczane przez repozytoria. Zmienianie statusu wizyty na przeszły realizowane jest za pomocą adnotacji @Scheduled która wykonuje się co 5 minut od momentu uruchomienia aplikacji.

**Repositories**

Repozytoria to interfejsy, które rozszerzają JpaRepository udostępniane przez Spring JPA (Java Persistance Api) umożliwiają wykonywanie podstawowych operacji na tabelach (findAll(), findById(), save()). Umożliwiają również dodanie własnych metod np. warunku szukania po statusie wizyty.

**Model: Entity**

Klasy z adnotacją @Entity repreztujące tabele w bazie danych. Zawarte w nich również są relacje między tabelami poprzez odpowiednie adnotacje i mapowanie (@OneToOne, @OneToMany/@ManyToOne, @ManyToMany).

**Model: Forms**

Klasy formularzy wraz z odpowiednimi polami.

**Model: Forms: Constrait**

Utworzenie adnotacji @FieldMatch w celu walidacji czy dwa pola formularza mają tę samą wartość. Używane przy potwierdzeniu nowego hasła. Klasa FieldMatchValidator implementuje ConstraintValidator<FieldMatch, Object>.

**Security**

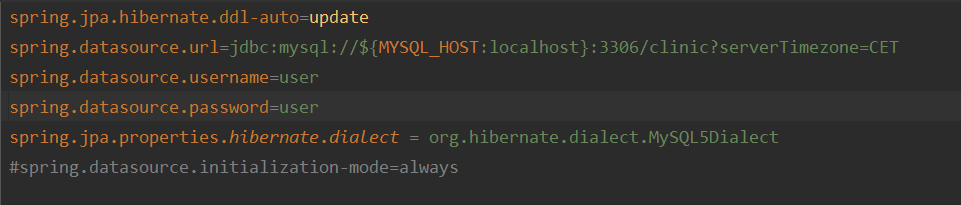
Wykorzystany został Spring Security pomagający w walidacji użytkownika przez logowanie. Hasła trzymane w bazie używają BCrypt implementowanego w SpringSecurity, zatem nie ma do nich bezpośredniego dostępu. Zalogowany użytkownik jest przechowywany w sesji i można go „dostać” poprzez metody SpringSecurity. Konfiguracja została przedstawiona w klasie WebSecurityConfig, który rozszerza klasę WebSecurityCOnfigureAdapter. Wmetodzie configure zostały ustalone zasoby dostępne dla wszystkich (permitAll()), reszta tylko dla zalogowanych (authenticated()), oraz strony logowania, sukcesu logowania, tak samo jak logoutu.

**Thymeleaf templates**

Do wymiany informacji między kontrolerami a widokami posłużyłam się Thymeleafem, template’y znajdują się w folderze resources/templates/.

**Application.properties**

Konfiguracja bazy oraz połączenie z nią. Przy pierwszym uruchomieniu spring.jpa.hibernate.ddl-auto powinno być ustawione na wartość *create*, a initialation-mode na *always*, jednak z tego względu, że create usuwa cały schemat musi być to wykonywane z poziomu roota (użytkownik stworzony user nie posiada opcji DROP na tabelach). Po pierwszym uruchomieniu aplikacji należy zmienić wartość na *update*, która będzie wprowadzała zmiany w schematach tylko jeśli takie zaszły oraz zakomentować initialization-mode.

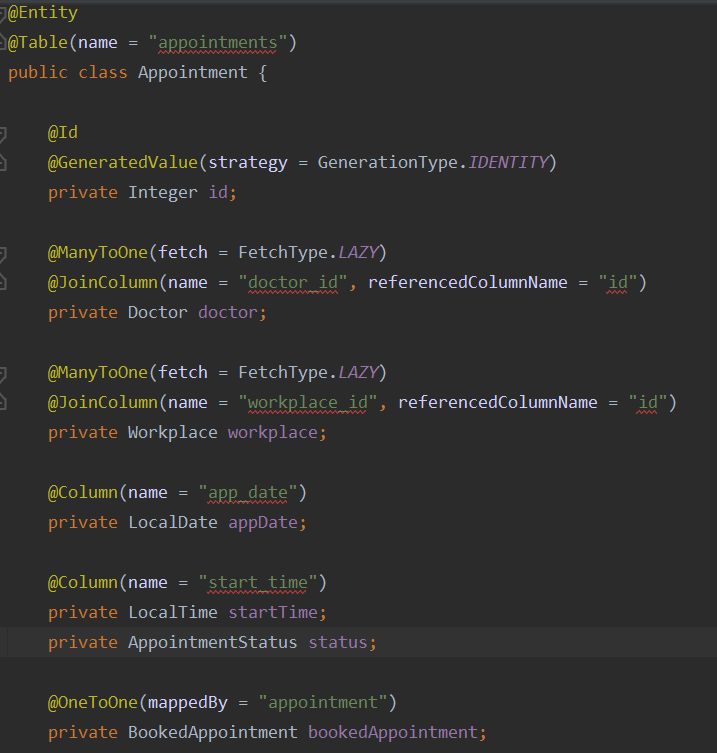


**Data.sql**

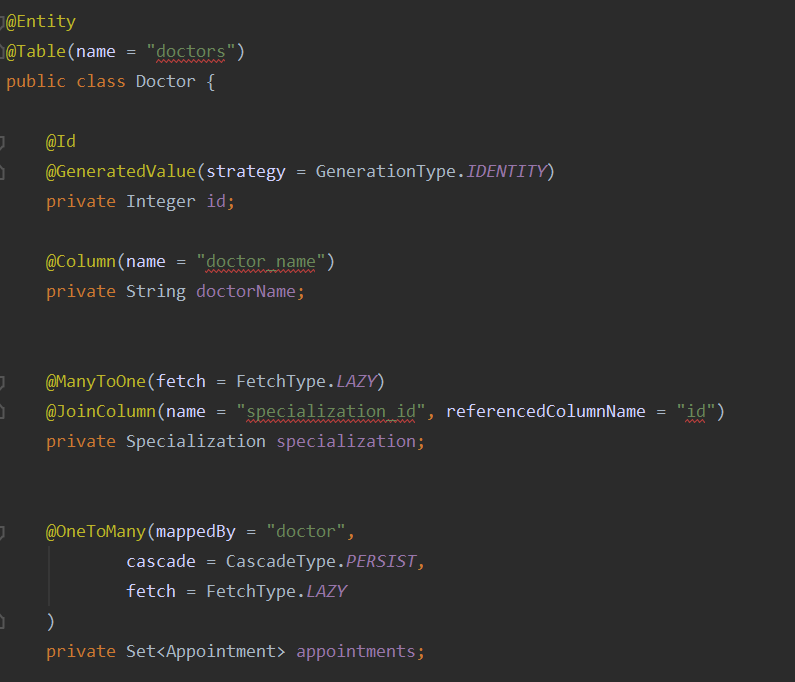
Skrypt SQL do wypełnienia tabel.

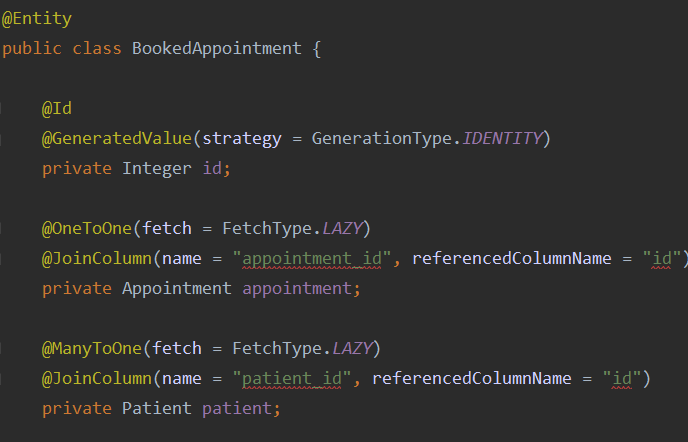
1. **Opis bazy danych**

Tabele zostały utworzone przez framework Hibernate, który stosuje ORM – mapowanie obiektowo – relacyjne. Zależności między tabelami, które powinien uwzględnić Hibernate przy ich tworzeniu zostały zdefiniowane w kodzie w klasach z adnotacją @Entity. Przykład poniżej.

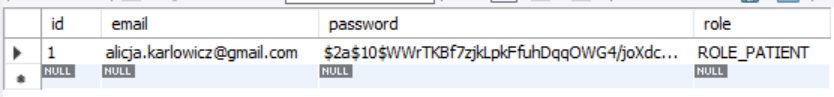


Z @Entity workplace oraz @Entitty doctor (relacja @OneToMany / @ManyToOne)

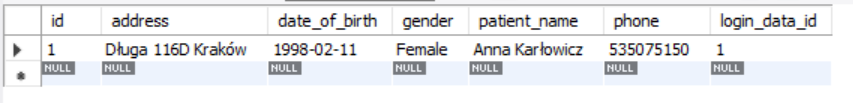


****

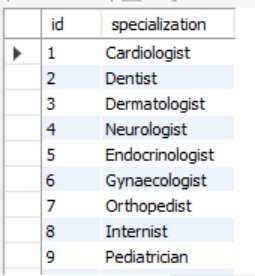
**Tabela login\_data –** przechowuje dane logowania takie jak email, hashowane hasło oraz rola konta. W tym przypadku jest to tylko ROLE\_PATIENT, jednak w przyszłości można dodawać konta lekarzy z rolą doktora, którzy mieliby dostęp do innych endopintów ustalane właśnie na podstawie roli.



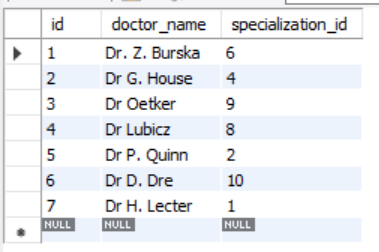
**Tabela patients** – przechowuje dane o pacjentach z formularza rejestracyjnego. Posiada odnośnik w formie klucza obcego do danych logowania (relacja OneToOne).



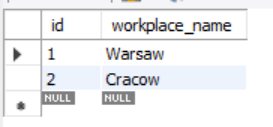
**Tabela specializations** – przechowuje oferowane przez klinikę usługi w postaci specjalizacji. Zawiera id specjalizacji i jej nazwę. Specjalizacja jest następnie przypisywana lekarzowi (relacja OneToMany).



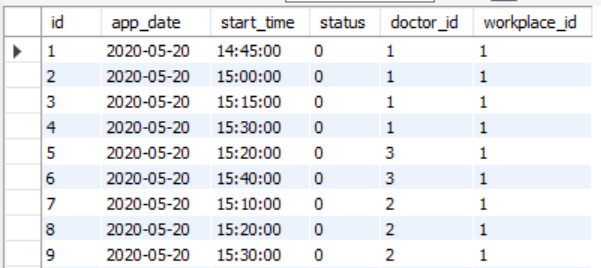
**Tabela doctors** – zawiera spis lekarzy. Lekarz ma przypisaną specjalizacje, posiada id oraz imię i nazwisko. Nie jest natomiast przypisany do jednego miejsca pracy.

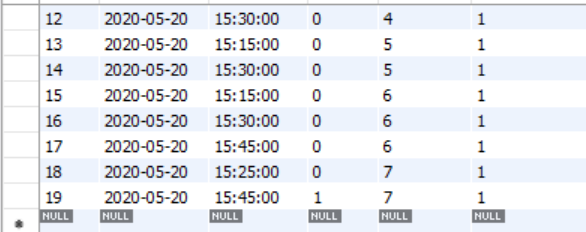


**Tabela workplaces –** zawiera oddziały kliniki. Jej atrybutami są id i nazwa kliniki (ograniczyłam się do nazwy miasta). W relacji z tabelą appointments OneToMany.

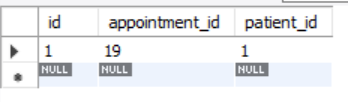


**Tabela appointments** – zawiera wszystkie wizyty. Jej atrybutami są id, data wizyty, godzina startu wizyty, status, lekarz oraz lokalizacja. Status w Springu jest określony jako enum i przyjmuje wartości FREE, BOOKED, CANCELLED, PAST, które w SQL odpowiadają integer 0, 1, 2, 3. Appointment z id 19 jest zabookowany na 15:45 ze statusem 1 czyli BOOKED, natomiast 14:45 została odwołana i posiada status 0 czyli FREE – jest zwolniona i dostępna do zarezerwowana.





**Tabela booked\_appointments** - przechowuje wizyty powiązane z danym pacjentem.



**Użytkownik user:**

