Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Radioelektroniki i Technik Multimedialnych

Praca dyplomowa inżynierska

na kierunku Inżynieria Biomedyczna
w specjalności Informatyka Biomedyczna

Aplikacja mobilna „Dzienniczek seniora”

Zuzanna Adamiuk
Numer albumu 300444

promotor
dr inż. Robert Kurjata

WARSZAWA 2022

Streszczenie

Aplikacja mobilna „Dzienniczek Seniora” to projekt umożliwiający osobom starszym przechowywanie wyników codziennych pomiarów. W niniejszej pracy przedstawiono proces implementacji programu na system operacyjny Android ze szczególnym naciskiem na odpowiedni, dla wybranej grupy docelowej, interfejs graficzny. Oprócz archiwizacji wyników, użytkownicy mogą obejrzeć ich wizualizacje oraz udostępnić je wcześniej wybranemu opiekunowi. Zaimplementowano również możliwość dodawania przypomnień o badaniach, które wysyłane są do użytkownika w formie powiadomień push. W pracy opisano wykorzystane technologie oraz narzędzia – zwrócono szczególną uwagę na sposób w jaki komunikują się ze sobą warstwy kliencka i serwerowa aplikacji.

Słowa kluczowe:

aplikacja mobilna, codzienne pomiary, monitorowanie zdrowia, osoby starsze, urządzenia mobilne, Android, Python, Django, React Native

Abstract

Mobile application “Dzienniczek Seniora” is a project, which aim is to enable the elderly to store their daily measurements. The thesis describes the process of implementing the software for Android operating system with special emphasis on appropriate graphic interface for project’s target group. Besides archiving, users are able to display data visualizations and share them with previously chosen supervisor. There is also a possibility to add reminders about measurements, that would be sent to user as push notifications. Technologies and software used during project realisation are described in the thesis, emphasising the way client and server sides communicate with each other.

Keywords:

mobile application, daily measurements, health monitoring, senior, mobile devices, Android, Python, Django, React Native

Spis treści

1 Wstęp

W stale starzejącym się społeczeństwie niezwykle ważne jest, aby dbać o potrzeby osób w podeszłym wieku, które stanowią co raz większy procent ludności. Szacuje się, że w 2020, na całym świecie, żyło ponad 727 milionów osób powyżej 65 roku życia – było to niemal 10% ówczesnej populacji. Ponadto, według Organizacji Narodów Zjednoczonych, liczba ta prawdopodobnie zwiększy się dwukrotnie w ciągu najbliższych 30 lat [1]. Wraz z wiekiem ciało człowieka się zmienia - staje się coraz słabsze, pogarsza się wzrok i słuch a osoby starsze są bardziej podatne na choroby.

1.1 Choroby cywilizacyjne

Oprócz naturalnych zmian zachodzących w organizmie, ludzie coraz częściej cierpią na choroby cywilizacyjne, które często ujawniają się dopiero w podeszłym wieku. Dzięki ogromnemu postępowi w wielu dziedzinach nauki i techniki, warunki życia istotnie zmieniły się względem zeszłego stulecia, czego skutkiem są nieznane dotąd zagrożenia, takie jak występowanie właśnie chorób cywilizacyjnych. Tego typu problemy pojawiają się głównie w obszarach wysoko zurbanizowanych, gdzie ludzie żyją w pośpiechu i ciągłym stresie. Ponadto, większość mieszkańców regionów rozwiniętych cechuje brak aktywności fizycznej, siedzący tryb życia oraz skłonność do niezdrowego odżywiania. Należy też zwrócić uwagę na zanieczyszczenie środowiska, które pojawia się w zurbanizowanych obszarach. Wszystkie wymienione czynniki mogą wpłynąć na prawdopodobieństwo wystąpienia chorób cywilizacyjnych takich jak choroby serca i układu krążenia (nadciśnienie tętnicze, miażdżyca), cukrzyca (typu 2), nowotwory czy otyłość. [2]

1.1.1 Choroby układu krążenia

Nadciśnienie tętnicze jest najczęściej występującą w Polsce chorobą i jednocześnie, według Światowej Organizacji Zdrowia, najczęstszą przyczyną przedwczesnych zgonów na świecie. Rozpoznanie tego schorzenia polega na pomiarze ciśnienia tętniczego krwi – występuje, jeśli średnia zmierzonych wartości, wyliczona z co najmniej dwóch różnych wizyt, jest równa lub wyższa niż 140 mm Hg skurczowego ciśnienia tętniczego, zaś wartość rozkurczowego ciśnienia tętniczego jest równa lub wyższa niż 90 mm Hg. Głównym celem w leczeniu nadciśnienia tętniczego jest przywrócenie wymienionych wcześniej parametrów do wartości mieszczących się w normie dla osób chorych (**Błąd! Nie można odnaleźć źródła odwołania.**), co najczęściej udaje się osiągnąć z wykorzystaniem farmakoterapii oraz zmianą stylu życiu na bardziej aktywny. Jednak niezbędne jest też monitorowanie wartości ciśnienia tętniczego, aby kontrolować to, czy wdrożona terapia przynosi efekty. Ponadto, codzienne wykonywanie pomiarów zmniejsza ryzyko wystąpienia reakcji białego fartucha, która charakteryzuje się zwiększonym ciśnieniem w gabinecie lekarskim spowodowanym stresem. [3] Często zdarza się, że nadciśnienie tętnicze jest przyczyną pojawienia się innego schorzenia atakującego układ krwionośny – miażdżycy. Jest to przewlekła choroba tętnic cechująca się złożonymi zmianami w błonie wewnętrznej zaatakowanych naczyń. Wspomniane zmiany prowadzą do zwężenia lub zamknięcia światła naczyń, co może prowadzić do niedokrwienia lub martwicy narządu, do którego krew była dostarczana przez zablokowaną tętnicę. [4]

Tabela 1 Docelowe i niezalecane wartości ciśnienia tętniczego w zależności od wieku pacjenta z kryteriami rozpoczęcia terapii. (SCT – skurczowe ciśnienie tętnicze, RCT – rozkurczowe ciśnienie tętnicze) [3]

	Nadciśnienie tętnicze skurczowo-rozkurczowe Pacjent w wieku < 65 lat	Nadciśnienie tętnicze skurczowo-rozkurczowe Pacjent w wieku 65–80 lat	Nadciśnienie tętnicze skurczowo-rozkurczowe Pacjent w wieku > 80 lat	Izolowane nadciśnienie skurczowe
Kryterium CT rozpoczęcia terapii	≥ 140/90	≥ 140/90	≥ 160/90	≥ 140
I° (pośredni) cel terapeutyczny SCT	< 140	–	–	< 140*
II° (ostateczny) cel terapeutyczny SCT	< 130	< 140	< 150	< 130*
SCT niezalecane	< 120	< 130	< 130	< 120*
Cel terapeutyczny RCT	< 80	< 80	< 80	–
RCT niezalecane	< 70	< 70	< 70	< 65

1.1.2 Cukrzyca i otyłość

Inną chorobą cywilizacyjną, która dotyka co raz więcej osób jest cukrzyca. Światowa Organizacja Zdrowia definiuje ją jako „grupę chorób metabolicznych charakteryzującą się hiperglikemią wynikającą z upośledzenia wydzielania i/lub działania insuliny". [5] Według danych organizacji cukrzyca powoduje 2% wszystkich zgonów w Polsce, przy rozpowszechnieniu równym 9,5%. [6] Obecnie szacuje się, że 20% osób chorych na cukrzycę ma więcej niż 65 lat (136 milionów osób). [7] Wymienia się dwa typy tego schorzenia: typ 1 - spowodowany brakiem produkcji insuliny w trzustce oraz typ 2 - występujący na skutek zaburzeń działania insuliny w organizmie, co prowadzi do podwyższenia stężenia tego hormonu. Cukrzyca typu 2 jest ściśle powiązana z otyłością, dlatego też w profilaktyce niezwykle ważne jest monitorowanie wskaźnika masy ciała (BMI – *Body Mass Index*), w zależności od którego ustala się pewnego rodzaju ograniczenia dietetyczne.

$$\mathrm{BMI} = \frac{\text{masa ciała [kg]}}{\text{wzrost [m]} \times \text{wzrost [m]}}$$

Równanie 1. Wzór na obliczenie wartości wskaźnika BMI

Chorobę tę rozpoznaje się najczęściej na podstawie badania glikemii na czczo, podczas którego określa się stężenie cukru we krwi – wartości powyżej 99 mg/dl wskazuje na nieprawidłową glikemię na czczo (IFG), zaś wartości powyżej 125 mg/dl (zmierzone w kilku dniach) wskazują na cukrzycę. W leczeniu niezwykle ważna jest dieta, aktywność fizyczna i przyjmowanie leków, w celu kontrolowania efektów terapii zaleca się kontrolowanie poziomu cukru we krwi każdego dnia. [5]

1.1.3 Archiwizacja pomiarów

Codzienne pomiary są niezbędne do monitorowania stanu zdrowia osób starszych. Chociaż do dokonania rejestracji wyników wykorzystują nowoczesne glukometry czy ciśnieniomierze, to do ich przechowywania wykorzystują tradycyjne, papierowe notesy,

które nie są trwałym i bezpiecznym nośnikiem informacji. W czasach szybkiego rozwoju technologicznego, najstarsi członkowie społeczeństwa mogą czuć się zagubieni, jednak warto zadbać, aby oni też mogli korzystać z ułatwień jakie ów rozwój ze sobą niesie.

2 Cel i wymagania pracy

Celem pracy jest zaprojektowanie i wykonanie aplikacji mobilnej dla systemu Android, będącej narzędziem do gromadzenia danych pomiarowych - ciśnienia krwi, glikemii oraz wagi - ze specjalnym naciskiem na wsparcie osób starszych. W przypadku niniejszej pracy wymagało to odpowiedniego zaprojektowania interfejsu użytkownika z uwzględnieniem wieku korzystających osób, ich słabego wzroku oraz trudności w poruszaniu się. Aplikacja ma umożliwiać archiwizację danych oraz dawać możliwość przekazania dostępu do danych wybranym przez seniora użytkownikom poprzez panel opiekuna. Aplikacja umożliwia graficzną prezentację wyników w postaci wykresów oraz pozwala na ustawienie przypomnień w formie powiadomień *push*.

2.1 Doświadczenia użytkownika

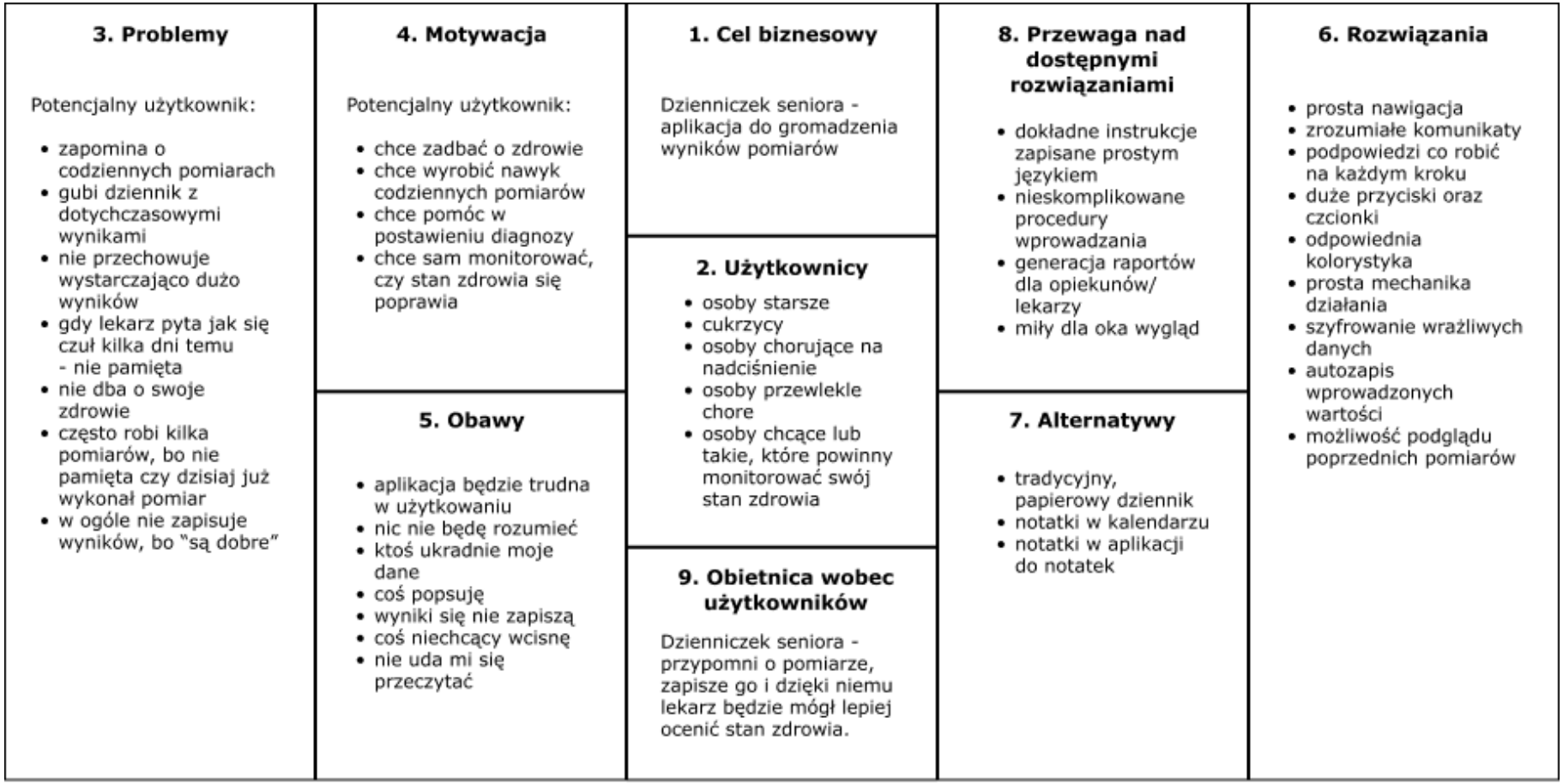
Wielu seniorów czuje się niepewnie korzystając z nowych technologii, woli używać tradycyjnych nośników informacji. Bardzo często wynika to z obaw, że niechcący doprowadzą do awarii używanego systemu. Obecnie jednak, co raz więcej starszych użytkowników decyduje się na korzystanie ze smartfonów, a co za tym idzie - aplikacji mobilnych.

Narzędziami wykorzystywanymi, aby lepiej zrozumieć potrzeby grupy docelowej w procesie projektowym są tak zwane kanwy - są to plansze z hasłami i pytaniami pomocniczymi, które wypełniają projektanci doświadczeń. Podczas realizacji niniejszej pracy postanowiono wykorzystać *User Centered Design Canvas* [8] (Rysunek 1). Większość kanw opiera się na biznesowych aspektach projektów, jednak wybrana plansza, skupia się na potrzebach użytkowników. Uzupełnienie planszy wskazuje kierunek w jakim należy podążać – w niniejszej pracy, dzięki niej wywnioskowano, że między innymi nie powinno się tworzyć skomplikowanych ścieżek użytkownika. Dla osób starszych najważniejsze jest to, aby oglądane treści były zrozumiałe. Seniorom powinno się zapewnić poczucie bezpieczeństwa, aby nie obawiali się konsekwencji za popełnione pomyłki podczas użytkowania aplikacji – bardzo dobrym na to sposobem jest używanie ikony „strzałki" umożliwiającej cofnięcie do poprzedniego ekranu czy operacji. Kolejną dobrą praktyką jest minimalizacja używania klawiatury - o wiele bardziej efektywne jest wykorzystywanie paneli z wyborem dostępnych opcji niż ręczne wprowadzanie całych haseł – klawiatury w smartfonach mają bardzo małe przyciski, które sprawiają problem osobom starszym. Badania wykazały, że seniorzy bardzo często, niechcący, przyciskają elementy znajdujące się przy krawędziach ekranu. Wynika to ze sposobu trzymania przez nich smartfonu, dlatego zaleca się, aby w tych obszarach nie umieszczać przycisków wywołujących różnego rodzaju akcje. [9]

2.2 Interfejs graficzny

Projektując interfejs graficzny należało, przede wszystkim, skupić się na dobrej widoczności elementów, zważając na bardzo często występujące wady wzroku seniorów. Warto zwrócić uwagę na to, że większość osób starszych czuje się pewniej korzystając z klasycznych guzików niż przycisków na ekranie dotykowym. Nie obawiają się wtedy, że wcisną nie ten element, który chcieli, dlatego przyciski powinny mieć odpowiednią wielkość oraz powinna być zachowana odległości między nimi. Ponadto, na guzikach

powinno się umieszczać tekst z informacją, jaką akcję wykonuje dany element - okazuje się, że nie zawsze ikonografia jest wystarczająco jednoznaczna dla osób starszych. [10] W kontekście wyglądu aplikacji, należy zadbać również o odpowiednią kolorystykę. Wraz z wiekiem, ludzie co raz gorzej odróżniają elementy w tym samym kolorze, ale o innym odcieniu, dlatego zawartość ekranu powinna być w kolorze kontrastującym do tła. [11] Ponadto, poprzez ekspozycję na światło ultrafioletowe, rogówki oraz soczewki nabierają żółtego odcienia. Ma to bezpośredni wpływ na percepcję kolorów – utrudnia rozróżnianie barw, zwłaszcza niebieskiego, zielonego i fioletowego, [12] dlatego powinno się unikać łączenia ze sobą właśnie tych kolorów. Wybierając szatę graficzną dla projektowanej aplikacji należy pamiętać o tym, jak częstą przypadłością są zaburzenia percepcji barw i projektować tak, aby program był dostępny dla wszystkich.

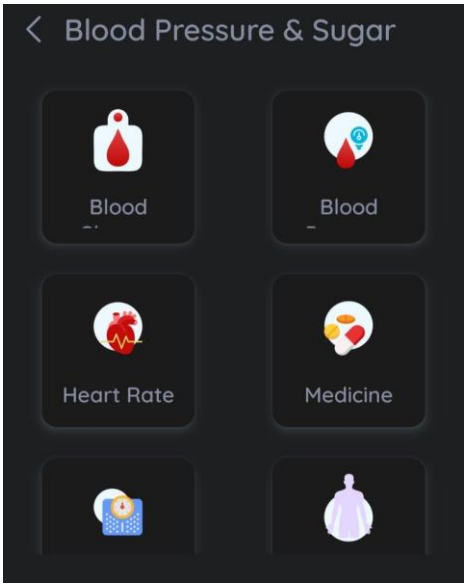


Rysunek 1 Wypełniona User Centered Design Canva dla aplikacji Dzienniczek Seniora

3 Rozwiązania istniejące na rynku

Na platformie Sklep Play, z której użytkownicy systemu Android mogą pobierać aplikacje na swoje urządzenia jest bardzo szeroki wybór aplikacji umożliwiających przechowywanie wyników badań. Jednak duża część z nich nie jest przystosowana do użytkowania przez osoby starsze – nie są zachowane odpowiednie kontrasty między elementami czy stosowanie nieodpowiedniej wielkości tekstu.

Wszystkie aplikacje testowano przy użyciu smartfonu Xiaomi Mi 9, z systemem Android w wersji 10. Podczas analizowania dostępnych na rynku aplikacji, sprawdzano ich działanie z ustawioną największą możliwą dla systemu czcionką, co jak się okazało negatywnie wpływało na działanie oraz czytelność części z testowanych aplikacji. (Rysunek 2, Rysunek 5) Warto również zaznaczyć, że testom poddano jedynie rozwiązania teoretycznie dostępne w języku polskim – jednak w części z nich pojawiały się błędy językowe, które negatywnie wpłynęły na ogólny odbiór aplikacji.



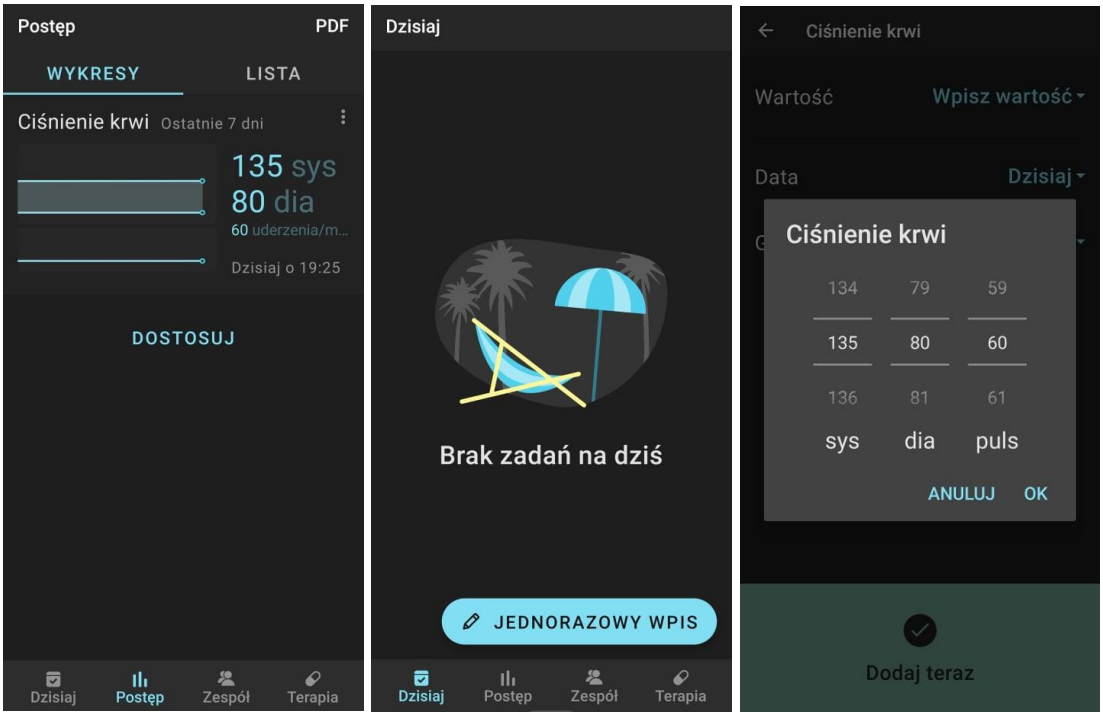
Rysunek 2 Zrzut ekranu wyboru pomiaru w aplikacji Blood Pressure and Sugar Tracker (Little Angel Apps). [13] Zrzut ekranu został wykonany z największą możliwą czcionką systemową.

Wiele dostępnych aplikacji ma takie funkcje jak dodawanie przypomnienia o lekach czy o wizytach u specjalistów. Ponadto, dostępne w Sklepie Play programy umożliwiają generację oraz wysyłkę raportów z wynikami pomiarów. Aplikacja opisywana w niniejszej pracy nie posiada wymienionych funkcji, jednak jak najbardziej byłoby wskazane wprowadzenie ich w przyszłości.

3.1 MyTherapy – Przypomnienia o lekach

Aplikacja MyTherapy została wydana przez niemiecką firmę smartpatient, w Sklepie Play jest dostępna od kwietnia 2015 roku i cały czas jest aktualizowana. Podczas już niemal 7 lat została ona pobrana ponad 5 000 000. [14]

Program pozwala na archiwizację pomiarów ciśnienia krwi z pulsem, tętna spoczynkowego, wagi oraz poziomu cukru we krwi przed i po posiłku. Ponadto, umożliwia ustawienie przypomnień o pomiarach, zażyciu leków, wizytach lekarskich, ale także o aktywności fizycznej (trening fitness, spacer, fizjoterapia, bieganie, jazda na rowerze). MyTherapy umożliwia udostępnienie swoich danych (o zażywanych lekach i wizytach lekarskich) z wykorzystaniem specjalnie generowanego kodu, który należy przekazać wybranej osobie, aby z użyciem tego kodu uzyskała dostęp w swojej aplikacji.



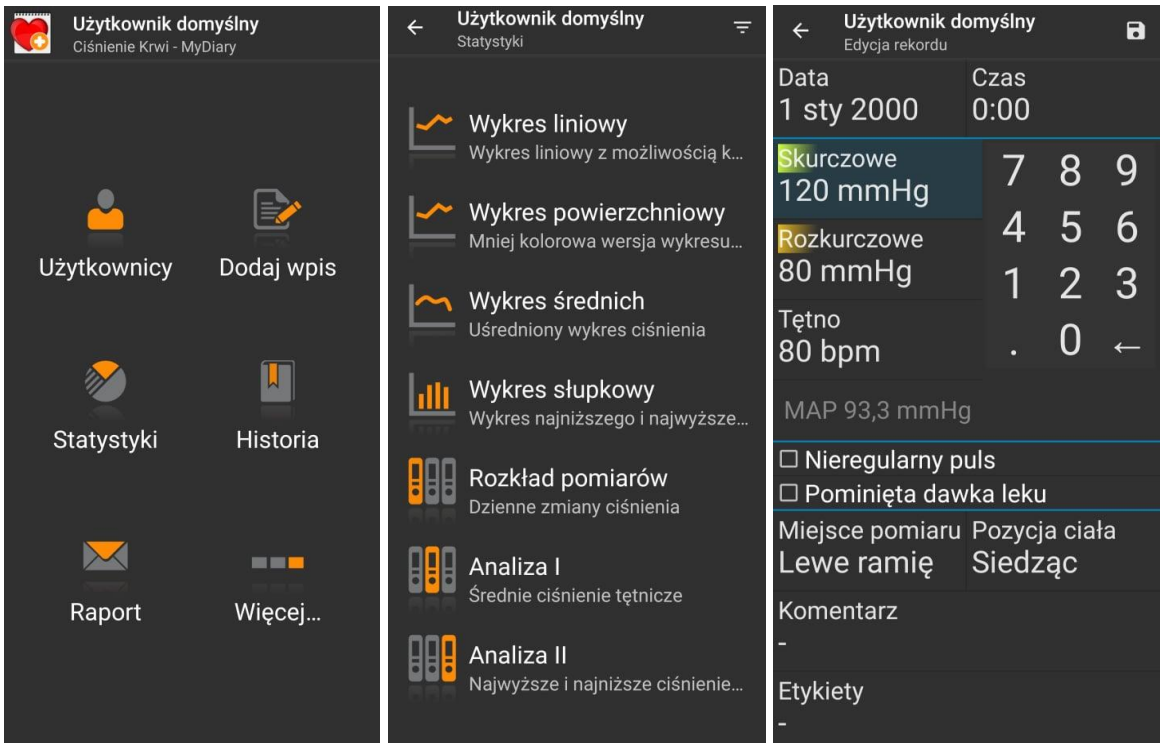
Rysunek 3. Zrzut ekranów (od lewej) - postępu i statystyk, panelu głównego oraz ekranu dodawania wyniku pomiaru ciśnienia krwi. Zrzuty ekranów zostały wykonane z ustawioną największą dostępną czcionką systemową.

MyTherapy to najbardziej przyjazna seniorom aplikacja z testowanych w ramach przygotowań do realizacji niniejszej pracy. Spełnia ona niemal wszystkie założenia wymienione w rozdziałach 2.1 i 2.2, nie występują w niej żadne błędy językowe i jest przystosowana do każdego rozmiaru czcionki systemowej.

3.2 Ciśnienie Krwi (MyDiary)

Ciśnienie krwi (MyDiary) to aplikacja utworzona przez brytyjską firmę Blip Software LTD. Od 2014 roku został ona pobrana ponad 1 000 000 razy ze Sklepu Play. [15]

Program umożliwia zapisywanie wyników pomiaru ciśnienia krwi z pulsem oraz poziomu cukru, wagi i temperatury przypisanych do pomiaru ciśnienia. Ponadto, można podać inne szczegóły pomiaru tj. miejsce pomiaru, pozycję ciała, komentarz oraz etykiety, w których użytkownik może wybrać takie wartości jak towarzyszące mu w danej chwili objawy, zażyte leki czy pora dnia uzależniona od posiłku. Aplikacja pozwala również na generację raportu w pisemnej bądź graficznej.



Rysunek 4. Zrzut ekranów (od lewej) - panelu głównego, panelu wyboru rodzaju prezentacji statystyk oraz ekranu dodawania wyniku pomiaru. Zrzuty ekranów zostały wykonane z ustawioną największą dostępną czcionką systemową.

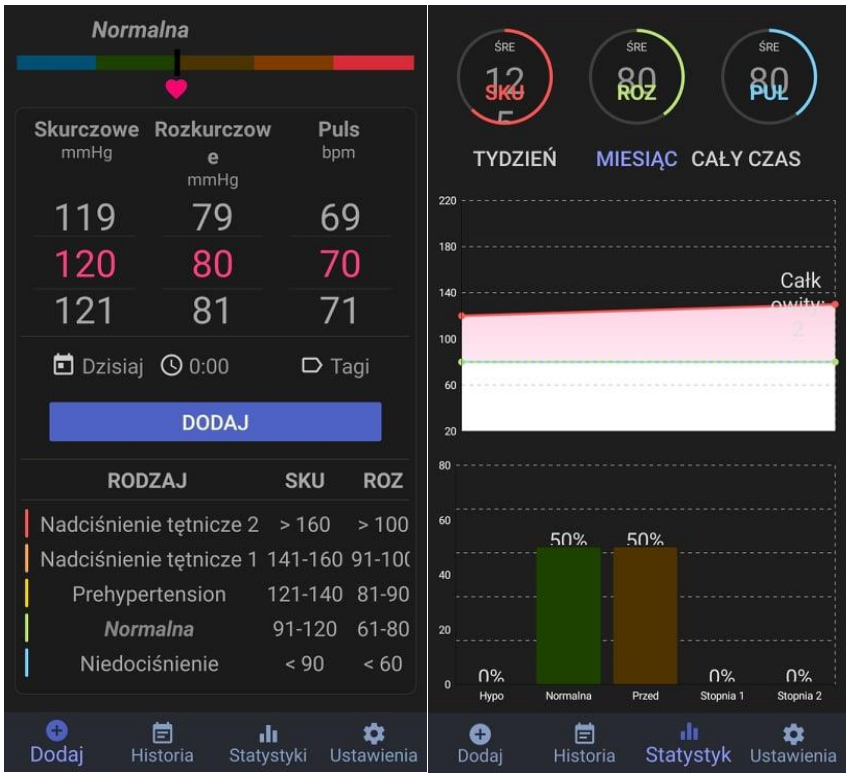
MyDiary spełnia wiele wymagań optymalnego dla seniorów design-u w kontekście widoków z menu czy listami wyborów użytkownika – panel główny (Rysunek 4) ma duże ikony menu z, co ważne, podpisami. Panel dodawania pomiaru chociaż spełnia część wymagań jest nieco chaotyczny. Sposób w jaki wprowadza się wyniki poszczególnych badań różni się od tego jak wprowadza się cyfry w innych aplikacjach dla systemów Android. W przypadku MyDiary, w celu podania wyniku, wybiera się cyfry z panelu obok pól z pomiarami – w większości dostępnych na rynku aplikacji dane wprowadza się z użyciem klawiatury numerycznej, która wyświetlana jest w formie modal-a bądź na dole ekranu w formie klawiatury numerycznej.

3.3 Dziennik ciśnienia krwi

Dziennik ciśnienia krwi jest aplikacją dostarczaną przez developera tworzącego pod pseudonimem Health & Fitness AI Lab. W Sklepie Play pojawia się pod koniec 2018 roku,

odkąd została pobrana ponad 1 000 000 razy. Warto zaznaczyć, że od kwietnia 2020 roku, nie jest ona aktualizowana. [16]

Program pozwala na zapisywanie wyników pomiarów ciśnienia krwi z pulsem oraz znaczników (*tag*) pozwalających określić położenie opaski ciśnieniomierza podczas pomiaru i pozycji badanego. Użytkownik ma też możliwość wprowadzenia własnych znaczników np. określających samopoczucie, dolegliwości występujące danego dnia czy inne wskazane przez lekarza parametry. Co wyróżnia Dziennik ciśnienia na tle innych testowanych aplikacji to podgląd klasyfikacji wyniku pomiaru – czy klasyfikuje się do nadciśnienia tętniczego (pasek na górze panelu głównego Rysunek 5)



Rysunek 5. Zrzut ekranów (od lewej) – dodawania wyników pomiaru i zarazem panelu głównego oraz ekranu dodawania wyniku pomiaru ciśnienia krwi. Zrzuty ekranów zostały wykonane z ustawioną największą dostępną czcionką systemową.

Testowana aplikacja nie jest idealnym rozwiązaniem dla osób starszych. Po ustawieniu największej czcionki systemowej aplikacja w wielu miejscach staje się nieczytelna – czcionka może być za mała dla osób z wadą wzroku, ale również w niektórych miejscach czcionka została powiększona co zupełnie uniemożliwia odczytanie treści np. na ekranie ze statystykami (średnie wartości na górze ekranu Rysunek 5). Poza tym, chociaż wspomniany wcześniej podgląd klasyfikacji wydaje się użytecznym rozwiązaniem, w wersji polskiej występują w nim błędy, co również negatywnie wpływa na odbiór aplikacji.

3.4 Porównanie aplikacji

Porównując ze sobą wcześniej opisane aplikacje należy pamiętać, że nie każda z nich została zaprogramowana przez firmy specjalizujące się w dostarczaniu tego typu oprogramowania np. aplikacja Dziennik ciśnienia krwi została zaprogramowana przez jednoosobową działalność. Najbardziej rozwiniętym z rozpatrywanych programów jest zdecydowanie MyTherapy – aplikacja jako jedyna jest zupełnie kompletna nawet dla polskiego użytkownika. Rozwiązanie oferowane przez smartpatient wydaje się być najlepsze – jest to firma, która specjalizuje się w dostarczaniu tego typu oprogramowania i współpracuje z globalnymi liderami przemysłu farmaceutycznego.

Większość z dostępnych w Sklepie Play aplikacji jest do siebie podobnych, oferują bardzo zbliżone do siebie funkcjonalności, dlatego testując je zwracano głównie uwagę na aspekty wizualne, gdyż w przypadku rozpatrywanej w niniejszej pracy grupie docelowej ma to niemal największe znaczenie.

4 Założenia pracy

Uwzględniając wnioski uzyskane podczas testów rozwiązań dostępnych na rynku oraz argumenty wynikające z analizy materiałów naukowych - aspektów przytoczonych rozdziałach 2.1 i 2.2, ustalono następujące założenia projektowe:

- Aplikacja ma być kompatybilna z systemem operacyjnym Android.
- Program ma być przeznaczony do użytku przez osoby w podeszłym wieku.
- System ma umożliwić przechowywanie danych pomiarów ciśnienia tętniczego krwi, poziomu glukozy we krwi oraz wagi.
- Dane przechowywane będą przechowywane w bazie danych, która połączona jest z serwerem.
- Odpowiedni interfejs graficzny dla grupy docelowej (duży rozmiar czcionki, kontrastująca kolorystyka) z jak najmniej skomplikowanymi ścieżkami użytkownika.
- Aplikacja ma umożliwić udostępnianie wyników wyznaczonemu wcześniej opiekunowi.
- Umożliwienie ustawienia przypomnień w formie powiadomień *push*, wywoływanych o wybranej przez użytkownika godzinie.
- Aplikacja ma dbać o bezpieczeństwo danych użytkownika.

Powyższe założenia dla aplikacji stanowią podsumowanie wymagań przedstawionych z wykorzystaniem techniki *User Centered Design Canva* a przytoczonych na planszy (Rysunek 1).

5 Technologie

Dzienniczek Seniora został napisany w języku *Python*, z użyciem platformy *Django*, w połączeniu z językiem *JavaScript* i framework-iem *React Native*. Wybrane technologie to jedne z najbardziej nowoczesnych i popularnych rozwiązań. Zważywszy na wcześniejszą znajomość webowego framework-u React.js, postanowiono napisać część kliencką projektu w analogicznej dla aplikacji mobilnych platformie, aby zapoznać się z metodyką tworzenia rozwiązań dla systemu Android. Warto zaznaczyć, że zarówno Django, jak i React Native mają skrupulatnie opracowane dokumentacje, które w bardzo przystępny sposób prezentują ich możliwości.

5.1 Serwer – Python

Python jest językiem programowania stworzonym przez Guido van Rossum'a w latach 90-tych ubiegłego wieku. Język ten niezwykle rozwija się na przestrzeni lat, a to za sprawą wciąż rosnącej popularności, na którą na pewno duży wpływ ma jego prosta składnia i fakt, że Python to język, który łączy najlepsze cechy i rozwiązania innych języków. Warto też wspomnieć, że jest to oprogramowanie otwarte, co oznacza, że użytkownicy mają dostęp do kodu źródłowego oraz mogą go dowolnie zmieniać i rozpowszechniać. [17] Python to język obiektowy, wysokopoziomowy i interpretowany – kod w nim napisany najpierw jest kompilowany do postaci pośredniej (bitowej), zaś następnie jest wykonywany przez *wirtualną maszynę Python'a*. [18] Programy napisane językach obiektowych mogą opierać się na strukturach danych zwanych obiektami – będącymi połączeniem danych i metod na tych danych operujących oraz pozwalają na zabieg zwany dziedziczeniem, który ogranicza liczbę powtarzalnych bloków w kodzie źródłowym aplikacji, co jest zgodne z zasadą programowania *Don't Reapeat Yourself (DRY)*.

Część serwerowa Dzienniczka Seniora została napisana w języku Python z wykorzystaniem bibliotek umożliwiających komunikację ze stroną kliencką przy użyciu protokołu HTTP, obsługę bazy danych oraz uwierzytelnienie.

5.1.1 Django

Django jest wysokopoziomowym *framework*-iem *web*-owym napisanym w języku Python, jest bezpłatnym, otwartym oprogramowaniem. Został stworzony przez doświadczonych developerów, którzy postanowili stworzyć platformę programistyczną, która będzie generowała szkielet aplikacji internetowej, który jest powtarzalny w niemal każdym tego typu programie. Django ma na celu przyspieszenie pracy programistów, tak aby mogli się skupić funkcjonalności aplikacji, a nie na niskopoziomowym działaniu serwera. [19] Platforma między innymi pozwala na wygenerowanie bazy danych na podstawie odpowiednio zadeklarowanych modeli w postaci klas i jej pól – klasy odpowiadają kolejno tabelom, zaś pola – kolumnom. Django ma również wbudowaną możliwość wygenerowania panelu administratora z gotowym interfejsem graficznym.

Ponadto, dba ona o bezpieczeństwo przetwarzanych danych – zapewnia ochronę przed takimi atakami jak *cross site scripting* (XSS), *cross site request forgery* (CSRF) i *SQL injection*. [20] XSS jest jednym z najczęściej występujących ataków – „polega na „osadzeniu" na stronie docelowej niekontrolowanego kodu JavaScript." [21] CSRF to atak polegający na wymuszeniu (bez wiedzy użytkownika) wysłania zapytania do serwera w kontekście aktualnie zalogowanego użytkownika. Kolejnym z przytoczonych typów ataków jest SQL injection polegające na manipulacji zapytaniem SQL z wykorzystaniem danych przekazywanych przez użytkownika, może ono doprowadzić do np. ujawnienia, modyfikacji czy skasowania rekordów obsługiwanej bazy danych. [22]

5.1.2 Django REST Framework

Django REST Framework to platforma umożliwiająca łatwe zaimplementowanie szczególnego rodzaju API jakim jest REST(ful) API, czyli *Representational State Transfer API*. Dzięki niemu serwer z klientem mogą porozumiewać się za pośrednictwem protokołu HTTP i za pomocą jego metod tj. GET (pobieranie), PUT (aktualizowanie), POST (wysyłanie) oraz DELETE (usuwanie). [23] Django REST Framework umożliwia szybki proces serializacji polegający na konwersji modelu zdefiniowanego przy użyciu Django do obiektu o wybranym formacie – w niniejszej pracy zastosowano format JSON. Proces ten ułatwia tworzenie odpowiedzi przez serwer. Ponadto, platforma dostarcza liczne klasy ułatwiające implementację widoków i przypisanym im funkcji – jest to zgodne ze wcześniej już wspomnianą zasadą DRY.

5.1.3 Simple JWT

Simple JWT to wtyczka do Django REST Framework umożliwiająca korzystanie z *JSON Web Token*-ów (JWT) w celach uwierzytelnienia użytkowników. Rozszerzenie umożliwia dokładne określenie przekazywanych w tokenach (żetonach) informacji, sposobu ich kodowania oraz wielu innych parametrów takich jak ważność tokenów. Simple JWT dostarcza również wbudowane widoki pobierające żetony przy logowaniu oraz wykorzytujące tokeny odświeżenia. Szczegółowy opis działania JSON Web Token znajduje się w rozdziale 6.4.2.

5.2 Aplikacja mobilna – React Native

Część kliencka Dzienniczka Seniora została napisana w języku JavaScript z wykorzystaniem framework-u React Native. Jest to biblioteka umożliwiająca tworzenie multiplatformowych aplikacji mobilnych - zarówno dla systemu Android, jak i iOS. Podczas uruchamiania napisanego wcześniej programu wykonywany jest kod JavaScript, który następnie jest tłumaczony na kod natywny danego systemu operacyjnego. [24] Platforma dostarcza wiele wbudowanych komponentów, które działają identycznie niezależnie od urządzenia na jakim są uruchamiane - dzieje się tak ponieważ wspomniane elementy „opakowują" kod natywny.

React Native jest platformą stworzoną przez firmę Facebook, która stale rozwija swoje oprogramowanie, gdyż wszystkie dostarczane przez nią aplikację opierają się właśnie na wspomnianym framework-u. Niesie to za sobą liczne zalety, ale i również konsekwencje – jeżeli firma zdecyduje się zmienić obecnie stosowaną technologię, może to oznaczyć koniec biblioteki, gdyż nie będzie już ona rozwijana. Jednak w chwili obecnej React Native znajduje się w grupie najczęściej wybieranych framework-ów do międzyplatformowego tworzenia aplikacji mobilnych (był numerem jeden w 2019 i 2020 roku). [25]

5.2.1 Biblioteki

Większość elementów interfejsu graficznego w Dzienniczku Seniora opiera się na bibliotece React Native Paper. Elementy, które ona oferuje są zgodne z założeniami stylu graficznego Material Design, jednocześnie umożliwiając ich łatwą modyfikację. Pozostałymi bibliotekami użytymi podczas realizacji części klienckiej aplikacji były m.in.:

- React Native AsyncStorage – biblioteka wykorzystana do osługi „magazynu lokalnego” [26]
- React Native DateTimePicker – biblioteka dostarczająca komponent do wyboru godziny [27]
- Native Stack Navigator – biblioteka definiująca sposób przemieszczania się między poszczególnymi widokami [28]
- Axios – biblioteka umożliwiającą przesyłanie zapytań i odbierania odpowiedzi od serwera [29]
- Victory Native – biblioteka ułatwiająca generowanie wykresów [30]
- React Native Toast Message – biblioteka służąca do wyświetlania powiadomień typu *toast* (wykorzystywana głównie do informowania użytkownika o stanie wysyłanych żądań czy walidacji) [31]
- React Native Push Notifications – biblioteka umożliwiająca wysyłanie powiadomień *push* [32]
- React Navigation – biblioteka dostarczająca funkcje sprawdzającą obecny stan ekranu

5.3 Formatowanie kodu

W celu zapewnienia lepszej czytelności kodu, podczas realizacji pracy korzystano z tak zwanych *formatter*’ów. Narzędzia te pozwalają na automatyczne formatowanie kodu, nie wpływając na jego finalne działanie. W aplikacji, będącej przedmiotem tej pracy, użyto dwóch oprogramowań tego typu – Prettier dla części klienckiej i Autopep8 dla części serwerowej. Oba narzędzia oferują wtyczki do programu Visual Studio Code.

5.4 Narzędzia

Aplikację będącą tematem niniejszej pracy utworzono w edytorze *Visual Studio Code* oferowanym przez firmę Microsoft. Oprogramowanie jest kompatybilne z

najpopularniejszymi systemami operacyjnymi – Windows, macOS oraz Linuks. Jest szczególnie zalecane do tworzenia aplikacji webowych z wykorzystaniem języka JavaScript. Środowisko to jest prężnie rozwijane i używane do programowania w niemal każdym dostępnym języku. Posiada też bogaty zasób wtyczek rozszerzających jego możliwości.

Android Studio jest środowiskiem przeznaczonym do tworzenia aplikacji dla systemu Android, dostarczanym przez firmę JetBrains oraz Google. Podobnie jak Visual Studio Code, oprogramowanie dostępne jest na takie systemy operacyjne jak Windows, macOS i Linuks. Językami, które obsługuje Android Studio są Java, C++ oraz Kotlin. Środowisko umożliwia także tworzenie i korzystanie z emulatorów, czyli programów naśladujących działanie systemu komputerowego [33], czy w tym przypadku smartfonu. Podczas tworzenia Dzienniczka Seniora, Android Studio zostało wykorzystane tylko w tym celu.

Postman to platforma pozwalająca na testowanie API (*application programming interface*) w łatwy i szybki sposób. Umożliwia między innymi zapisywanie treści oraz adresów zapytań do serwera z odpowiednimi nagłówkami np. w przypadku wymogu uwierzytelnienia. Obecnie działanie większości aplikacji internetowych na świecie opiera się o, testowane w aplikacji Postman, interfejsy programowania aplikacji. [23]

6 Implementacja

Działanie Dzienniczka Seniora opiera się na komunikacji aplikacji mobilnej z serwerem, który następnie łączy się z bazą danych. Pierwszy etap polega na połączeniu z użyciem protokołu HTTP, zaś drugi wykorzystuje zapytania SQL. (Rysunek 6) Warto zaznaczyć, że jest to rozwiązanie stosowane w większości programów mobilnych, które nie wymagają komunikacji w czasie rzeczywistym.

Rysunek 6. Struktura działania aplikacji wygenerowana przy użyciu strony diagrams.net. [34]

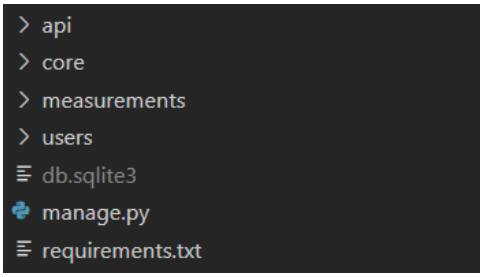
6.1 Serwer

Struktura projektu, odpowiadająca za część serwerową Dzienniczka Seniora, jest zgodna ze standardem proponowanym przez Django. Tworząc aplikację opierano się na hierarchii wygenerowanej automatycznie i przez cały proces przestrzegano nałożonych wcześniej zasad.

Katalogiem przechowującym kod źródłowy serwera jest backend, znajdują się w nim cztery podkatalogi:

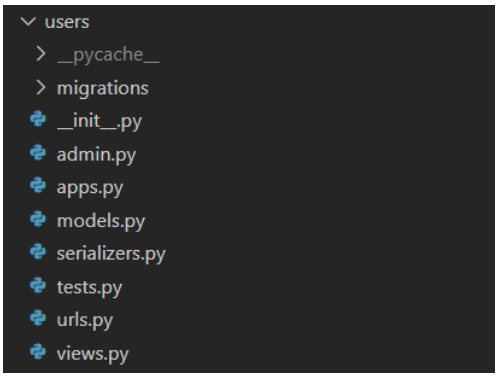
- `api` – katalog zawierający definicje widoków, serializatorów, adresów URL oraz funkcji niezbędnych do poprawnego przesyłania danych na linii klient-serwer
- `core` – katalog z głównymi plikami aplikacji, z ustawieniami oraz funkcjami pomocniczymi
- `measurements` – katalog przechowujący definicje modeli pomiarów i ich rejestracje w panelu administratora
- `users` – katalog z definicją pól klasy użytkownika, widoków i adresów związanych z modelem użytkownika (w tym metod uwierzytelniających)

Oprócz podkatalogów znajdują się tam również pliki: manage.py ze skryptami niezbędnymi do uruchomienia serwera, `requirements.txt`, w którym zapisane zostały wszystkie biblioteki niezbędne do prawidłowego działania aplikacji oraz plik bazy danych – `db.sqlite3`. (Rysunek 7)



Rysunek 7. Zrzut ekranu struktury części serwerowej projektu.
Zrzut ekranu został wykonany w programie Visual Studio Code.

Django, przy tworzeniu nowego projektu, generuje folder o wybranej nazwie (w tym przypadku katalog został nazwany `core`) oraz plik `manage.py`. Pozostałe katalogi są generowane później jako swego rodzaju aplikacje – każdy z nich jest generowany z podstawowymi plikami. Ponadto, w każdym z folderów aplikacji ręcznie dodano plik `serializers.py`, w którym zdefiniowane są serializatory. (Rysunek 8)

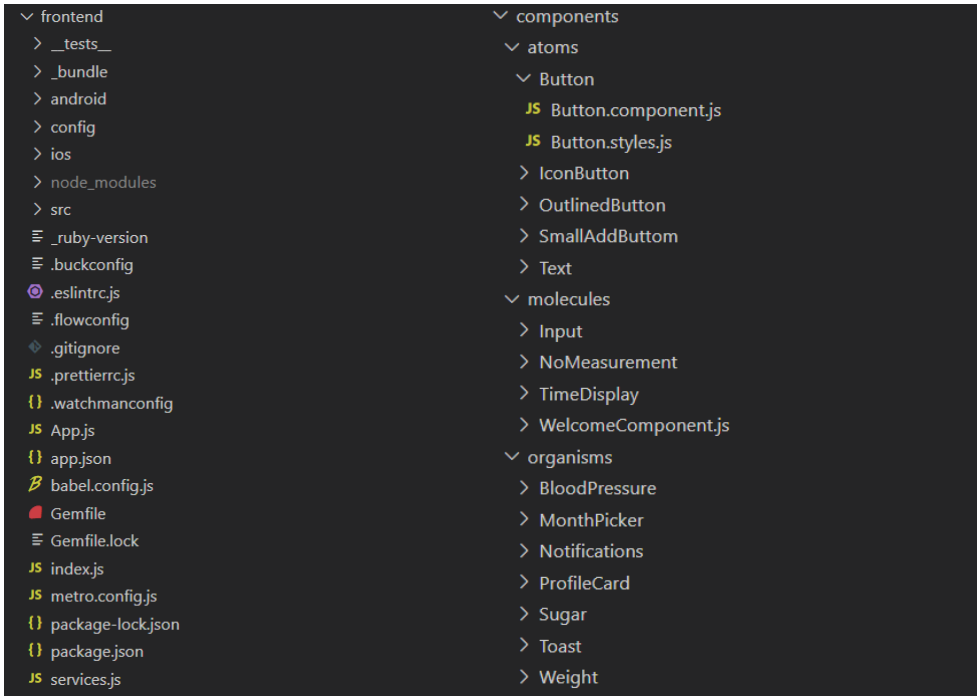


Rysunek 8. Zrzut ekranu struktury katalogu nowoutworzonej aplikacji generowanej przez Django.
Zrzut ekranu został wykonany w programie Visual Studio Code.

6.2 Aplikacja mobilna

Struktura plików kodu źródłowego w większej części została wygenerowana automatycznie przy tworzeniu projektu w React Native. Podkatalog `components`, zawierający definicję używanych w aplikacji elementów, został utworzony w oparciu o wskazówki Muskan Jain, programistki z firmy Habilelabs. [35] Zakłada się, że folder główny folder z komponentami powinien być podzielony na trzy podkatalogi – `atoms`, `molecules` i `organisms`. Kolejno każdy z nich zawiera coraz bardziej złożone elementy to znaczy, że w folderze `atoms` znajdą się podstawowe komponenty takie jak przyciski czy pola tekstowe, zaś w katalogu `molecules` – elementy złożone ze zdefiniowanych w folderze `atoms` komponentów i analogicznie dla katalogu `organisms`. Dodatkowo zastosowano zasadę tworzenia poszczególnych elementów w formie – folder nazwany tak samo jak tworzony komponent, a w nim dwa pliki – z component.js w nazwie definiujący

działanie oraz z dopiskiem `styles.js` w nazwie definiujący style nadane danemu komponentowi.

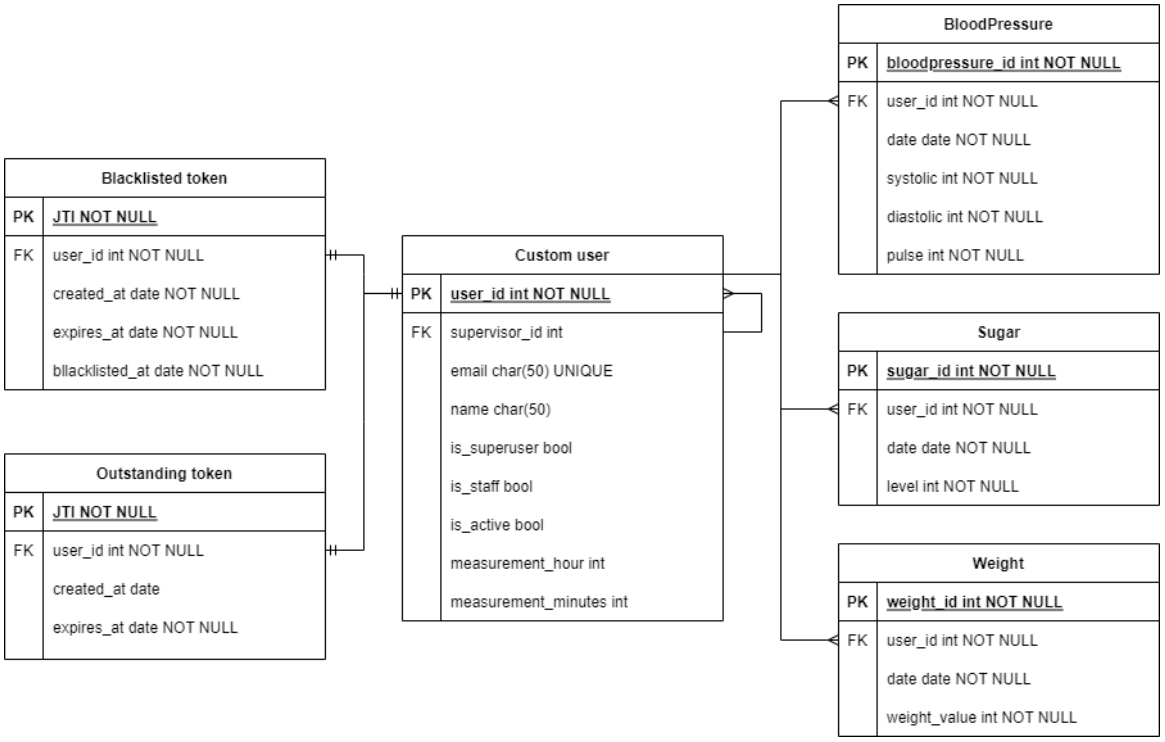


Rysunek 9. Zrzut ekranu struktury części klienckiej aplikacji (po lewej) oraz rozwiniętej struktury katalogu src/components (po prawej). Zrzut ekranu został wykonany w programie Visual Studio Code.

Oprócz wspomnianych powyżej katalogów, w folderze `src` znajduje się podkatalog `screens` zawierający kody źródłowe poszczególnych widoków aplikacji.

6.3 Baza danych

Dzienniczek Seniora wykorzystuje obecnie lokalną, relacyjną bazę danych *SQLite*. Jest to rozwiązanie proponowane przez Django, w pełni z nim kompatybilne. W bazie przechowuje informacje na temat użytkowników w tabeli *Custom users*, wyniki pomiarów w tabelach *Blood pressure*, *Sugar* i *Weight* oraz dwie tabele przechowujące *tokeny* (żetony) wykorzystywane do uwierzytelniania. Poniżej zamieszczono *diagram ER* wykorzystywanej w projekcie bazy danych. Schemat ten pokazuje podstawowe zależności zachodzące między poszczególnymi tabelami oraz ograniczenia, jakie zostały nałożone na poszczególne kolumny. (Rysunek 10) Dokładny opis relacji tabel zostanie przedstawiony w kolejnych rozdziałach.



Rysunek 10. Diagram ER bazy danych projektu wygenerowany przy użyciu strony diagrams.net. [34]

6.3.1 Model użytkownika

Tak jak wcześniej wspomniano (5.1.1), framework Django generuje bazę danych na podstawie zadeklarowanych klas w odpowiednich plikach (`models.py`). Na potrzeby Dzienniczka Seniora utworzono model użytkownika zawierający poniższe pola:

- *id* – klucz główny, służący do identyfikacji użytkownika
- adres email – pełniący funkcję loginu (musi być unikalny), pole jest wymagane
- *name* – postanowiono stworzyć taką kolumnę, aby pozwolić na spersonalizowanie komunikatów dla użytkownika po stronie klienckiej, pole jest wymagane
- *supervisor_id* – id innego użytkownika, któremu przydzielany jest dostęp do wyników pomiarów
- *is_superuser, is_staff, is_active* – pola decydujące o dostępach danego użytkownika
- *measurement_hour* – godzina wysyłania powiadomień
- *measurement_minutes* – uzupełnienie minut godziny wysyłania powiadomień

Implementacja powyższych reguł została zamieszczona poniżej. (Fragment kodu 1)

```python
class CustomUser(AbstractBaseUser, PermissionsMixin):
    id = models.AutoField(primary_key=True)
    email = models.EmailField(_('email address'), unique=True)
    name = models.CharField(max_length=50)
    supervisor_id = models.ForeignKey(
        'self', on_delete=models.SET_NULL, null=True)
    is_superuser = models.BooleanField(default=False)
    is_staff = models.BooleanField(default=False)
    is_active = models.BooleanField(default=False)
    measurement_hour = models.PositiveSmallIntegerField(
        null=True, default=None)
    measurement_minutes = models.PositiveSmallIntegerField(
        null=True, default=None)

    objects = CustomAccountManager()

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['name']

    def __str__(self):
        return self.name
```

Fragment kodu 1. Klasa definiująca tabelę użytkowników.

6.3.2 Modele pomiarów

Tworząc modele do przechowywania wyników pomiarów wykorzystano obiektowość języka Python. Zauważono, że w każdej tabeli powinny pojawić się pola z id użytkownika oraz datą pomiaru, dlatego, zgodnie z zasadą DRY, postanowiono skorzystać z mechanizmu dziedziczenia. Utworzono klasę *Measurement*, która ma dwa, wspomniane wcześniej pola oraz trzy klasy odpowiadające dostępnym pomiarom – *BloodPressure*, *Sugar* oraz *Weight.*

Klasa *BloodPressure* ma trzy pola:

- *systolic* – odpowiadające ciśnieniu skurczowemu [mmHg]
- *diastolic* – odpowiadające ciśnieniu rozkurczowemu [mmHg]
- *pulse* – odpowiadające wartości tętna [liczba uderzeń/minutę]

Klasa *Sugar* ma jedno pole:

- *level* – odpowiadające poziomowi glukozy we krwi [mg/dl]

Klasa *Weight* ma również jedno pole:

- *weight_value* – odpowiadające zmierzonej wadze [kg]

Wszystkie powyższe pola są wymagane i ustawiono w nich przedziały wartości jakie mogą osiągać, aby uniknąć nierealistycznych wyników w bazie danych. Granice zostały nałożone przy użyciu tak zwanych walidatorów - *MinValueValidator* oraz *MaxValueValidator*.

Każda z opisanych powyżej klas dziedziczy po klasie *Measurement*, która składa się z dwóch pól:

- *user_id* – klucz obcy, wskazujący na osobę, która dokonuje pomiaru, w przypadku usunięcia tego użytkownika, usuwane są też przypisane do niego pomiary
- *date* – data pomiaru

```
class Measurement(models.Model):
    date = models.DateField(auto_now=True)
    user = models.ForeignKey(
        get_user_model(), related_name="users", on_delete=models.CASCADE, blank=False)

    objects = models.Manager()


class BloodPressure(Measurement):
    systolic = models.PositiveSmallIntegerField(
        blank=False, validators=[MinValueValidator(70), MaxValueValidator(190)])
    diastolic = models.PositiveSmallIntegerField(
        blank=False, validators=[MinValueValidator(40), MaxValueValidator(120)])
    pulse = models.PositiveSmallIntegerField(
        blank=False, validators=[MinValueValidator(30), MaxValueValidator(250)])

    def __str__(self):
        return str(self.systolic) + ' / ' + str(self.diastolic) + ' pulse: ' + str(self.pulse)
```

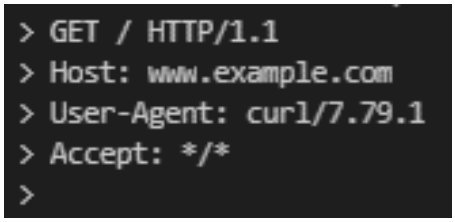
Fragment kodu 2. Klasa definiująca tabelę wyników pomiaru ciśnienia krwi.

Tabela 2. Wartości graniczne ustalone dla każdego typu pomiaru

	Jednostka	**Minimalna wartość**	**Maksymalna wartość**
Ciśnienie skurczowe	mmHg	70	190
Ciśnienie rozkurczowe [36]	mmHg	40	120
Puls [37]	l. ud./min	30	250
Poziom glukozy we krwi [38]	mg/dl	10	300
Waga	kg	20	250

6.4 Komunikacja klient-serwer

Komunikacja pomiędzy klientem a serwerem odbywa się przy użyciu API, a konkretnie REST API, którego podstawy działania zostały już opisane w rozdziale 5.1.2. Wspomniana technologia wykorzystuje protokół HTTP, który jest wykorzystywany w niemal wszystkich aplikacjach webowych dostępnych na rynku. Dzięki niemu, użytkownicy mogą wysyłać żądania do serwera w formie wiadomości, które mają ściśle określoną składnię:



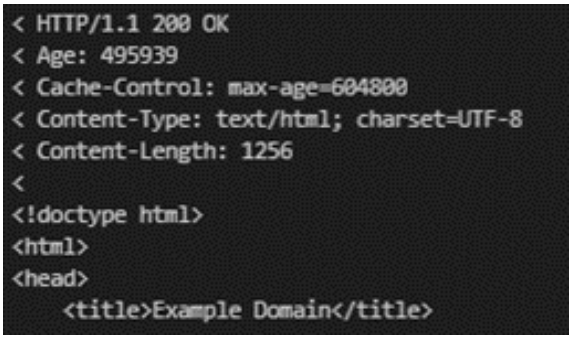
Rysunek 11 Przykład zapytania HTTP

- Linia określająca czasownik HTTP (nazwa wykorzystywanej metody), zasób (wołany adres) i wersję używanego protokołu
- Linie zawierające nagłówki (w nich np. przesyłane są dane niezbędne do uwierzytelnienia)
- Pusta linia wyznaczająca koniec sekcji z nagłówkami
- Ciało wiadomości – część opcjonalna

Podobnie prezentuje się schemat odpowiedzi od serwera – różni się on jedynie pierwszą linijką, w której to znajduje się wersja protokołu wraz ze statusem odpowiedzi. (Rysunek 12) Statusy HTTP są przesyłane w formie liczby trzy cyfrowej, każda odpowiada innemu rodzajowi odpowiedzi. Jedne z najczęściej występujących statusów to:

- *200 OK* świadczący o poprawnym przetworzeniu zapytania
- *400 Bad Request* oznaczający błąd w przesyłanym zapytaniu (informuje jednocześnie o nie przetworzeniu zapytania)
- *404 Not Found* – żądany zasób nie istnieje
- *500 Internal Server Error* świadczący o błędzie serwera

Pierwsza cyfra statusu już przekazuje informacje o stanie zapytania – cyfra 2 zawsze informuje o poprawnym przetworzeniu zapytania, cyfra 4 wskazuje na błąd po stronie klienta, zaś cyfra 5 – po stronie serwera. [39]



Rysunek 12 Przykład odpowiedzi HTTP

6.4.1 REST API

RESTful API to rodzaj interfejsu wykorzystujący opisany w poprzednim rozdziale protokół HTTP do przesyłania zasobów między aplikacjami. Bardzo ważne jest, aby, korzystając z tej technologii, w pełni rozdzielić od siebie część serwerową, w której znajduje się zasób, i kliencką, która ów zasób chce uzyskać – powinny one działać niezależnie od siebie. [40]

W Dzienniczku Seniora ciała wiadomości są przesyłane w formacie JSON, dlatego niezbędne było zastosowanie wcześniej wspomnianych serializatorów (5.1.2.). Działają one na kształt tłumacza – ze wspomnianego formatu na język zrozumiały dla Django i co za tym idzie, dla bazy danych. (Fragment kodu 3)

```
class SugarSerializer(serializers.ModelSerializer):
    user = serializers.SlugRelatedField(
        allow_null=False,
        required=True,
        slug_field="id",
        queryset=CustomUser.objects.all(),
    )

    class Meta:
        model = Sugar
        fields = ('id', 'user', 'date', 'level')
```

Fragment kodu 3 Kod definiujący serializator klasy Sugar.

W celu pozyskania danych z backend'u, należy wysłać żądanie z metodą GET. Django REST Framework oferuje szeroką gamę gotowych już widoków, które implementują obsługę najczęściej używanych zapytań. W aplikacji będącej przedmiotem tej pracy, wykorzystano kilka z nich – między innymi *ListCreateAPIView* i *ListAPIView*. Oba zapewniają zwracanie żądanego zasobu, z czym pierwszy z nich pozwala też na dodawanie nowych rekordów do bazy danych. W kodzie źródłowym wystarczy zdefiniować takie parametry jak *serializer_class* (wskazująca na serializator obiektów, które żądanie ma zwrócić) oraz zdefiniować dwie metody *get_queryset* oraz post (tylko w przypadku widoków umożliwiających zapytania z tą metodą). Analogicznie wygląda to w przypadku tworzenia widoków umożliwiających korzystanie z pozostałych metod HTTP. Dostępne widoki *ListCreateAPIView* zostały wykorzystane do utworzenia widoków list wyników pomiaru ciśnienia krwi, wagi (Fragment kodu 4) i poziomu cukru. Wszystkie widoki zostały zdefiniowane w wygenerowanych przez Django plikach `views.py`.

```python
class WeightList(generics.ListCreateAPIView):
    permission_classes = [MeasurementPermission]
    serializer_class = WeightSerializer

    def get_queryset(self):
        return custom_get_queryset(self, Weight)

    def post(self, request):
        return custom_post(self=self, request=request, serializer_class=WeightSerializer)
```

Fragment kodu 4. Kod definiujący klasę WeightList będącą widokiem umożliwiającym pobieranie oraz dodawanie zasobów tabeli Weight. Analogicznie zaimplementowano widoki dla pozostałych pomiarów.

W celu uniknięcia powtarzania kodu, postanowiono zdefiniować funkcje zwracające odpowiedzi serwera w zależności od poprawności otrzymanego wcześniej zapytania. (Fragment kodu 5)

```python
def custom_post(self, request, serializer_class):
    request_data = {'user': get_user_id(self.request)}
    # dodanie do ciała zapytania id jego autora
    request_data.update(request.data)
    # przekazanie ciała do serializatora w celu sprawdzenia
    # ich poprawności
    serializer = serializer_class(data=request_data)
    if serializer.is_valid():
        serializer.save()
        return Response('Dodano pomiar')
    else:
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)


def custom_get_queryset(self, model):
    # sprawdzenie, czy użytkownik przesyłający żądanie jest zalogowany
    if get_user_id(self.request):
        # jeśli tak, zwracana jest lista zasobów, które przypisane są
        # do autora zapytania
        return model.objects.filter(user_id=get_user_id(self.request))
    else:
        return [] # jeśli nie, zwracana jest pusta lista
```

Fragment kodu 5. Kod definiujący metody używane podczas tworzenia widoków dla większości tabel dostępnych w bazie danych.

Do implementacji panelu opiekuna wykorzystano dostępny widok *ListAPIView*, gdyż zgodnie z założeniami opiekun nie ma możliwości dodawania podopiecznych czy ich wyników pomiarów, dlatego jedyną dostępną metodą HTTP w tych widokach jest GET. (Fragment kodu 6 oraz Fragment kodu 7)

```python
class SeniorsList(generics.ListAPIView):
    serializer_class = CustomUserSerializer

    def get_queryset(self):
        if get_user_id(self.request):
            return CustomUser.objects.filter(supervisor_id=get_user_id(self.request))
        else:
            return []
```

Fragment kodu 6. Kod definiujący klasę SeniorsList będącą widokiem umożliwiającym pobieranie zasobów tabeli CustomUsers, spełniających określone warunki (w tym przypadku pobierani są jedynie użytkownicy, których opiekunem jest autor zapytania).

```python
def custom_senior_list_queryset(self, model):
    try:
        # sprawdzenie, czy użytkownik ma jakichkolwiek podopiecznych
        CustomUser.objects.get(
            pk=self.kwargs['pk'], supervisor_id=get_user_id(self.request))
    except CustomUser.DoesNotExist:
        return [] # jeśli nie, zwracana jest pusta tablica
    # jeśli tak, zwracana jest lista podopiecznych
    return model.objects.filter(user_id=self.kwargs['pk'])
```

Fragment kodu 7. Kod definiujący funkcję zwracającą podopiecznych autora zapytania.

Kolejnym z wykorzystanych widoków oferowanych przez Django REST Framework jest *APIView*. Został on wykorzystany w widokach, w których wymagane było użycie metody PATCH np. w widoku dodawania opiekuna. (Fragment kodu 8) Funkcja ta pozwala na aktualizowanie rekordów w bazie danych podobnie jak metoda PUT, jednak PATCH nie wymaga przesłania wszystkich pól modyfikowanego obiektu – wystarczy przesłać nową wartość wybranego pola wraz z jego nazwą. Zdecydowano wykorzystać właśnie tę metodę, aby uniknąć potrzeby przechowywania i przesyłania wszystkich informacji o użytkowniku.

```python
class YourSupervisor(APIView):
    permission_classes = [AllowAny]

    def get(self, request, format=None):
        user = CustomUser.objects.get(pk=get_user_id(self.request))
        # sprawdzenie, czy autor zapytania ma opiekuna
        if user.supervisor_id:
            # jeśli tak, zwracane są o jego informacje
            supervisor = CustomUser.objects.get(pk=user.supervisor_id.pk)
            serializer = CustomUserSerializer(supervisor, many=False)
            return Response(serializer.data)
        else:
            # jeśli nie, przsynałana jest poniższa wiadomość
            return Response({'name': 'Brak opiekuna', 'email': ''})

    def patch(self, request, format=None):
        user = CustomUser.objects.get(pk=get_user_id(self.request))
        try:
            # sprawdzenie, czy istnieje użytkownik z wprowadzonym adresem email
            supervisor = CustomUser.objects.get(
                email=request.data['supervisor_email'])
            data = {'supervisor_id': supervisor.pk}
        except CustomUser.DoesNotExist:
            return Response('Nie znaleziono opiekuna z podanym adresem email', status=status.HTTP_400_BAD_REQUEST)
        # jeżeli użytkownik istnieje, konto autora zapytania zostaje zaktualizowane o podanego opiekuna
        serializer = CustomUserSerializer(user, data=data, partial=True)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Fragment kodu 8. Kod definiujący widok dotyczący opiekuna użytkownika wysyłającego żądanie.

6.4.2 Uwierzytelnianie z użyciem JWT

W celu zapewnienia bezpieczeństwa danym przesyłanym przez użytkowników, zdecydowano wykorzystać mechanizm uwierzytelniania opierający się na użyciu JSON Web Token'ów. Są to obiekty składające się z trzech części, rozdzielonych kropką [41]:

- *nagłówek* (header) – określający typ tokenu oraz algorytm szyfrowania
- *zawartość* (payload) – przechowująca dane o użytkowniku i tokenie
- *podpis* – zawierający zaszyfrowany nagłówek, zawartość oraz klucz publiczny lub prywatny

Odróżnia się dwa typy JWT – *dostępu* (access) oraz *odświeżania* (refresh). Pierwszy z nich jest przesyłany w nagłówku zapytania HTTP w kategorii Authorization z tytułem 'Bearer', odpowiada on za autoryzację podczas trwania bieżącej sesji. Dzięki niemu, przy wysyłaniu poszczególnych zapytań do serwera, za każdym razem użytkownik nie jest zmuszony podawać swoich danych – są one zawarte w owym tokenie. Co ważne, traci on swoją ważność po zakończeniu sesji lub po upłynięciu określonego w ustawieniach określonego czasu (w Dzienniczku Seniora jest to 5 minut). Bardzo często zdarza się, że użytkownicy korzystają z aplikacji dłużej niż czas żywotności żetonu dostępu, wtedy wykorzystywany jest drugi typ JWT – odświeżenia. Ma znacznie on dłuższą ważność - w Dzienniczku Seniora 30 dni, co oznacza, że użytkownik, który się wcześniej nie wyloguje, będzie poproszony o kolejne podanie danych autoryzacyjnych dopiero po 30 dniach od poprzedniego logowania. W przypadku, gdy serwer zwraca informację o nieprawidłowym

tokenie, oznaczającą najczęściej, że żeton, w nagłówku zapytania HTTP (dostępu), stracił ważność, przesyłane jest zapytanie zawierające w ciele token odświeżenia – jeśli jest on poprawny, serwer zwróci nowy token dostępu.

JWT wykorzystywane przez aplikację są przechowywane w bazie danych – ważne tokeny w tabeli *Outstanding tokens*, zaś te nieaktualne – w tabeli *Blacklisted tokens*. Szczegółowe ustawienia tokenów używanych w Dzienniczku Seniora zostały określone w pliku `settings.py` w folderze `core`.

```
SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=5),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=1),
    'ROTATE_REFRESH_TOKENS': False,
    'BLACKLIST_AFTER_ROTATION': False,
    'UPDATE_LAST_LOGIN': False,

    'ALGORITHM': 'HS256',
    'SIGNING_KEY': settings.SECRET_KEY,
```

Fragment kodu 9. Kod definiujący konfigurację wykorzystywanych tokenów, fragment pliku settings.py.

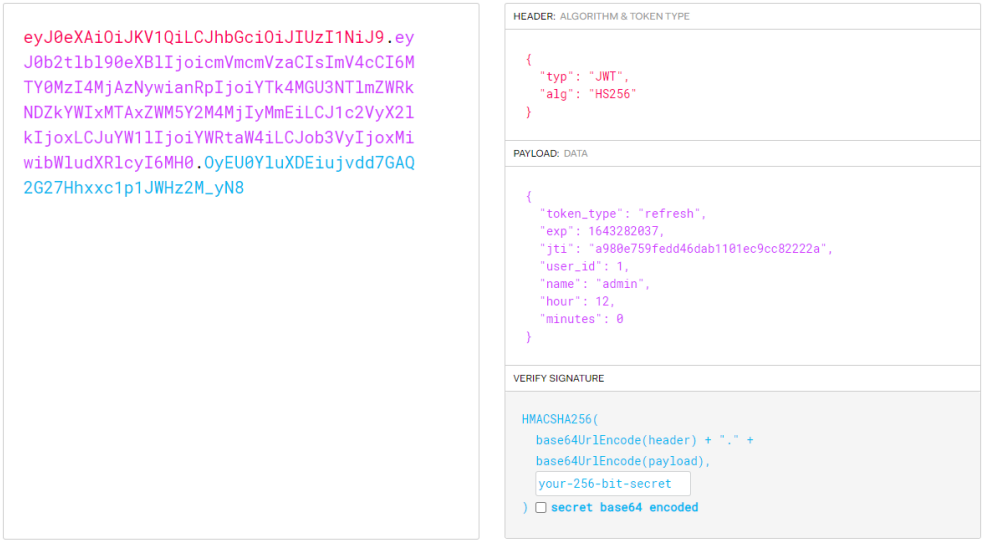
W trakcie logowania użytkownik jest proszony o podanie adresu email oraz hasła – te dane przesyłane są na adres `api/user/login/` metodą POST, bez ustawienia nagłówka autoryzacji, gdyż logują się jeszcze niezidentyfikowani wcześniej użytkownicy. Po przesłaniu poprawnych danych, autorowi zapytania zwracana jest odpowiedź zawierająca tokeny obu typów, imię oraz szczegóły godziny powiadomienia. (Rysunek 13)


```
{
    "refresh": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
        eyJ0b2tlbl90eXBlIjoicmVmcmVzaCIsImV4cCI6MTY0MzI4MjAzNywianRpIjoiYTk4MGU3NTlmZWRkNDZkYWIxMTAxZWM5Y2M4M
        jIyMmEiLCJ1c2VyX2lkIjoxLCJuYW1lIjoiYWRtaW4iLCJob3VyIjoxMiwibWludXRlcyI6MH0.
        OyEU0YluXDEiujvdd7GAQ2G27Hhxxc1p1JWHz2M_yN8",
    "access": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
        eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjQzMTk1OTM3LCJqdGkiOiJiYjE3MDUyOGFiMzY0ZjM3OWU0MTYxOWYzOWNhZ
        Dg2MSIsInVzZXJfaWQiOjEsIm5hbWUiOiJhZG1pbiIsImhvdXIiOjEyLCJtaW51dGVzIjowfQ.
        qYb4Vj6tjA6etom_bLl4Y_sRCIj_Z9JgJAqkd-bKKhY",
    "name": "admin",
    "hour": 12,
    "minutes": 0
}
```


Rysunek 13. Zrzut ekranu odpowiedzi serwera na poprawne dane logowania użytkownika o imieniu "admin". Zrzut ekranu został wykonany w aplikacji Postman.

```python
class CustomTokenObtainPairSerializer(TokenObtainPairSerializer):
    def validate(self, attrs):
        data = super().validate(attrs)
        refresh = self.get_token(self.user)
        data['refresh'] = str(refresh)
        data['access'] = str(refresh.access_token)
        data['name'] = self.user.name
        data['hour'] = self.user.measurement_hour
        data['minutes'] = self.user.measurement_minutes
        return data

    @classmethod
    def get_token(cls, user):
        token = super().get_token(user)
        token['name'] = user.name
        token['hour'] = user.measurement_hour
        token['minutes'] = user.measurement_minutes
        return token
```

Fragment kodu 10. Kod definiujący serializator dla tokenów.



Rysunek 14 Zrzut ekranu odkodowanego tokenu typu refresh.
Zrzut ekranu został wykonany na stronie jwt.io [42]

Jeśli użytkownik decyduje się wylogować się z aplikacji, przypisane bieżącej sesji żetony tracą swoją ważność. Oba z nich zostają przeniesione w bazie danych z tabeli *Outstanding tokens* do *Blacklisted tokens*. Użytkownik zostaje wylogowany poprzez wysłanie zapytania POST z tokenem odświeżenia w ciele na adres `user/logout/blacklist/`.

```python
class BlacklistTokenUpdateView(APIView):
    permission_classes = [AllowAny]
    authentication_classes = ()

    def post(self, request):
        try:
            # sprawdzenie, czy przesłany token jest ważny
            token = RefreshToken(request.data["refresh_token"])
            # jeśli tak, jest on unieważniany
            token.blacklist()
            return Response(status=status.HTTP_205_RESET_CONTENT)
        except Exception as e:
            print(e)
            return Response(status=status.HTTP_400_BAD_REQUEST)
```

Fragment kodu 11. Kod definiujący widok niezbędny do wylogowania użytkownika.

Po stronie klienckiej, żetony są przechowywane w magazynie lokalnym. (Fragment kodu 12) Wszystkie operacje związane z wysyłaniem zapytań są zdefiniowane w pliku `services.js` – w tym definicja tworzenia nagłówka Authorization z odpowiednim tokenem. (Fragment kodu 13)

```javascript
export const Login = ( payload, navigation ) => {
    axios
    .post(LOGIN_URL, payload)
    .then(response => {
      const name = response.data.name;
      const hour = response.data.hour ? response.data.hour : '';
      const minutes = response.data.minutes ? response.data.minutes : '';

      const refreshToken = response.data.refresh;
      const accessToken = response.data.access;

      const token = {
        access: accessToken,
        refresh: refreshToken
      }

      setConfig(token);

      storeStringValueInLocalStorage('refresh', refreshToken)
      storeStringValueInLocalStorage('name', name)
      storeStringValueInLocalStorage('schedule_hour', hour.toString())
      storeStringValueInLocalStorage('schedule_minutes', minutes.toString())

      successToast('Pomyślnie zalogowano.')
      navigation.navigate('Home')
    })
```

Fragment kodu 12. Kod definiujący metodę logowania użytkownika po stronie klienckiej.

```
let config = {};
const setConfig = (token) => {
  config = {
    headers: {
      "Authorization" : "Bearer " + token.access
    }
  }
}
```

Fragment kodu 13. Kod definiujący zmienną config przechowującą informacje o używanym nagłówku autoryzacji oraz metodę setConfig odpowiadającą za ustawianie wartości wspomnianego nagłówka.

Metodą odpowiadającą za używanie tokenów odświeżenia jest *refreshToken*. Pobiera ona wartość przypisaną do klucza ‘refresh’ z magazynu lokalnego i przesyła ją w ciele zapytania POST do serwera. Jeśli zapytanie zostanie poprawnie przetworzone, metoda ta zapisuje nowe wartości tokenów w zmiennej *config* oraz w magazynie lokalnym.

```
export const refreshToken = async ({ navigation }) => {
  const payload = { 'refresh': await getStringValueFromLocalStorage('refresh') }
  axios
  .post(REFRESH_TOKEN_URL, payload)
  .then(response => {
    const accessToken = response.data.access;
    const token = {
      access: accessToken,
      refresh: getStringValueFromLocalStorage('refresh')
    }
    setConfig(token);
  })
```

Fragment kodu 14. Kod definiujący metodę refreshToken.

6.4.3 Adresy URL

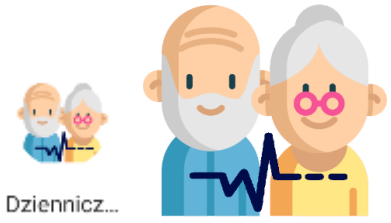
Wszystkie zapytania opisane w poprzednich rozdziałach muszą być wysyłane na odpowiedni adres URL. Wszystkie punkty końcowe zostały zadeklarowane w wygenerowanych przez Django plikach `urls.py`. Importuje się w nich wcześniej zdefiniowane widoki i następnie przypisuje się je do odpowiednich adresów.

```
urlpatterns = [
    path('measurements/blood-pressure/<int:pk>/',
        BloodPressureDetail.as_view(), name='blood-pressure-detail'),
    path('measurements/blood-pressure/',
        BloodPressureList.as_view(), name='blood-pressure-list'),
    path('measurements/sugar/<int:pk>/',
        SugarDetail.as_view(), name='sugar-detail'),
    path('measurements/sugar/', SugarList.as_view(), name='sugar'),
    path('measurements/weight/<int:pk>/',
        WeightDetail.as_view(), name='weight-detail'),
    path('measurements/weight/', WeightList.as_view(), name='weight-list'),

    path('home-page/', CurrentData.as_view(), name='home-page'),

    path('supervisor/users/', SeniorsList.as_view(), name='senior-list'),
    path('supervisor/users/<int:pk>/blood-pressure',
        SeniorsListBloodPressure.as_view(), name='senior-bp-list'),
    path('supervisor/users/<int:pk>/weight',
        SeniorsListWeight.as_view(), name='senior-weight-list'),
    path('supervisor/users/<int:pk>/sugar',
        SeniorsListSugar.as_view(), name='senior-sugar-list')
]
```

Fragment kodu 15. Kod definiujący część dostępnych adresów URL.

6.5 Interfejs graficzny

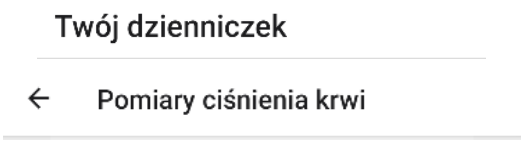
Interfejs graficzny został zaimplementowany zgodnie z zasadami określonymi w rozdziale 2.2. Zadbano o jego przejrzystość, kompatybilność z każdym rozmiarem czcionki systemowej, intuicyjność oraz o odpowiednią kolorystykę. Wszystkie ekrany oraz ich elementy zostały zaprogramowane jako komponenty funkcyjne. Aby poprawić odbiór aplikacji przez osoby starsze, zaprojektowano miłą dla oka ikonę aplikacji. (Rysunek 15)



Rysunek 15. Ikona aplikacji Dzienniczek Seniora [43]

6.5.1 Nawigacja

Użytkownik jest przekierowywany pomiędzy poszczególnymi ekranami aplikacji po wybraniu odpowiednich przycisków. Na każdym ekranie, wyświetlany jest *topbar* z nazwą widoku oraz w większości przypadków, przycisk z ikoną strzałki w lewo, umożliwiający powrót do poprzedniego widoku. (Rysunek 16) Użytkownicy mogą również cofać się przy użyciu dostępnego w urządzeniach Android guzika powrotu.



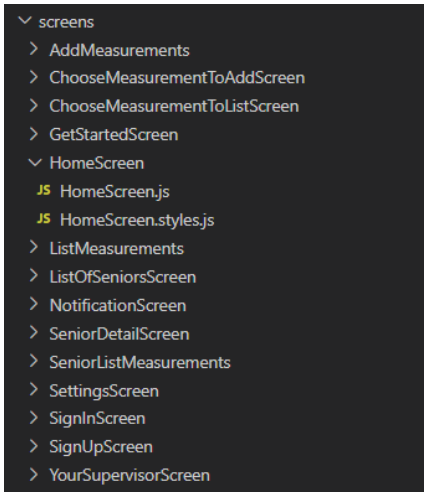
Rysunek 16. Zrzut fragmentu ekranu przedstawiający topbar bez opcji cofania oraz z taką opcją.

Nawigację zaimplementowano przy użyciu biblioteki React Navigation, konkretnie wykorzystano nawigację stosową (*stack navigation*). Deklaracja wszystkich dostępnych widoków – odpowiadającym im komponentom, ich tytuły oraz nazwy ścieżek - została zamieszczona w pliku `App.js`. (Fragment kodu 16)

```
export default function App() {
  ScheduledPushNotification();
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Home" gestureEnabled={false}>
        <Stack.Screen name="GetStarted" component={GetStartedScreen} options={{ title: "Twój dzienniczek", headerLeft: () => <></>}}  />
        <Stack.Screen name="SignIn" component={SignInScreen} options={{ title: "Logowanie" }} />
        <Stack.Screen name="SignUp" component={SignUpScreen} options={{ title: "Rejestracja" }} />
        <Stack.Screen name="Home" component={HomeScreen} options={{ title: "Twój dzienniczek", headerLeft: () => <></> }} />
        <Stack.Screen name="AddBloodPressure" component={AddBloodPressureScreen} options={{ title: "Dodaj pomiar ciśnienia krwi" }} />
        <Stack.Screen name="AddSugarLevel" component={AddSugarLevelScreen} options={{ title: "Dodaj pomiar cukru we krwi" }} />
        <Stack.Screen name="AddWeight" component={AddWeightScreen} options={{ title: "Dodaj swoją wagę" }} />
        <Stack.Screen name="ChooseMeasurementToAdd" component={ChooseMeasurementToAddScreen} options={{ title: "Wybierz pomiar" }} />
        <Stack.Screen name="ListBloodPressure" component={ListBloodPressureScreen} options={{ title: "Pomiary ciśnienia krwi" }} />
        <Stack.Screen name="ListSugarLevel" component={ListSugarLevelScreen} options={{ title: "Pomiary cukru" }} />
        <Stack.Screen name="ListWeight" component={ListWeightScreen} options={{ title: "Pomiary wagi" }} />
        <Stack.Screen name="ChooseMeasurementToList" component={ChooseMeasurementToListScreen} options={{ title: "Wybierz pomiar" }} />
        <Stack.Screen name="ListOfSeniors" component={ListOfSeniorsScreen} options={{ title: "Lista podopiecznych" }} />
        <Stack.Screen name="SeniorDetail" component={SeniorDetailScreen} options={{ title: "Szczegóły seniora" }} />
        <Stack.Screen name="SeniorBloodPressureList" component={SeniorListBloodPressureScreen} options={{ title: "Lista ciśnienia podopiecznego" }} />
        <Stack.Screen name="SeniorSugarList" component={SeniorListSugarScreen} options={{ title: "Lista cukru podopiecznego" }} />
        <Stack.Screen name="SeniorWeightList" component={SeniorListWeightScreen} options={{ title: "Lista wagi podopiecznego" }} />
        <Stack.Screen name="Notification" component={NotificationScreen} options={{ title: "Przypomnienie" }} />
        <Stack.Screen name="YourSupervisor" component={YourSupervisorScreen} options={{ title: "Twój opiekun" }} />
        <Stack.Screen name="Settings" component={SettingsScreen} options={{ title: "Ustawienia" }} />
      </Stack.Navigator>
      <Toast config={toastConfig}/>
    </NavigationContainer>
  );
}
```

Fragment kodu 16. Kod definiujący wszystkie dostępne dla użytkownika widoki. Fragment pliku App.js.

Kod źródłowy wszystkich dostępnych ekranów znajduje w katalogu `src/screens` (Rysunek 17), zachowując wcześniej opisaną konwencję struktury i nazewnictwa plików.



Rysunek 17 Struktura katalogu screens oraz przykładowego podkatalogu z widokiem ekranu.

6.5.2 Panel logowania i rejestracji

Po pierwszym uruchomieniu aplikacji wyświetlany jest ekran startowy zdefiniowany w pliku `GetStartedScreen.js`. Znajdują się na nim dwa przyciski prowadzące do panelu logowania (plik `SignIn.js`) oraz rejestracji (plik `SignUp.js`). (Rysunek 18)

Na stronie logowania użytkownik może wprowadzić email oraz hasło do wcześniej utworzonego konta, w celu uzyskania dostępu do swojego profilu. Pola tekstowe zostały zdefiniowane jako osobny komponent, aby wszystkie pola do wprowadzania wartości wyglądały jednakowo. (Fragment kodu 17) Po wybraniu opcji „Zaloguj się”, wywoływana jest metoda *Login* zdefiniowana w pliku `services.js`, która wysyła zapytanie do serwera przy użyciu biblioteki axios. (Fragment kodu 12)

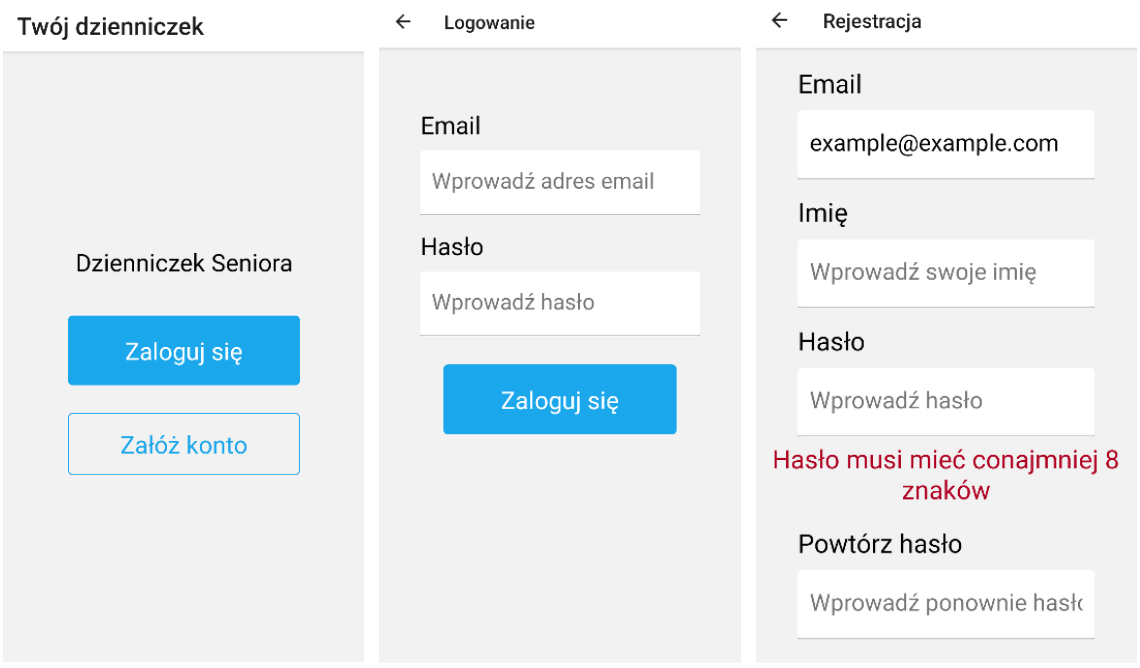
```
export const CustomTextInput = ({
  label,
  placeholder,
  value,
  setValue,
  secureTextEntry = false,
}) => {
  return (
    <View style={styles.root}>
      <Text style={styles.label}>{label}</Text>
      <TextInput
        placeholder={placeholder}
        value={value}
        onChangeText={(value) => setValue(value)}
        style={styles.input}
        secureTextEntry={secureTextEntry}
      />
    </View>
  )
}
```

Fragment kodu 17 Kod źródłowy komponentu TextIntput.

Panel rejestracji pozwala użytkownikowi na założenie konta. Pola z hasłami są sprawdzane już w trakcie wprowadzania wartości – czy długość hasła jest wystarczająca oraz czy hasła wprowadzone są jednakowe. Po wybraniu przycisku „Zarejestruj się”, wywoływana jest funkcja *Register* w pliku `services.js`, która wysyła dane do serwera.

```
export const Register = ({ payload, navigation }) => {
    axios
    .post(REGISTER_URL, payload)
    .then(response => {
      navigation.navigate('SignIn')
      successToast('Utworzono konto, zaloguj się.')
    })
    .catch(error => {
      console.log(error);
      if (error.response.status == 406) {
        errorToast(error.response.data)
      } else {
        if (!payload.email) {
          errorToast('Podaj adres email.')
        } else if (!payload.name) {
          errorToast('Podaj imię.')
        } else if (!payload.password) {
          errorToast('Podaj hasło.')
        } else {
          errorToast('Wystąpił błąd, spróbuj ponownie.')
        }
      }
    });
};
```

Fragment kodu 18 Kod definiujący metodę rejestrującą nowego użytkownika.



Rysunek 18 Zrzuty ekranu aplikacji. Od lewej: panelu startowego, panelu logowania i panelu rejestracji.

Wszystkie zamieszczone w pracy zrzuty ekranu aplikacji zostały wykonane z ustawioną największą czcionką systemową. Wygląd aplikacji dla tekstu regularnej wielkości może się nieznacznie różnić.

Zarówno strona logowania, jak i rejestracji, posiadają funkcje interpretujące odpowiedź serwera. Większość komunikatów w aplikacji jest wyświetlana w formie powiadomień *toast*. W projekcie zdefiniowano i wykorzystano ich dwa rodzaje *errorToast* świadczące o błędzie zwróconym przez backend oraz *successToast* informujące o poprawnym przetworzeniu żądania. (Fragment kodu 19, Rysunek 19) Ponadto, w większości przypadków, po uzyskaniu odpowiedzi ze statusem 200, użytkownik zostaje dodatkowo przekierowany do kolejnego widoku aplikacji.

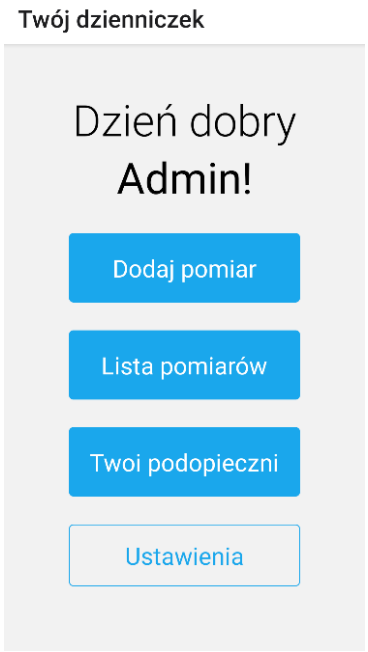
Rysunek 19 Zrzut ekranu przedstawiający powiadomienia toast.

```
    successToast('Pomyślnie zalogowano.')
    navigation.navigate('Home')
  })
  .catch(error => {
    if (!payload.email) {
      errorToast('Podaj adres email.')
    } else if (!payload.password) {
      errorToast('Podaj hasło.')
    } else {
      errorToast('Błędny email lub hasło.')
    }
  });
```

Fragment kodu 19. Kod przedstawiający sposób wywołania powiadomień toast.

6.5.3 Panel główny

Po pomyślnym zalogowaniu, użytkownik zostaje przekierowany do panelu głównego aplikacji. Na tym ekranie wyświetlane jest spersonalizowane powitanie (ze względu na nie w bazie danych została dodana kolumna z imieniem) oraz menu z operacjami, które mogą wykonywać jedynie zalogowani użytkownicy. (Rysunek 20) Aby ograniczyć ilość opcji menu głównego, postanowiono, że pozycja „Twoi podopieczni” będzie wyświetlana jedynie w przypadku, gdy zalogowany użytkownik jest czyimś opiekunem.



Rysunek 20 Zrzut ekranu panelu głównego.

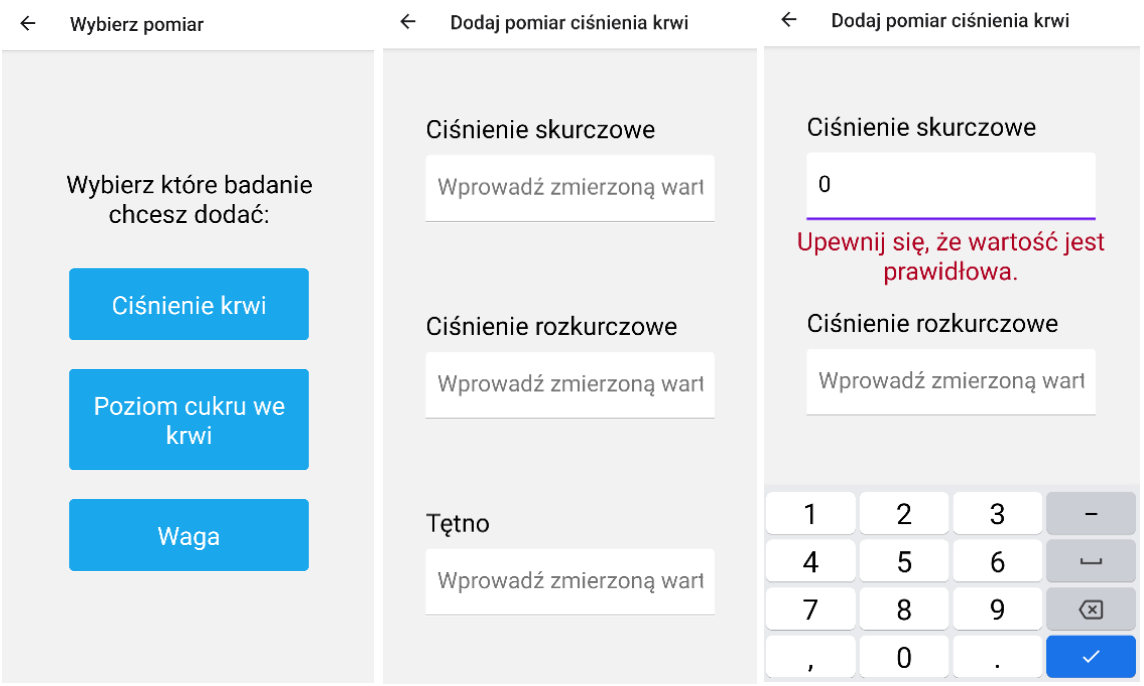
Do pobierania niezbędnych dla widoku panelu głównego, danych zaimplementowano punkt końcowy `api/home-page`, zwracający imię oraz wartość boolean określającą czy dany użytkownik jest opiekunem. (Fragment kodu 20Fragment kodu 20. Kod definiujący widok zwracający dane dla adresu api/home-page.)

```
class CurrentData(APIView):
    permission_classes = [MeasurementPermission]

    def get(self, request, format=None):
        user = CustomUser.objects.get(pk=get_user_id(self.request))
        is_supervisor = len(CustomUser.objects.filter(supervisor_id=user)) > 0
        return Response({'name': user.name, 'supervisor': is_supervisor})
```

Fragment kodu 20. Kod definiujący widok zwracający dane dla adresu api/home-page.

6.5.4 Dodawanie wyników

Jedną z najważniejszych funkcjonalności Dzienniczka Seniora jest możliwość zapisywania danych pomiarowym. Po wyborze opcji „Dodaj pomiar” z menu głównego, użytkownik zostaje przekierowany na stronę z wyborem badania, które chce dodać, zaś następnie – na stronę z formularzem wybranego pomiaru.



Rysunek 21. Zrzut ekranu (od lewej) panelu wyboru badania, formularza wyników pomiaru ciśnienia krwi oraz widok formularza z kontrolą błędów

Formularze dla wszystkich badań wyglądają niemal tak samo, gdyż są budowane z takich samych komponentów. Warto zwrócić uwagę, na zaimplementowaną walidację – na wszystkie pola w widokach dodawania pomiarów nałożono pewne ograniczenia, które w razie ich przekroczenia od razu informują użytkownika. (Rysunek 21, pierwszy widok po prawej) Wspomniane przedziały zostały zdefiniowane w pliku `constsForMeasurements.js` w katalogu `config`. Informacje o błędach w formularzu są wyświetlane przy użyciu komponentu *HelperText* z biblioteki React Native Paper. Ponadto, aby ułatwić wprowadzanie danych pomiarowych oraz jednocześnie uniknąć błędów, pola formularza zdefiniowano jakie komponenty *NumericInput* czyli szczególny rodzaj *TextInput* z określonym typem klawiatury wyświetlanym przy wyborze pola – w tym przypadku klawiatury numerycznej.

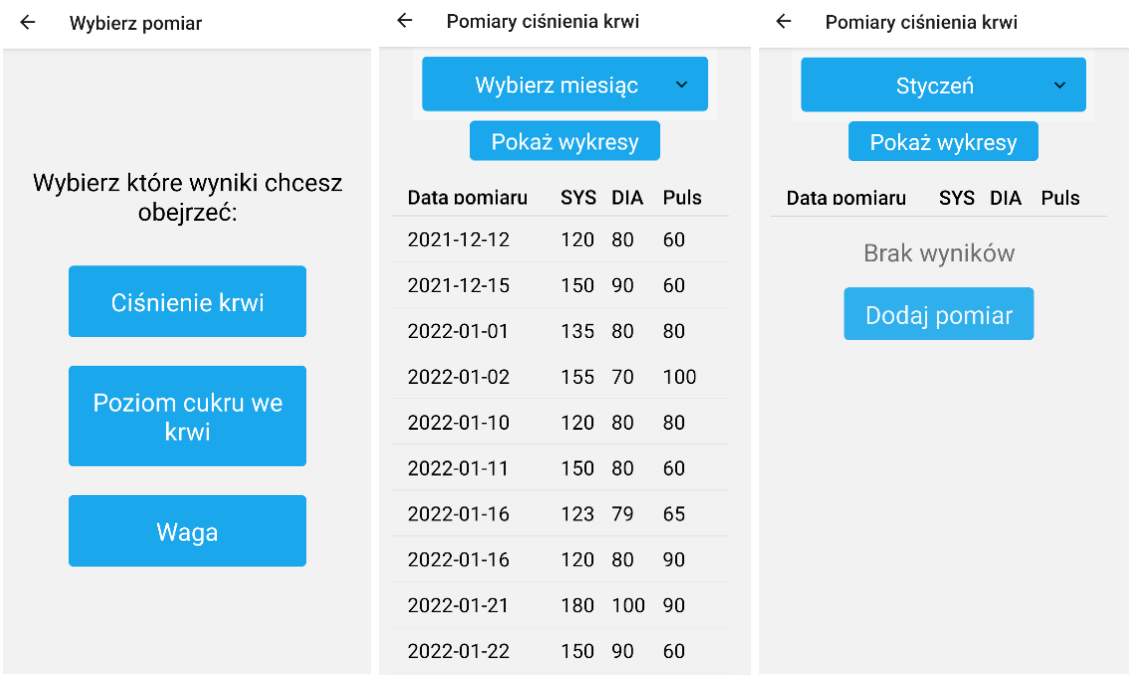
```
// funkcja sprawdzająca czy wpisana wartość mieści się
// w przedziale (MIN_SUGAR; MAX_SUGAR)
const sugarCheck = () => {
  return (level < MIN_SUGAR || level > MAX_SUGAR) && level !== ''
}

return (
  <ScrollView>
    <View style={styles.root}>
      <NumericInput
        label="Poziom cukru"
        placeholder="Wprowadź zmierzoną wartość"
        value={level}
        setValue={setLevel}
      />
      <HelperText type="error" visible={sugarCheck()} style={styles.helper}>
        Podaj prawidłowy poziom cukru
      </HelperText>
      <CustomButton
        label="Zapisz wyniki pomiaru"
        onPress={() => onAdd(level)}
      />
    </View>
  </ScrollView>
)
```

Fragment kodu 21. Kod definiujący metodę sprawdzającą poprawność wprowadzonej przez użytkownika wartości oraz część kodu generująca widok formularza dla pomiaru glukozy.

Wprowadzone wyniki zostają przesłane do serwera po naciśnięciu przycisku „Zapisz wyniki pomiaru" poprzez wysłanie zapytania POST pod adres `api/measurements/nazwa_pomiaru`. Operacja ta, tak jak pozostałe funkcje znajdujące się w pliku `services.js`, jest wyposażona w obsługę błędów.

6.5.5 Prezentacja wyników

Kolejną opcją, którą może wybrać użytkownik z menu panelu głównego jest lista pomiarów. Podobnie jak w przypadku dodawania wyników, najpierw należy wybrać rodzaj badania, którego wyniki użytkownik chce poddać obejrzeć. Następnie wyświetlany jest panel wybranych wyników składa się on z przycisku służącego do wyboru miesiące, z którego pomiary mają zostać wyświetlone – domyślnie w panelu wyświetlają się wszystkie dostępne w bazie wyniki zapisane przez aktualnie zalogowanego użytkownika. Pod wspomnianym przyciskiem znajduje się guzik pokazujący wykresy, postanowiono, aby, w stanie początkowym, wykresy były schowane, gdyż użytkownicy mogliby wtedy nie zauważyć tabeli mieszczącej się pod nimi. W szczególnym przypadku, w którym użytkownik wybierze obecny miesiąc i nie dodał w nim jeszcze żadnych wyników, wyświetlany jest przycisk dodaj pomiar, który przekierowuje do strony dodawania danego badania. Dla każdego typu pomiarów zdefiniowano komponenty generujące tabele, dane do nich przekazane są odpowiednio mapowane na kolejne wiersze. (Fragment kodu 22)



Rysunek 22. Zrzut ekranu (od lewej) panelu wyboru badania, którego wyniki mają być wyświetlane, lista wyników wszystkich pomiarów ciśnienia krwi oraz widoku listy, gdy użytkownik nie posiada danych w bazie

Dane wyświetlane w tabeli są pobierane z serwera poprzez wysłanie żądania GET pod adres `api/measurements/nazwa_pomiaru` (`blood_pressure`, `sugar` lub `weight`). Backend w odpowiedzi przesyła jedynie wyniki, które w polu *user_id* mają taki sam numer id jak zakodowana w tokenie wartość. Przykładowy kod został już przytoczony w jednym z poprzednich rozdziałów. (Fragment kodu 4)

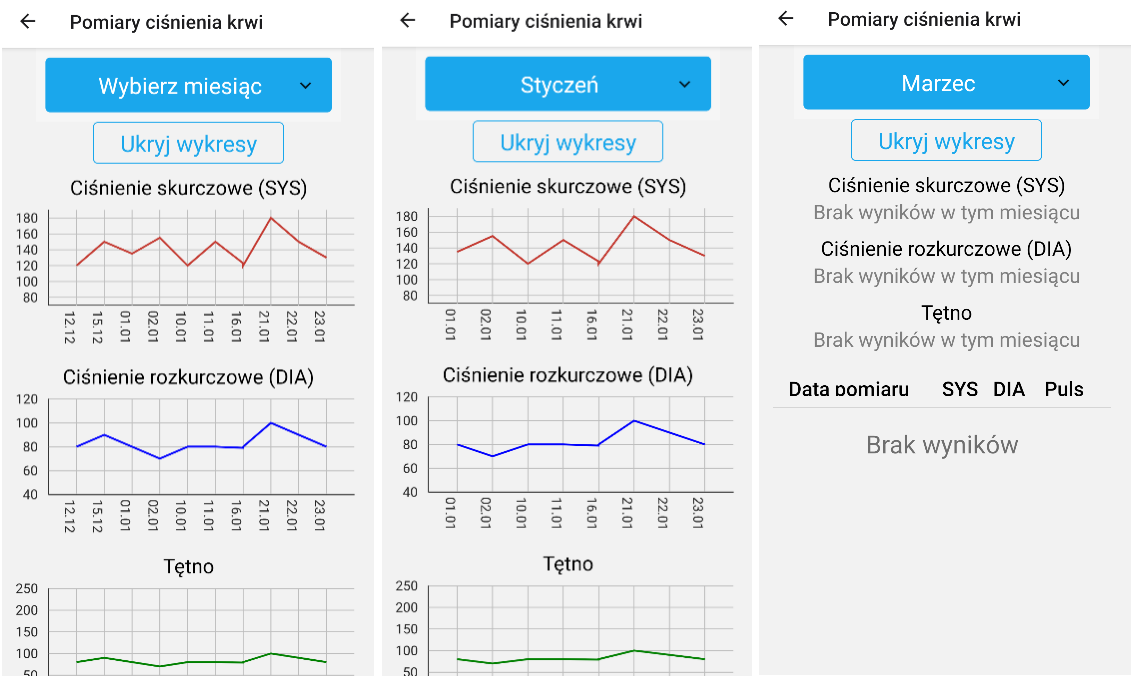
```
const BloodPressureTable = () => {
  return data.map((measurement, key) => {
    if (
      month === 0 ||
      Number(measurement.date.toString().slice(5, 7)) === month
    ) {
      return (
        <DataTable.Row key={key}>
          <DataTable.Cell style={{ flex: 3 }}>
            <Text style={styles.cell}>{measurement.date}</Text>
          </DataTable.Cell>
          <DataTable.Cell>
            <Text style={styles.cell}>{measurement.systolic}</Text>
          </DataTable.Cell>
          <DataTable.Cell>
            <Text style={styles.cell}>{measurement.diastolic}</Text>
          </DataTable.Cell>
          <DataTable.Cell>
            <Text style={styles.cell}>{measurement.pulse}</Text>
          </DataTable.Cell>
        </DataTable.Row>
      )
```

Fragment kodu 22 Kod mapujący przekazane do funkcji dane na wiersze tabeli, fragment kodu źródłowego BloodPressureTableComponent

Oprócz wspomnianego powyżej widoku prezentującego wyniki w formie tabeli, użytkownik ma również możliwość wyświetlenia ich na wykresach. Dla każdego rodzaju pomiarów zdefiniowano komponenty, w których dane są odpowiednio przetwarzane, aby umożliwić ich wizualizację z użyciem biblioteki Victory Native. Wartości przedstawiane na wykresach muszą być przekazane w formie tabeli obiektów o strukturze {x, y}. Taki format danych uzyskano wykorzystując ponownie funkcję mapującą – w aplikacji Dzieniczek Seniora, wszystkie wykresy na osi X przedstawiają datę pomiaru, zaś na osi Y jego wynik, stąd funkcja mapująca została zaimplementowana w sposób przedstawiony poniżej. (Fragment kodu 23) Należało również zdefiniować zmienną (*formattedData*) przechowującą przefiltrowane dane w przypadku, gdy użytkownik określi miesiąc, z którego wyniki chce obejrzeć. W przypadku nieotrzymania żadnych danych z serwera, wyświetlana jest informacja o braku wyników w bazie. (Rysunek 23)

```
const dataAll = data?.map((measurement) => {
  return { x: measurement.date, y: Number(measurement.level) }
})
const formattedData =
  month !== 0
    ? dataAll.filter(
        (coords) => Number(coords.x.toString().slice(5, 7)) === month
      )
    : dataAll
```

Fragment kodu 23. Definicja formatowania zmiennej przechowującej wszystkie dane sformatowane w sposób wymagany przez bibliotekę Victory Native.



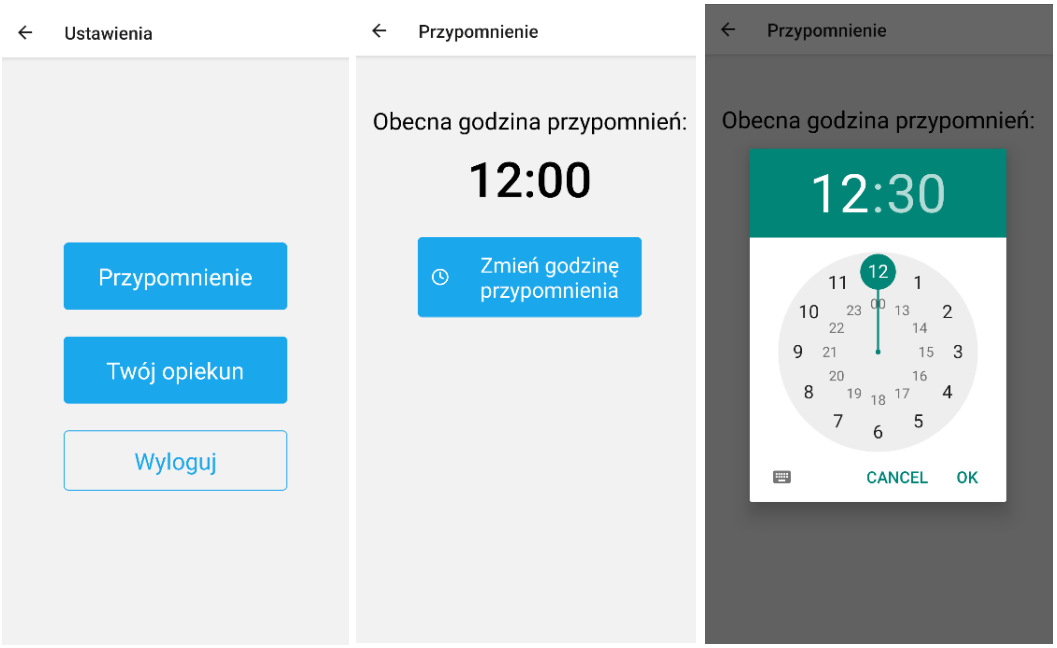
Rysunek 23 Zrzut ekranu widoku wykresów przedstawiających wyniki dla (od lewej) wszystkich wyników, wyników pomiarów przeprowadzonych w styczniu oraz dla miesiąca bez danych.

Warto również zaznaczyć, że widok przedstawiający wyniki wszystkich pomiarów jest możliwy do przewijania.

6.5.6 Ustawienia konta

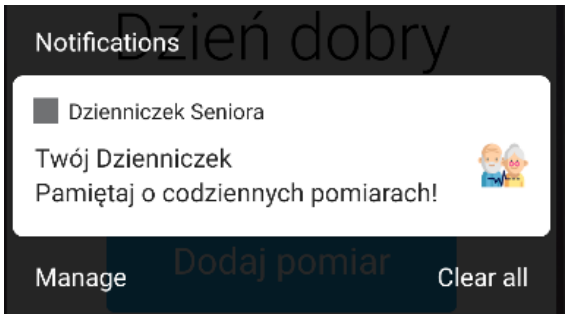
Ostatnią opcją z menu głównego, która jest dostępna dla wszystkich użytkowników jest panel ustawienia – umieszczono w nim opcje, które nie są wykorzystywane na co dzień i są przypisane bezpośrednio do konta użytkownika. Operacjami tymi jest możliwość ustalenia terminu przypomnień o pomiarach, wysyłanych w formacie powiadomień push, możliwość wybrania swojego opiekuna oraz przycisk „Wyloguj", który wywołuje metodę *Logout* z pliku `services.js`. (Rysunek 24) Przesyła ona zapytanie POST do serwera z tokenem odświeżenia w ciele, zaś następnie wykonywane są operację opisane w rozdziale 6.4.2 (Fragment kodu 11) oraz usuwa wszystkie dane użytkownika przechowywane w magazynie lokalnym.

Widok „Przypomnienie" pozwala użytkownikowi na ustalenie dokładnej godziny otrzymywania powiadomień push. Na górze ekranu wyświetlana jest obecnie ustawiona godzina przypomnień, zaś po dodaniu nowej jest ona, oczywiście, aktualizowana. Poniżej znajduje się przycisk z ikoną zegara, zważywszy na fakt, że część seniorów może mieć problem z interpretacją samej grafiki, o czym wspomniano w rozdziale 2.2, dołączono również podpis wykonywanej przez przycisk operacji. Po jego wybraniu, użytkownikowi wyświetlany jest *DateTimePicker* z biblioteki o tej samej nazwie. Po wybraniu godziny i wciśnięciu OK, na adres `api/user/add-schedule`, jest wysyłane zapytanie PATCH z wybraną godziną w ciele.



Rysunek 24 Zrzut ekranu widoków (od lewej) panelu ustawień konta użytkownika, widok ustawiania godziny przypomnienia oraz podgląd sposobu wprowadzania godziny.

Powiadomienia push wysyłane są o wyznaczonej przez użytkownika godzinie każdego dnia. Przy uruchomianiu aplikacji oraz przy aktualizacji godziny powiadomień, wywoływana jest funkcja *ScheduledPushNotification*, która pobiera z magazynu lokalnego aktualną godzinę przypomnień i ustawia ją jako termin powiadomienia. Do realizacji tej funkcji wykorzystano bibliotekę React Native Push Notification.

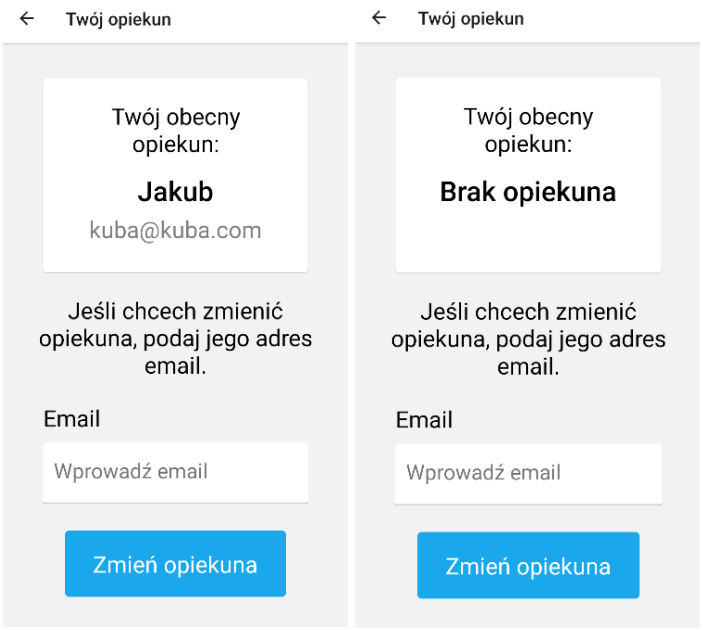


Rysunek 25 Zrzut ekranu powiadomienia push wysyłanego przez aplikację.

```
PushNotification.localNotificationSchedule({
  channelId: 'main-channel',
  title: 'Twój Dzienniczek',
  message: 'Pamiętaj o codziennych pomiarach!',
  date: new Date(
    today.getFullYear(),
    today.getMonth(),
    today.getDate(),
    hour,
    minutes
  ),
  allowWhileIdle: true,
  id: 1,
})
```

Fragment kodu 24 Definicja szczegółów wysyłanego użytkownikowi powiadomienia push.

Drugim widokiem dostępnych w sekcji ustawienia jest panel „Twój opiekun", wyświetlane są tam informacje na temat wybranego wcześniej opiekuna – jego imię oraz adres email – lub w przypadku jego braku, takowa informacja. (Rysunek 26) Dane na temat opiekuna są pobierane z adresu `api/users/your-supervisor`. Użytkownik może udostępnić dostęp do swoich wyników jedynie jednej osobie, którą określa poprzez podanie jej adresu email w odpowiednim polu. Co ważne, osoba, która ma otrzymać dostęp do danych musi mieć aktywne konto w Dzienniczku Seniora. To, czy adres wprowadzony przez użytkownika jest poprawny, weryfikowane jest po stronie serwera – frontend przesyła zapytanie PATCH, które zawiera dane potencjalnego opiekuna. Backend sprawdza, czy taki użytkownik istnieje i jeśli tak, to zwraca jego dane, a jeśli nie – wysyła odpowiedź z takową informacją (Fragment kodu 25), która wyświetlana jest w formie komunikatu toast.



Rysunek 26 Zrzut ekranu "Twój opiekun" (od lewej) dla użytkownika z wyznaczonym opiekunem i bez niego

```python
class YourSupervisor(APIView):
    permission_classes = [AllowAny]

    def get(self, request, format=None):
        user = CustomUser.objects.get(pk=get_user_id(self.request))
        if user.supervisor_id:
            supervisor = CustomUser.objects.get(pk=user.supervisor_id.pk)
            serializer = CustomUserSerializer(supervisor, many=False)
            return Response(serializer.data)
        else:
            return Response({'name': 'Brak opiekuna', 'email': ''})

    def patch(self, request, format=None):
        user = CustomUser.objects.get(pk=get_user_id(self.request))
        try:
            supervisor = CustomUser.objects.get(
                email=request.data['supervisor_email'])
            data = {'supervisor_id': supervisor.pk}
        except CustomUser.DoesNotExist:
            return Response('Nie znaleziono opiekuna z podanym adresem email', status=status.HTTP_400_BAD_REQUEST)
        serializer = CustomUserSerializer(user, data=data, partial=True)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

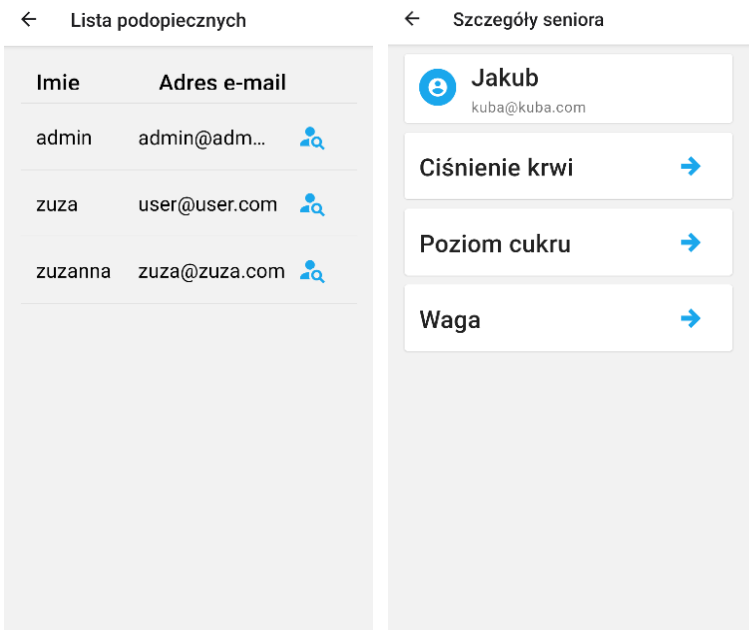
Fragment kodu 25 Kod definiujący widok YourSupervisor.

6.5.7 Panel opiekuna

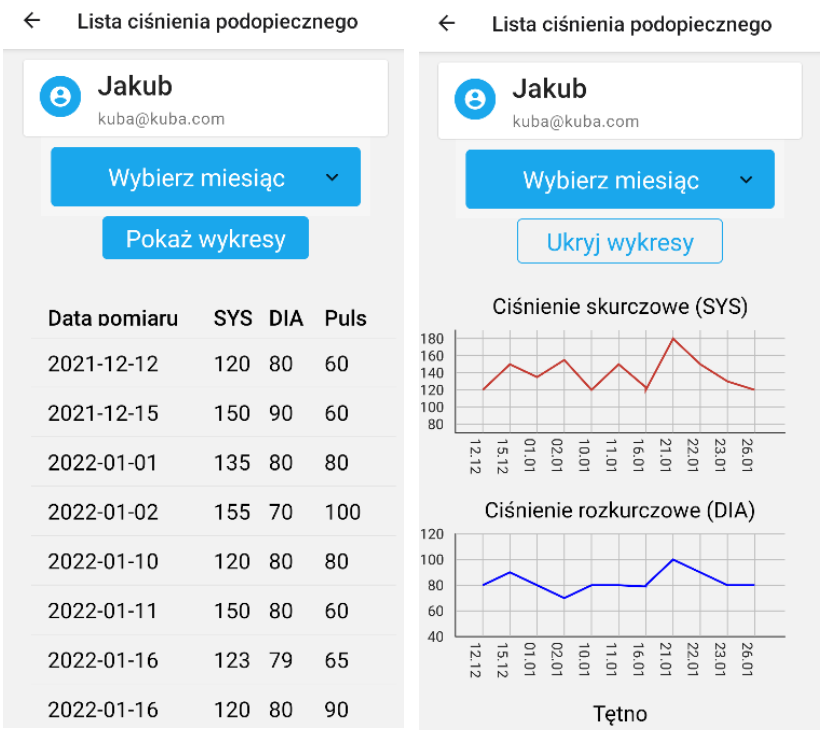
Użytkownicy, którzy zostali wybrani jako opiekunowie widzą w panelu głównym dodatkową opcję – „Twoi podopieczni” – po jej wybraniu są przekierowywani do listy wszystkich osób, do których danych mają dostęp. Lista użytkowników jest pobierana z adresu `api/supervisor/users/` metodą GET (Fragment kodu 6), a następnie uzyskane z serwera dane są następnie mapowane do postaci klikalnych wierszy tabeli, które prowadzą do widoku szczegółów danego podopiecznego.

Panel szczegółów seniora przedstawia imię i adres email wybranego użytkownika w formie karty oraz listę typów badań, których wyniki opiekun może obejrzeć – są to standardowo ciśnienie krwi, poziom glukozy oraz waga. Karta seniora została zaimplementowana jako komponent funkcyjny `ProfileCard.js`, gdyż element ten powtarza się w wielu widokach.

Wybierając konkretny typ badania opiekun zostaje przekierowany do strony z wynikami podopiecznego. Widok ten jest analogiczny do podglądu wyników przypisanych do konta aktualnie zalogowanego użytkownika, jedyne co odróżnia ten panel to wspomniana wcześniej karta ze szczegółami seniora na górze ekranu. Nie jest również wyświetlany przycisk „Dodaj pomiar" w przypadku, gdy wybrano filtr bieżącego miesiąca. Dane są pobierane z serwera poprzez zapytanie GET pod adres `api/supervisor/users/id/nazwa_badania`, gdzie id to identyfikator seniora, którego dane użytkownik chce uzyskać. (Fragment kodu 7)



Rysunek 27 Zrzut ekranu (od lewej) listy podopiecznych zalogowanego użytkownika i panelu szczegółów wybranego seniora



Rysunek 28 Zrzut ekranu (od lewej) podglądu wyników pomiarów ciśnienia krwi podopiecznego w formie tabeli i podgląd wyników w formie wykresów.

7 Podsumowanie

Aplikacja została stworzona z myślą o osobach starszych, które nie czują się pewnie korzystając z rozwiązań mobilnych. Celem pracy było zaprojektowanie rozwiązania przystępnego i, przede wszystkim, w pełni zrozumiałego dla seniorów.

Pomysł na stworzenie Dzienniczka Seniora zrodził się podczas wizyty u babci, która, jak na osobę w podeszłym wieku, naprawdę dobrze radzi sobie z nowoczesnymi urządzeniami. Często jednak zdarza się, że zwraca się do młodszego pokolenia z prośbą o pomoc – wtedy tworzy liczne notatki, zapisując instrukcje krok po kroku, jak rozwiązać napotkany problem. Babcia każdego dnia musi monitorować poziom glukozy oraz ciśnienie krwi – dokładnie te parametry, które może zanotować w Dzienniczku Seniora. Projektując aplikację zwracano uwagę na to, aby do jej obsługi użytkownicy nie potrzebowali żadnych instrukcji – aby interfejs był na tyle intuicyjny.

Osoby starsze mają często tendencję do przekazywania, nie do końca zgodnych z prawdą, informacji na temat swojego zdrowia i samopoczucia, tłumacząc takie zachowanie tym, że nie chcą zamartwiać bliskich. Aplikacja będąca przedmiotem tej pracy, pozwala na udostępnianie danych np. jednemu z członków rodziny, aby ten miał dostęp do rzetelnych danych na temat zdrowia swojej bliskiej osoby. Nieprawidłowości, widoczne w wynikach pomiarów, bardzo często okazują się później przyczyną złego samopoczucia. Zakładając, że dziecko lub wnuk osoby starszej ma dostęp do jej danych pomiarowych, będzie wiedział, kiedy istnieje ryzyko, że babcia czy dziadek może się nie najlepiej czuć.

Podczas realizacji pracy dyplomowej stworzyłam pierwszą w życiu aplikację mobilną – poznałam framework do tworzenia programów dla systemu Android - React Native. Udoskonaliłam także swoją znajomość platformy Django i świetnie uzupełniającej jej biblioteki Django REST Framework. W trakcie tworzenia Dzienniczka Seniora starałam się również zwracać uwagę na to, czy pisany przeze mnie kod jest w przejrzysty i będzie w pełni zrozumiały dla innych programistów.

Gdyby projekt miał być realizowany od nowa, na pewno, należałoby lepiej zadbać o jego płynność działania – obecnie, miewa opóźnienia (np. przy aktualizowaniu wartości), co może negatywnie wpływać odbiór aplikacji. Warto byłoby również przeprowadzić wywiady oraz testy użyteczności z osobami starszymi, żeby upewnić się, czy rzeczywiście interfejs jest dla nich zrozumiały. Kolejną rzeczą, która zostałaby zmieniona jest DateTimePicker - nie jest wizualnie spójny z pozostałymi widokami oraz występują w nim przyciski z tytułami w języku angielskim. Jeżeli chodzi o część serwerową, zostałaby zmieniona biblioteka wykorzystywana do uwierzytelniania – obecna wymaga przechowywania tokenów, których ważność się skończyła (tabela *Blacklisted tokens*), co zajmuje niepotrzebnie pamięć.

Perspektywy rozwoju aplikacji są bardzo szerokie, ponieważ obecnie aplikacja posiada niewielką ilość funkcjonalności w porównaniu do dostępnych na rynku rozwiązań takich jak MyTherapy opisane w rozdziale 3.1. Zdecydowana większość osób w podeszłym wieku musi każdego dnia przyjmować leki oraz suplementy, dlatego warto by rozważyć dodanie możliwości ustawiania o nich przypomnień. Aplikacja mogłaby również umożliwić generowanie i przesyłanie okresowych raportów podsumowujących stan zdrowia użytkowników. Podsumowania mogłyby być dostępne do pobrania dla użytkownika, opiekuna, a w przyszłości może nawet dla lekarza, jeśliby dodano nowy typ dostępu jakim byłby medyk. W chwili obecnej, użytkownik może określić tylko jednego opiekuna, w przyszłości należałoby się zastanowić, czy jednak senior nie powinien mieć możliwości przypisania sobie większej ilości opiekunów. Warto by również zadbać o ulepszenie warstwy wizualnej projektu, zaimplementować interfejs nadal tak samo intuicyjny i przejrzysty, ale jednocześnie bardziej przyjemny dla oka. Nadanie aplikacji nowocześniejszego designu pozytywnie wpłynęłoby na jej odbiór także w młodszym gronie – chętniej zainstalowaliby oni lepiej wyglądającą aplikację. Przy wprowadzaniu zmian należy pamiętać dla kogo pierwotnie powstała aplikacja, tak aby nie zaniedbać oryginalnej grupy docelowej.

8 Bibliografia

1. Y. Kamiya, N. Mun Sim Lai, K. Schmid, *World Population Ageing 2020 Highlights* [online] s.l., w.n. [dostęp: 09.06.2021] Dostępny w Internecie: https://www.un.org/development/desa/pd/sites/www.un.org.development.desa.pd/files/undesa_pd-2020_world_population_ageing_highlights.pdf
2. W.Kitajewska, W.Szeląg, Z.Kopański, Z.Maslyak, I.Sklyarov, *Choroby cywilizacyjne i ich prewencja* [online], s.l., Journal of Clinical Healthcare, Kraków, 2014 [dostęp: 09.06.2021] Dostępny w Internecie: http://jchc.eu/numery/2014_1/201411.pdf
3. A. Tykarski, K.J. Filipiak, A. Januszewicz, M. Litwin, K. Narkiewicz, A. Prejbisz, D. Ostalska-Nowicka, K. Widecka, K. Kostka-Jeziorny, *Zasady postępowania w nadciśnieniu tętniczym — 2019 rok, Wytyczne Polskiego Towarzystwa Nadciśnienia Tętniczego* [online], s.l., w.n, Nadciśnienie Tętnicze w Praktyce 2019, tom 5, nr 1, strony: 1–86 [dostęp: 09.06.2021] Dostępny w Internecie: https://mlodzilekarzerodzinni.pl/wp-content/uploads/2020/01/Wytyczne-PTNT-2019.pdf
4. *Internetowa encyklopedia PWN,* Wydawnictwo Naukowe PWN, Hasło: miażdżyca, [dostęp: 09.06.2021] Dostępny w Internecie: https://encyklopedia.pwn.pl/haslo/miazdzyca;3940414.html
5. W. Fabian, D. Zozulińska-Ziółkiewicz (red.*), Kolegium Lekarzy Rodzinnych w Polsce i Polskie Towarzystwo Diabetologiczne, Zasady postępowania w cukrzycy – zalecenia dla lekarzy POZ (2019), Wytyczne Kolegium Lekarzy Rodzinnych w Polsce i Polskiego Towarzystwa Diabetologicznego zalecane przez konsultantów krajowych w dziedzinie medycyny rodzinnej i w dziedzinie diabetologii* [online], s.l., Medycyna Praktycza, 2019 [dostęp: 09.06.2021] Dostępny do pobrania w Internecie: https://www.klrwp.pl/strona/624/zasady-postepowania-w-cukrzycy-2019/pl
6. *Światowy raport na temat cukrzycy przedstawia sytuację i działania podejmowane w celu walki z tą chorobą w Polsce* [online], s.l., World Health Organization REgional Office for Europe, 2016 [dostęp: 09.06.2021] Dostępny do pobrania w Internecie: https://www.gov.pl/attachment/e92c747d-695e-467c-ba45-61b17820e809
7. *IDF Diabetes Atlas 9th edition 2019,* [online] s.l., International Diabetes Federation, 2019 [dostęp: 09.06.2021] ISBN: 978-2-930229-87-4 Dostępny do pobrania w Internecie: https://diabetesatlas.org/atlas/ninth-edition/
8. A. Prelicz-Zawadzka, L. Zawadzki, *User Centered Design Canvas, New UX tool combining user needs with business goals* [online] s.l., The Rectangles [dostęp: 09.06.2021] Dostępny w Internecie: https://ucdc.therectangles.com/
9. A.Correia de Barrosa, R.Leitãob, J.Ribeiroa, *Design and evaluation of a mobile user interface for older adults: navigation, interaction and visual design recommendations, Procedia Computer Science 27 (2014) str. 369 – 378* [online], s.l., Elsevier, 2014 [dostęp: 09.06.2021] Dostępny w Internecie: https://www.sciencedirect.com/science/article/pii/S187705091400043X
10. M.Faisal, M.Yusof, N.Romli, *Design for Elderly Friendly: Mobile Phone Application and Design that Suitable for Elderly, International Journal of Computer Applications – No.3,* [online], s.l., w.n., czerweiec 2014 [dostęp: 09.06.2021] ISBN: 978-0-12-804467-4 Dostępny w Internecie:

https://www.researchgate.net/publication/314829622_Design_for_Elderly_Friendly_Mobile_Phone_Application_and_Design_that_Suitable_for_Elderly
11. J.Johnson, K.Finn, *Designing User Interfaces for an Aging Population, Towards Universal Design* [online], s.l., Elsevier, 2017 [dostęp: 09.06.2021] Dostępny w Internecie: https://books.google.pl/books?hl=pl&lr=&id=n5AxDQAAQBAJ&oi=fnd&pg=PP1&dq=J.Johnson,+K.Finn,+Designing+User+Interfaces+for+an+Aging+Population:+Towards+Universal+Design&ots=PcGfDTVpu_&sig=n5JGL2z-Jmclc5ceoviK6FLnjgI&redir_esc=y#v=onepage&q=J.Johnson%2C%20K.Finn%2C%20Designing%20User%20Interfaces%20for%20an%20Aging%20Population%3A%20Towards%20Universal%20Design&f=false
12. Zespół optometrystów KODANO Optyk, *Zaburzenia widzenia barw* [online], [dostęp: 09.06.2021] Dostępny w Internecie: https://kodano.pl/poradnik/zaburzenia-widzenia-barw.html
13. Little Angel, *Blood Pressure and Sugar Tracker* [online], [dostęp: 24.01.2022] Dostępny w Internecie: https://play.google.com/store/apps/details?id=com.angel.blood.pressure.sugar
14. smartpatient, *MyTherapy* [online], [dostęp: 24.01.2022] Dostępny w Internecie: https://play.google.com/store/apps/details?id=eu.smartpatient.mytherapy
15. Blip Software, *Ciśnienie Krwi (MyDiary)* [online], [dostęp: 24.01.2022] Dostępny w Internecie: https://play.google.com/store/apps/details?id=com.zlamanit.blood.pressure
16. Health & Fitness AI Lab, *Dziennik ciśnienia krwi* [online], [dostęp: 24.01.2022] Dostępny w Internecie: https://play.google.com/store/apps/details?id=com.bluefish.bloodpressure
17. L. Goncerz, K. Mazur, D. Mystek, *Open Source, definicja w Encyklopedii Zarządzania* [online], [dostęp: 24.01.2022] Dostępny w Internecie: https://mfiles.pl/pl/index.php/Open_Source
18. P. Tynecki, *O języku Python* [online], 2008, [dostęp: 24.01.2022] Dostępny w Internecie: https://pl.python.org/o,jezyku,python.html
19. *Oficjalna dokumentacja biblioteki Django* [online], [dostęp: 24.01.2022] Dostępny w Internecie: https://docs.djangoproject.com/
20. *Bezpieczeństwo w Django*, W: Oficjalna dokumentacja biblioteki Django [online], [dostęp: 24.01.2022] Dostępny w Internecie: https://docs.djangoproject.com/en/4.0/topics/security/
21. R.Kurjata, *Bezpieczeństwo Medycznych Systemów Informatycznych, Wykład 12*, *Bezpieczeństwo aplikacji internetowych (c.d.),* Warszawa, 31 maja 2021, slajd 9 [materiał udostępniony studentom]
22. R.Kurjata, *Bezpieczeństwo Medycznych Systemów Informatycznych, Wykład 12*, *Bezpieczeństwo aplikacji internetowych (c.d.),* Warszawa, 31 maja 2021, slajd 37, 46 [materiał udostępniony studentom]
23. S. Zawadzki, *Co to jest API? Wszystko o interfejsie programowania aplikacji*, W: Smartbees [online], [dostęp: 24.01.2022] Dostępny w Internecie: https://smartbees.pl/blog/co-jest-api-wszystko-o-interfejsie-programowania-aplikacji
24. *Oficjalna dokumentacja biblioteki React Native* [online], [dostęp: 24.01.2022] Dostępny w Internecie: https://reactnative.dev/

25. S. Liu, *Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021* [online] s.l., w.n., lipiec 2021, [dostęp: 24.01.2022] Dostępny w Internecie: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/
26. *Async Storage* [online], [dostęp: 25.01.2022] Dostępny w Internecie: https://react-native-async-storage.github.io/async-storage/docs/install
27. *React Native DateTimePicker* [online], [dostęp: 25.01.2022] Dostępny w Internecie: https://github.com/react-native-datetimepicker/datetimepicker
28. *Native Stack Navigator* [online], [dostęp: 25.01.2022] Dostępny w Internecie: https://reactnavigation.org/docs/native-stack-navigator/
29. *Getting Started, What is Axios?* [online], [dostęp: 25.01.2022] Dostępny w Internecie: https://axios-http.com/docs/intro
30. *Getting Started with Victory Native* [online], [dostęp: 25.01.2022] Dostępny w Internecie: https://formidable.com/open-source/victory/docs/native/
31. *Plik README.md, react-native-toast-message* [online], [dostęp: 25.01.2022] Dostępny w Internecie: https://github.com/calintamas/react-native-toast-message#readme
32. *React Native Push Notifications* [online], [dostęp: 25.01.2022] Dostępny w Internecie: https://github.com/zo0r/react-native-push-notification
33. *Słownik języka polskiego PWN*, Wydawnictwo Naukowe PWN, Hasło: emulacja, [dostęp: 25.01.2022] Dostępny w Internecie: https://sjp.pwn.pl/szukaj/emulacja.html
34. *Aplikacja Diagrams.net* [online], JGraph Ltd, [dostęp: 25.01.2022] Dostępny w Internecie: https://www.diagrams.net/
35. M. Jain, *Best Folder Structure for React Native Project* W: Minds Verse, Habilelabs [online], Habilelabs, 2021, [dostęp: 25.01.2022] Dostępny w Internecie: https://learn.habilelabs.io/best-folder-structure-for-react-native-project-a46405bdba7
36. *National Institute on Aging, High Blood Pressure and Older Adults* [online], [dostęp: 25.01.2022] Dostępny w Internecie: https://www.nia.nih.gov/health/high-blood-pressure-and-older-adults
37. Avram R. i in., *Real-world heart rate norms in the Health eHeart study* [online], s.l., Springer Nature, [dostęp: 25.01.2022] Dostępny w Internecie: https://www.nature.com/articles/s41746-019-0134-9
38. *Diabetes – Diagnosis & treatment* W: Mayo Clinic [online], [dostęp: 25.01.2022] Dostępny w Internecie: https://www.mayoclinic.org/diseases-conditions/diabetes/diagnosis-treatment/drc-20371451
39. Pietraszek M., *Protokół HTTP* W: Samouczek Programisty [online], 2018 [dostęp: 25.01.2022] Dostępny w Internecie: https://www.samouczekprogramisty.pl/protokol-http/
40. *Interfejsy API REST* W: IBM Cloud Learn Hub [online], 2021 [dostęp: 25.01.2022] Dostępny w Internecie: https://www.ibm.com/pl-pl/cloud/learn/rest-apis
41. Co to jest i jak działa JSON Web Tokens (JWT)? W: Global4net [online], 2019 [dostęp: 25.01.2022] Dostępny w Internecie: https://global4net.com/ecommerce/co-to-jest-i-jak-dziala-json-web-tokens-jwt/
42. *Oficjalna strona internetowa JSON Web Tokens* [online], [dostęp: 25.01.2022] Dostępny w Internecie: https://jwt.io/
43. Ikonografiki wykorzystane do stworzenia ikony aplikacji:

Grandparents premium icon [online], Freepik, [dostęp: 23.01.2022] Dostępny w Internecie: https://www.flaticon.com/premium-icon/grandparents_1662977
Heartbeat free icon [online], Freepik, [dostęp: 23.01.2022] Dostępny w Internecie: https://www.flaticon.com/free-icon/heartbeat_31704?term=medical%20signal&page=1&position=8&page=1&position=8&related_id=31704&origin=tag

9 Spis rysunków

10 Spis tabel

11 Spis załączników

1. Archiwum `dzienniczek-seniora.zip` przechowujące wszystkie pliki źródłowe aplikacji oraz autorskie pliki graficzne wykorzystane w niniejszej pracy.