

**AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE**

Wydział Inżynierii Metali i Informatyki Przemysłowej



PRACA DYPLOMOWA INŻYNIERSKA
pt.

„Opracowanie aplikacji mobilnej wspomagającej przyjmowanie leków przez osoby chore na stwardnienie rozsiane”

Imię i nazwisko dyplomanta:

Karol Skóra

Kierunek studiów:

Informatyka Stosowana

Nr albumu:

260468

Promotor:

dr hab. inż. Łukasz Rauch

Recenzent:

dr inż. Krzysztof Regulski

Podpis dyplomanta:

Podpis promotora:

Kraków, 2016

„Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, videogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej " sądem koleżeńskim "”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.”

Kraków, dnia

Podpis dyplomanta.....

Spis treści

1.	WSTĘP	4
2.	PROBLEMATYKA STWARDNIENIA ROZSIANEGO	6
3.	APLIKACJE WSPOMAGAJĄCE LECZENIE.....	8
3.1.	MOJE LECZENIE	8
3.2.	MY THERAPY	8
3.3.	MEDISAFE	9
3.4.	DZIENNICKA PACJENTA.....	10
4.	PROJEKT OPROGRAMOWANIA	12
4.1.	ZAŁOŻENIA FUNKCJONALNE I PLATFORMOWE.....	12
4.2.	ARCHITEKTURA OPROGRAMOWANIA.....	14
4.3.	WYKORZYSTANE BIBLIOTEKI.....	17
5.	SZCZEGÓŁY IMPLEMENTACYJNE.....	18
5.1.	EKRANY POCZĄTKOWE.....	18
5.2.	WIZUALIZACJA MIEJSCA ZASTRZYKU	19
5.3.	GENERATOR RAPORTÓW	23
5.4.	NOTYFIKACJE	25
6.	TESTY.....	28
6.1.	IMPLEMENTACJA TESTÓW	28
6.2.	URUCHOMIENIE TESTÓW	30
6.3.	TESTY WYDAJNOŚCI GENEROWANIA MODELU.....	30
7.	PODSUMOWANIE	32
8.	LITERATURA.....	33

1. Wstęp

Gwałtowny rozwój techniki oraz postępująca miniaturyzacja spowodowały, iż w dzisiejszych czasach większość z nas we własnej kieszeni posiada telefon komórkowy, który swoją mocą obliczeniową przewyższa komputery stacjonarne sprzed kilku lat. Te wielofunkcyjne urządzenia dawno przestały być tylko narzędziem do dzwonienia i wysyłania wiadomości tekstowych. Większość nowoczesnych smartphone'ów bez problemu radzi sobie z wyświetlaniem grafiki 3D, skomplikowanymi obliczeniami, wyświetlaniem notyfikacji użytkownikowi czy harmonogramowaniem planu dnia. Najczęściej wszystkie wymienione wyżej funkcje służą do dostarczania nam rozrywki, jednak coraz większa część producentów sprzętu oraz oprogramowania zaczyna zauważać w urządzeniach mobilnych bardzo duży potencjał do ułatwiania życia ludziom chorym i niepełnosprawnym. W ostatnich latach zaczęło powstawać bardzo dużo aplikacji mobilnych, których głównym przeznaczeniem jest pomóc np. osobom niewidomym w poruszaniu się po mieście. Przykładem takich rozwiązań może być „Seeing Assistant move”, czyli aplikacja, która potrafi wyszukać trasę, pomaga w poruszaniu się osobie niewidomej, rejestruje przebytą drogę czy relacjonuje obecną lokalizację. Aplikację można znaleźć pod adresem <http://seeingassistant.tt.com.pl>. Innym ciekawym zastosowaniem nowoczesnych technologii w służbie osobom chorym, może być aplikacje porządkująca wspomnienia i ułatwiająca sporządzanie notatek osobom cierpiącym na chorobę Alzheimera.

Dlatego też w niniejszej pracy podjęto próbę stworzenia podobnej aplikacji, która w tym przypadku wspomagałaby osoby chore na stwardnienie rozsiane, które do tej pory były pomijane podczas prób tworzenia różnego rodzaju oprogramowania wspomagającego. Stwardnienie rozsiane jest poważną i nieuleczalną chorobą układu nerwowego, w której dochodzi do nieodwracalnego uszkodzenia tkanki nerwowej. Jedyną szansą dla chorych jest regularne przyjmowanie leków z rodziny interferonów, które zwalniają postęp choroby, łagodzą jej objawy i dają szansę na normalne życie – osobiste i zawodowe. Lek przyjmowany jest w postaci zastrzyku w określone miejsce ciała według przyjętego, rygorystycznego harmonogramu. Pacjent zobowiązany jest do prowadzenia dziennika, w którym prowadzi ewidencję przyjmowanych dawek oraz notuje skutki uboczne towarzyszące terapii.

Celem pracy było stworzenie aplikacji mobilnej wspomagającej chorego w przyjmowaniu leku. Głównym założeniem projektu było całkowite zastąpienie

niewygodnego w prowadzeniu, nieczytelnego oraz łatwego do zgubienia papierowego dzienniczka przedstawioną w pracy aplikacją. Program ponadto ma pełnić funkcję wspomagającą w planowaniu procesu leczenia, zarządzaniu zasobami leku oraz ułatwiać kontakt z lekarzem prowadzącym pacjenta. Powstałe oprogramowanie jest dedykowane dla urządzeń z systemem Android.

W pierwszych dwóch rozdziałach nakreślono krótko problematykę stwardnienia rozsianego, opis choroby, jej przebiegu wraz z najpopularniejszymi objawami oraz dokonano przeglądu istniejących rozwiązań, które w mniejszym lub większym stopniu mogą wspomóc osoby chore. Kolejny rozdział został poświęcony przedstawieniu założeń funkcjonalnych, jakie ma spełniać opisywana aplikacja oraz wymaganiom platformowym i ograniczeniom stawianym oprogramowaniu. Następne rozdziały opisują szczegóły implementacji, opis problemów jakie musiały zostać rozwiązane w trakcie realizacji aplikacji, a także diagramy oraz zastosowane techniki dobrego programowania. W ostatnim rozdziale pracy przedstawiono testy funkcjonalne, jakim został poddany projekt, aby spełniać rygorystyczne normy niezawodności, jakie są stawiane aplikacjom pełniącym funkcje medyczne oraz dokonano krótkiego podsumowania projektu.

2. Problematyka stwardnienia rozsianego

Stwardnienie rozsiane (SM) to choroba wpływająca na mózg i rdzeń kręgowy. W wyniku jej działania osoba chora traci kontrolę nad mięśniami oraz ma problemy z czuciem i odbieraniem bodźców zewnętrznych. Przy chorobie tej nerwy w mózgu i rdzeniu kręgowym zostają uszkodzone przez własny układ odpornościowy. Stan taki jest nazywany chorobą autoimmunologiczną, czyli chorobą, z powodu której system odpornościowy organizmu zaczyna atakować zdrowe tkanki. W przypadku SM układ immunologiczny atakuje mózg oraz rdzeń kręgowy – składniki ośrodkowego układu nerwowego.

SM według Towarzystwa Stwardnienia Rozsianego jest, poza urazami po wypadkach, najczęstszą przyczyną niepełnosprawności neurologicznej, osób dorosłych do 50 roku życia. Stwardnienie rozsiane występuje dwa do trzech razy częściej u kobiet niż u mężczyzn, a jego występowanie jest bardzo rzadko spotykane przed okresem dojrzewania. Zwiększone ryzyko rozwoju choroby występuje szczególnie u osób nastoletnich do wieku 50 lat i stopniowo obniża się w kolejnych latach. Lekarze nie są pewni, co wywołuje chorobę, ale istnieją badania sugerujące, że wpływ na możliwość zachorowania ma genetyka, środowisko otaczające daną osobę, a nawet wirus.

Objawy stwardnienia rozsianego różnią się w zależności od osoby i często zmieniają się w czasie u tego samego pacjenta. Najczęstsze z nich obejmują:

- osłabienie mięśni,
- pogorszoną koordynację,
- zamazane lub zamglone widzenie,
- ból oczu,
- podwójne widzenie.

W miarę postępu choroby objawy mogą ponadto obejmować sztywnienie mięśni oraz bóle, trudności z kontrolowaniem oddawania moczu lub problemy z umiejętnościami poznawczymi. Więcej informacji na temat choroby można znaleźć pod adresem <http://www.stwardnienierozsiane.net/>.

Dostępny jest dość szeroki wachlarz nowoczesnych leków, które coraz skuteczniej zmniejszają częstotliwość i nasilenie objawów. Niektóre z nich mogą również spowalniać postęp niektórych rodzajów choroby. Jednym z dostępnych leków jest interferon beta 1b, który jest lekiem uwzględnionym w opisywanej pracy z uwagi przede wszystkim na fakt, iż osoba zainteresowana aplikacją używa właśnie tego leku.

Interferon beta-1b jest rodzajem białka z grupy interferonów. Jest on stosowany wśród pacjentów, u których po raz pierwszy wystąpiły objawy wskazujące na wysokie ryzyko wystąpienia SM. Kuracja polega na wstrzykiwaniu substancji w odpowiednie ściśle określone miejsce na ciele co 48 godzin. Kolejne obszary zastrzyków są ustalane w taki sposób, aby odstęp pomiędzy kolejnymi dawkami był jak największy, z uwagi na możliwe niepożądane objawy. Lek może powodować także bardzo nieprzyjemne skutki uboczne, takie jak:

- objawy grypopodobne (dreszcze, gorączka, bóle mięśniowo-stawowe, bóle głowy, osłabienie, nudności)
- niedokrwistość,
- stany zapalne i inne objawy w miejscu wstrzyknięcia,
- depresja,
- bezsenność i wymioty,
- zaburzenia czynności tarczycy.

Dlatego bardzo ważnym czynnikiem ułatwiającym skuteczne i w miarę bez objawowe leczenie są regularne zastrzyki według harmonogramu, zapisywanie wszystkich objawów i częste konsultacje z lekarzem. We wszystkich powyższych czynnościach ma pomagać budowana aplikacja.

3. Aplikacje wspomagające leczenie

W momencie pisania pracy istnieje dość duży wachlarz aplikacji, które mogą wspomóc osoby, chore na stwardnienie rozsiane, przyjmujące interferon. Są to jednak programy ogólnego przeznaczenia, pisane z myślą o szerokiej grupie odbiorców i dlatego nieposiadające specjalistycznych funkcji przydatnych pacjentom z MS. Mogą one, dlatego jedynie wspomóc pacjenta, ale nie są w stanie całkowicie zastąpić papierowego dzienniczka. W poniższym rozdziale dokonano przeglądu istniejących rozwiązań oraz krótkiego opisu wybranych najciekawszych pozycji.

3.1. Moje leczenie

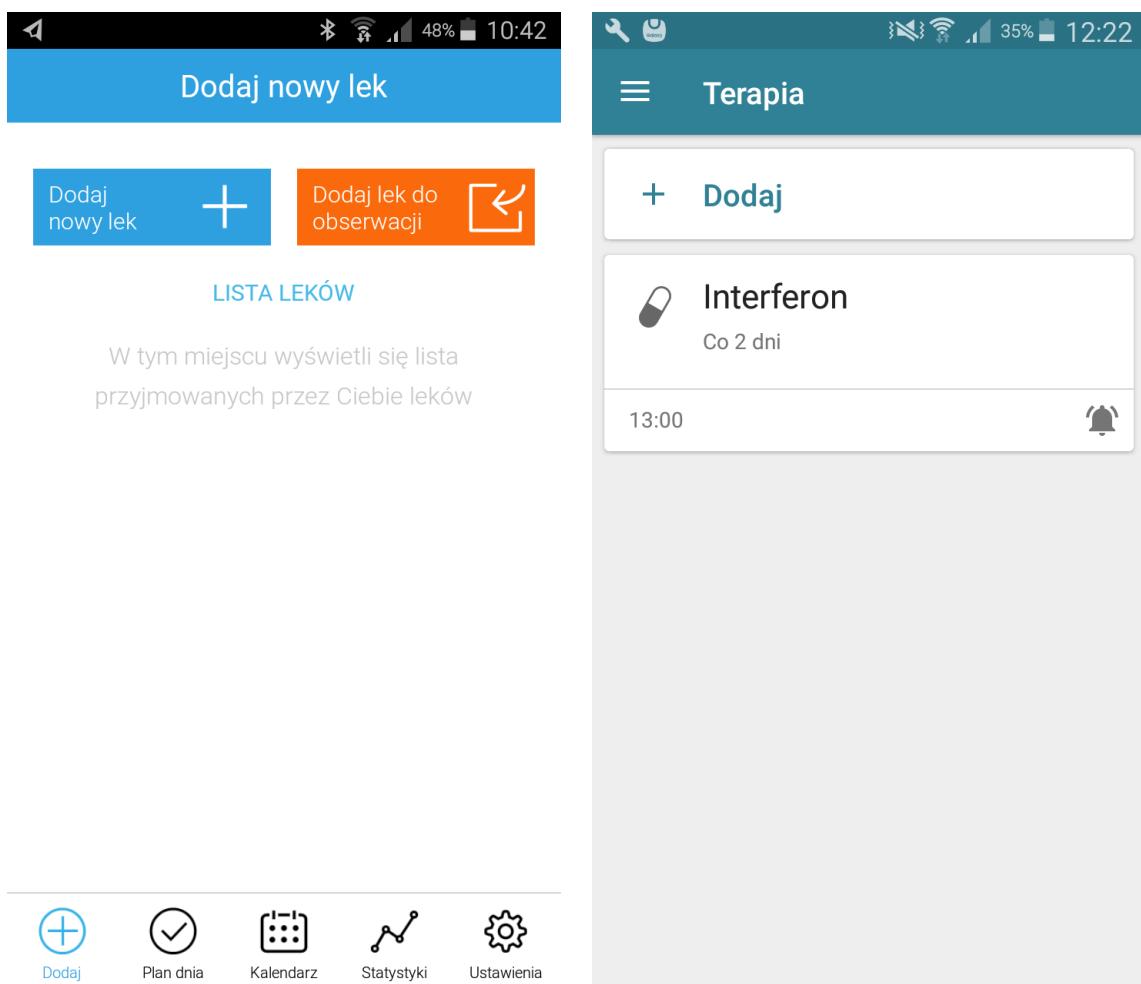
Aplikacja pozwala w prosty sposób zaplanować harmonogram przyjmowania leków lub wizytę u lekarza. Dodatkowo za pomocą notyfikacji, przypomina o każdej dawce leku i wydarzeniu. Umożliwia także bliskim sprawdzenie czy pacjent przyjmuje regularnie swoje leki. Program może być wykorzystany przez osoby chore na stwardnienie rozsiane do przypominania o kolejnym zastrzyku lub wizycie u lekarza, jednak nie posiada bardzo istotnych funkcji z punktu widzenia osoby cierpiącej na SM takich jak dodawanie niepożądanych objawów, generowanie oraz wysyłanie raportu do lekarza czy wskazywanie miejsca przyjęcia kolejnej dawki leku. Program nie posiada także w swojej bazie leków interferonu, co także stanowi pewne utrudnienie, osoba korzystająca z aplikacji musi więc zamiast interferonu wpisać inny lek i pamiętać, że jest on tylko zamiennikiem. Przykładowy ekran aplikacji pokazano na rysunku 1.

3.2. My Therapy

Aplikacja pozwala na dodanie harmonogramu zastrzyków, przypomina o zbliżającej się dawce leku oraz pozwala na wysyłanie bliskim powiadomienia, jeżeli pacjent pominie dawkę leku. Umożliwia też wprowadzenie dodatkowych parametrów takich jak ciśnienie krwi czy wyniki badań. Posiada także dodatkowo, bardzo ważną dla osób chorych na stwardnienie rozsiane, funkcje generowania i drukowania miesięcznych raportów z przebiegu leczenia. Rozwiązanie nie implementuje funkcji pomocnych dla opisywanych w pracy problemów, czyli podglądu miejsca zastrzyku, zapisywania szczegółowych informacji związanych z daną dawką leku czy wysyłania raportu z wybranych zastrzyków w formie wiadomości email. Przykładowy ekran aplikacji pokazano na rysunku 2.

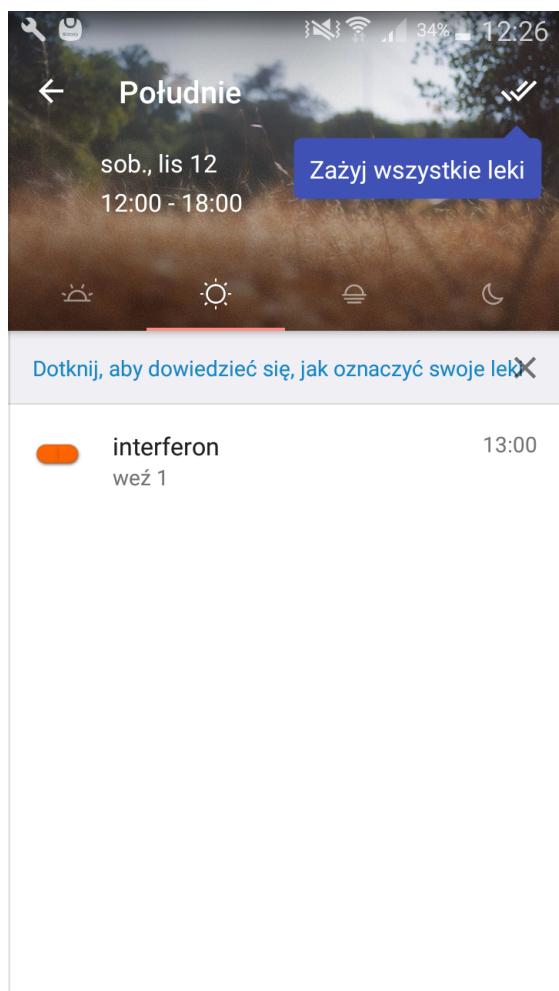
3.3. Medisafe

Aplikacja, podobnie jak prezentowane w poprzednich paragrafach, posiada wiele funkcji ogólnego przeznaczenia, takie jak powiadomienia o konieczności przyjęcia kolejnej dawki leku, wprowadzenie dodatkowych danych o stanie zdrowia pacjenta, wysyłanie powiadomień, także do innych wybranych osób, które pomagają pacjentowi w przyjmowaniu leków, generowanie raportów z leczenia i możliwość ich szybkiego wysyłania do lekarza, a także po darmowej rejestracji archiwizowanie historii zastrzyków po stronie serwera. Tak jak w poprzednich aplikacjach brakuje jednak wielu potrzebnych funkcji dla pacjentów chorych na MS. Przykładowy ekran aplikacji Medisafe przedstawiono na rysunku 3.



Rys. 1 Główny ekran aplikacji Moje leczenie [opracowanie własne]

Rys. 2. Główny ekran aplikacji My Therapy [opracowanie własne]

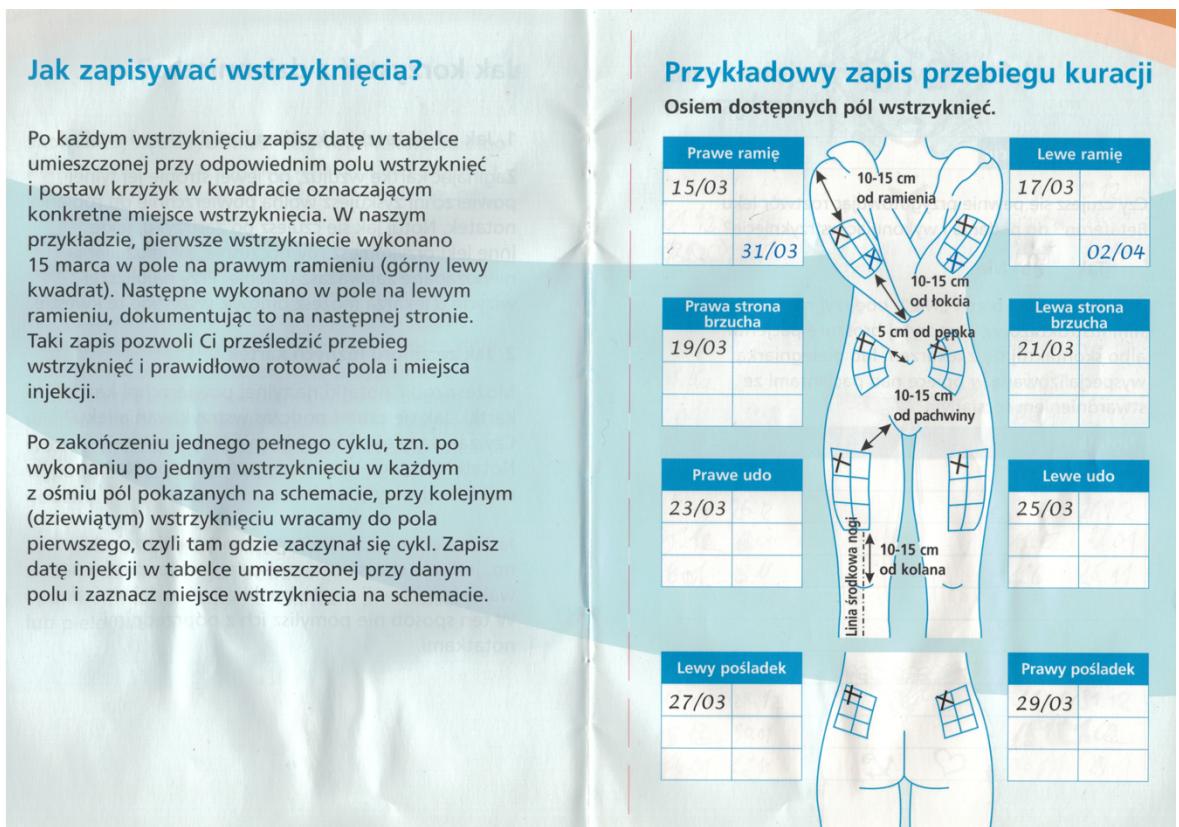


Rys. 3. Główny ekran aplikacji Medisafe [opracowanie własne]

3.4. Dzienniczek Pacjenta

Papierowy dzienniczek, zaprojektowany przez producenta leku, jest najczęściej stosowanym i jedynym obecnie istniejącym kompleksowym rozwiązaniem dla pacjentów. Umożliwia on planowanie dni kolejnych dawek, pokazuje schemat kolejnych zastrzyków oraz posiada część przeznaczoną na zapisywanie ewentualnych problemów z przyjmowaniem leku oraz notowania skutków ubocznych. Pacjent musi jednak samemu pamiętać o zaaplikowaniu zastrzyku w danym dniu o konkretnej ustalonej godzinie oraz o uzupełnieniu zapasu leku. Chory może jedynie wspomóc się jednym z wyżej opisanych rozwiązań lub zwykłym kalendarzem w telefonie. Papierowy dzienniczek, nie ułatwia także komunikacji z lekarzem oraz archiwizowania wyników. Pacjent musi zabierać ze sobą dzienniczek na każdą konsultację, a ewentualne zagubienie dzienniczka, w którym zapisana jest duża część procesu leczenia, może stanowić spory problem. Biorąc jednak pod uwagę sposób działania istniejących aplikacji wspomagających leczenie jest to obecnie najlepszy

sposób na sprawniejsze organizowanie leczenia. Przykładową stronę dzienniczka, pokazującą sposób wizualizacji miejsc zastrzyku pokazano na rysunku 4.



Rys. 4. Przykładowa strona z dzienniczka pacjenta [opracowanie własne]

4. Projekt oprogramowania

4.1. Założenia funkcjonalne i platformowe

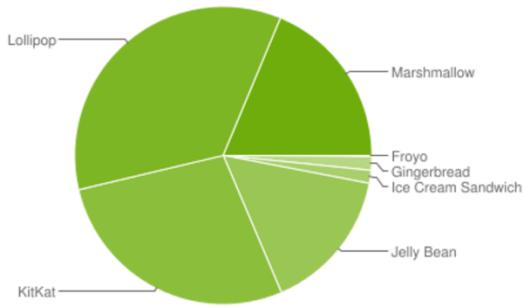
Celem pracy jest stworzenie aplikacji wspomagającej chorego w przyjmowaniu leku interferon beta 1b. Aplikacja ma pozwalać na zdefiniowanie harmonogramu powiadomień o zbliżającym się czasie aplikacji leku oraz wskazywać miejsce wykonania zastrzyku. Użytkownik ma możliwość wprowadzić dodatkowe informacje, takie jak skutki uboczne czy problemy w aplikacji zastrzyku. Dodatkowo aplikacja ma udostępniać prosty sposób zarządzania posiadanymi zasobami leku oraz przypominać o kończącym się ich zapasie. Możliwe ma być przeglądanie historii terapii wraz ze szczegółowymi informacjami o objawach czy innych dolegliwościach im towarzyszącym. Program ma pozwalać ponadto na generowanie łatwego w interpretacji raportu dla lekarza prowadzącego bądź pielęgniarki i wysłanie uprzednio wygenerowanego pliku za pomocą wiadomości sms lub email. Bardzo istotną funkcją, z uwagi na bardzo często występujące u osób cierpiących na stwardnienie rozsiane problemy ze wzrokiem, jest przejrzysty interfejs i wygodne sterowanie, co zostało osiągnięte poprzez możliwość zmiany wyglądu aplikacji na wysoki kontrast, zastosowanie dużych i czytelnych czcionek oraz przemyślaną nawigację po aplikacji. Na podstawie powyższego opisu wraz z osobą zainteresowaną aplikacją sporządzono historie użytkowników, które ułatwiły proces planowania oraz zarządzania pracą w poszczególnych etapach projektu. Następnie każdy z punktów wymaganych do pełnego zakończenia prac nad projektem wypunktowano pod względem ważności oraz czasochłonności w odniesieniu do całości projektu. Pozwoliło to na lepszą organizację harmonogramu prac i ewentualną redukcję opóźnień lub zmianę wymagań. Wyżej opisywane historie użytkowników prezentują się następująco:

- Jako użytkownik:
 - Chcę mieć możliwość wprowadzenia i zapamiętania moich danych w celu łatwiejszego i szybszego zatwierdzania zastrzyków, generowania raportów oraz ich wysyłania (3pkt).
 - Chcę mieć możliwość wprowadzenia pożąданiej godziny wykonywania zastrzyków oraz dnia pierwszej aplikacji w celu zarządzania procesem leczenia (1pkt).

- Chcę mieć możliwość utrzymywania notyfikacji o zbliżającym się zastrzyku w celu przypomnienia mi o kolejnej dawce (5pkt).
- Chcę mieć możliwość podglądu miejsca zbliżającego się zastrzyku, aby zaaplikować lek w odpowiednie miejsce ciała (13pkt).
- Chcę mieć możliwość odłożenia notyfikacji, jeżeli w danym momencie nie mogę wykonać zastrzyku (3pkt).
- Chcę mieć możliwość wygenerowania raportu z jednego lub wielu zastrzyków na raz w celu łatwego dokumentowania procesu leczenia (5pkt).
- Chcę mieć możliwość wysłania wygenerowanego raportu do lekarza lub pielęgniarki na podany wcześniej adres email lub nr telefonu w celu łatwiejszej i szybszej komunikacji (1pkt).
- Chcę mieć możliwość zmiany wprowadzonych wcześniej ustawień w celu łatwiejszej personalizacji aplikacji (3pkt).
- Chcę mieć wgląd do pomocy/samouczka w razie problemów z obsługą aplikacji (3pkt).
- Jako lekarz lub pielęgniarka:
 - Chcę mieć możliwość otrzymania w przejrzystej formie raportu z przyjmowanie zastrzyków, w którym zawarte będą miejsca aplikacji leku oraz zaobserwowane przez pacjenta niepożądane objawy.

Dodatkowo założono, że aplikacja będzie współpracować z urządzeniami, na których zainstalowano system operacyjny Android w wersji przynajmniej 4.4 KitKat. Wybrana minimalna wersja pozwala na zastosowanie w pisanej aplikacji wszystkich niezbędnych funkcji, przy zachowaniu stosunkowo dużego wsparcia dla większości użytkowników. Biorąc pod uwagę dane z września 2016r. (rysunek 4), aplikacja wspiera ponad 81.4% wszystkich urządzeń z zainstalowanym systemem Android.

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	1.5%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.4%
4.1.x	Jelly Bean	16	5.6%
4.2.x		17	7.7%
4.3		18	2.3%
4.4	KitKat	19	27.7%
5.0	Lollipop	21	13.1%
5.1		22	21.9%
6.0	Marshmallow	23	18.7%



Data collected during a 7-day period ending on September 5, 2016.

Any versions with less than 0.1% distribution are not shown.

Rys. 5. Fragmentacja systemu Android we wrześniu 2016. [źródło: <http://www.droid-life.com/2016/09/13/android-distribution-september/>]

Tworząc aplikację mobilną, należy także zawsze zwrócić szczególną uwagę na wydajność, z uwagi na ograniczone zasoby sprzętowe. Założono, iż miejsce kolejnego zastrzyku będzie wyświetlane na modelu 3D, co spowoduje dużą czytelność tego ekranu dla użytkownika. Jest to jednak miejsce potencjalnego spowolnienia działania aplikacji, szczególnie na telefonach z gorszymi parametrami sprzętowymi, co należy wziąć pod uwagę podczas implementacji oraz testowania.

4.2. Architektura oprogramowania

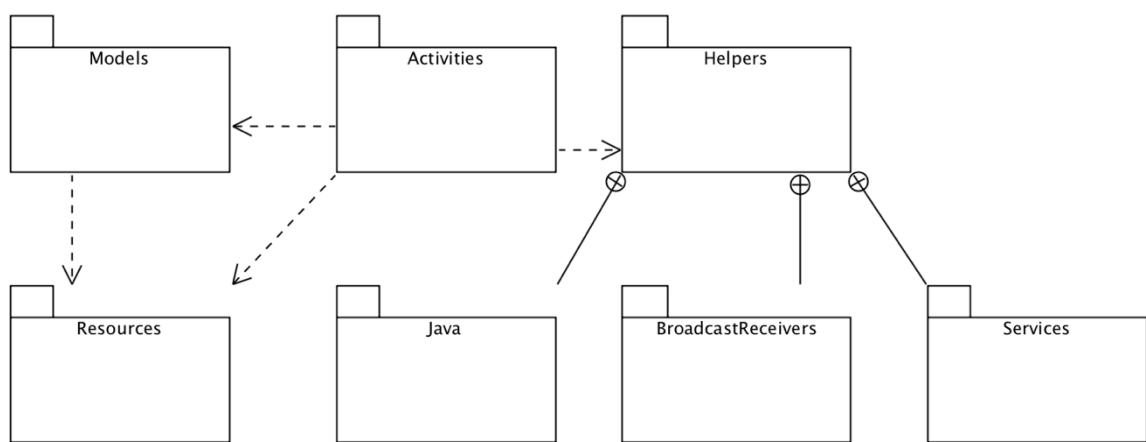
Dobre oprogramowanie wymaga przemyślanego projektu. Dlatego przed przystąpieniem do prac związanych z implementacją, wykonano diagramy UML kluczowych elementów systemu oraz zaprojektowano strukturę całego projektu.

Klasy używane w projekcie zostały podzielone na kilka grup:

- Activities/Fragments – klasy aktywności i fragmentów, czyli klasy związane z jednym pojedynczym ekranem lub jego częścią, odpowiedzialne za komunikację pomiędzy klasami modelowymi a interfejsem użytkownika, zawierające całą z tym związaną logikę oraz definiujące sposób wyświetlania interfejsu użytkownika
- Models - klasy modelowe, zawierające klasy encyjne mapowane na tabele w bazie danych oraz logikę biznesową związaną z tymi encjami.

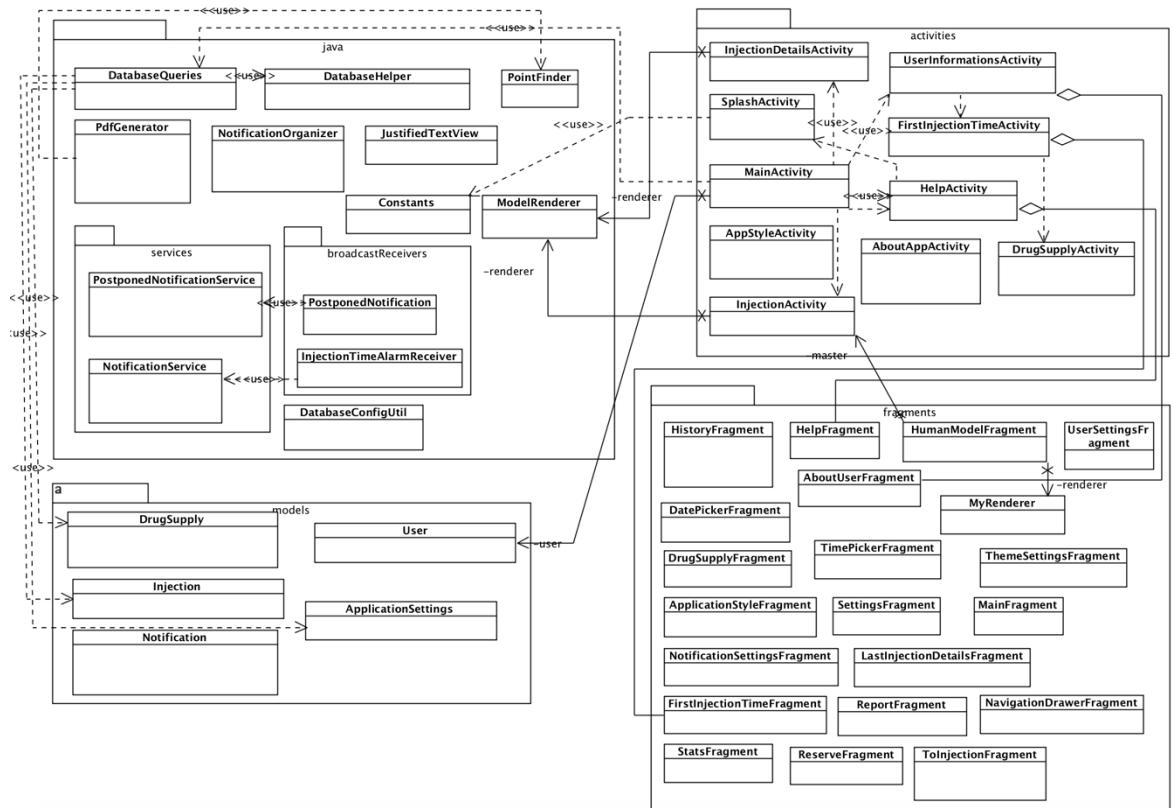
- Java/BroadcastReceivers/Services - klasy pomocnicze np. zarządzające powiadomieniami, odbierające powiadomienia oraz wykonujące operację na obiektach klas encyjnych, takich jak uaktualnienie stanu leku czy informacji o użytkowniku. Są to klasy używane w wielu miejscach aplikacji, dzięki temu kod jest bardziej przejrzysty, a zmiana logiki związanej z działaniem jednej z nich nie ponosi za sobą wielu zmian w projekcie.
- Resources - pliki XML definiujące wygląd interfejsu użytkownika, pliki graficzne rastrowe i wektorowe oraz pliki XML zawierające wszystkie etykiety używane w projekcie.

Przyjęty podział obrazuje zamieszczony poniżej na rysunku 6 diagram pakietów.



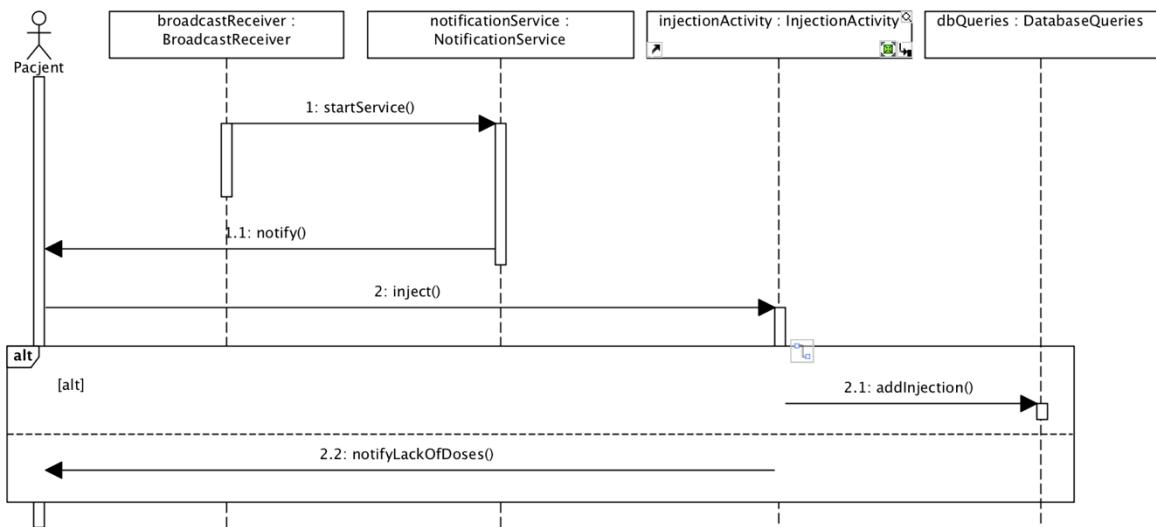
Rys. 6. Diagram pakietów tworzonej aplikacji [opracowanie własne]

Dodatkowo wykonano diagram klas widoczny na rysunku 7, na którym pokazano najważniejsze klasy w projekcie oraz najbardziej istotne relacje między nimi.



Rys. 7. Diagram klas użytych w projekcie [opracowanie własne]

Kluczowymi funkcjami, jakie ma spełniać aplikacja są przypomnienia o zbliżającym się zastrzyku oraz notyfikowanie o zbliżającym się czasie kolejnej dawki, dlatego wykonano dla nich dodatkowo diagram sekwencji widoczny na rysunku 8. Dzięki temu łatwo zobaczyć przepływ sterowania pomiędzy obiektami, a także ogólny schemat algorytmu.



Rys. 8. Diagram sekwencji dla procesu wykonywania zastrzyku [opracowanie własne]

4.3. Wykorzystane biblioteki

Podczas implementacji aplikacji starano się używać jak najczęściej funkcji dostępnych w oficjalnym SDK systemu Android bez używania dużej ilości bibliotek zewnętrznych. Takie podejście gwarantuje łatwe utrzymanie i rozwój aplikacji w przyszłości poprzez uniezależnienie kodu od zewnętrznych implementacji. Wybrano jednak kilka dodatkowych bibliotek, które w znacznych stopniu ułatwiły pracę nad projektem oraz poprawiły czytelność kodu oraz ogólny wygląd aplikacji.

- PCT-AE- jest najważniejszą biblioteką użytą w aplikacji, udostępnia ona spójny interfejs ułatwiający pisanie aplikacji mobilnych wykorzystujących grafikę 3D. Biblioteka wykorzystuje znikome zasoby systemu i jest bardzo wydajna z uwagi na to, że została oparta na OpenGL ES, czyli oficjalnym API dla grafiki 3D w systemie Android [5].
- MPAndroidChart- biblioteka przeznaczona do wyświetlania wykresów. W opisywanym projekcie została użyta do stworzenia ekranu wyświetlającego wykres podsumowującyczęstość występowania każdego rodzaju niepożądanych objawów [10].
- ORMLite- biblioteka znacznie ułatwiająca pracę z bazą danych. Jest to biblioteka oferująca mapowanie obiektowo-relacyjne, co umożliwia uniknięcie pisania zapytań SQL w obiektowym kodzie aplikacji. Dzięki temu znaczco ułatwiono obsługę bazy danych i czytelność obiektowego kodu [8].
- Material Drawer- jest wysoce konfigurowalną biblioteką umożliwiającą uzyskanie efektów identycznych jak w specyfikacji material design [11] na wersjach systemu niższych niż 5.0 Lollipop [6].
- Espresso- biblioteka służąca do pisania automatycznych niezawodnych testów interfejsu użytkownika. W aplikacji była główną biblioteką używaną do przetestowania najważniejszych funkcji programu [9].
- iText- jest biblioteką, która pozwala w łatwy i czytelny sposób tworzyć, dostosowywać, kontrolować i zmieniać dokumenty w formacie PDF. Generowanie dokumentów i raportów może odbywać się na podstawie danych z pliku XML lub bazy danych [7].

5. Szczegóły implementacyjne

5.1. Ekrany początkowe

Proces implementacji rozpoczęto od stworzenia ekranów odpowiedzialnych za zebranie niezbędnych informacji o użytkowniku, takich jak: adres email do wysyłania raportów, dacie wykonania pierwszego zastrzyku, godzinie wyświetlenia powiadomienia czy aktualnej ilości dawek leku posiadanych przez pacjenta. Głównymi zadaniami tych klas są zebranie informacji od użytkownika, przetworzenie ich i zapisanie w bazie danych oraz przejście do następnego ekranu z pytaniami. Poniżej na listingu 1 zaprezentowano przykładowy fragment kodu, który w tym przypadku jest odpowiedzialny za ustawienie godziny notyfikacji.

Listing. 1. Klasa FirstInjectionTimeActivity

```
public static final String HOUR="hour";
public static final String MINUTE="minute";

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    int style=getIntent().getIntExtra(AppStyleActivity.USER_STYLE, 0);
    if(style == 2) {
        setTheme(R.style.darkAppTheme);
    }
    setContentView(R.layout.layout_first_injection_time);
}

public void onButtonNextClick(View view){
    int[] time = getTime();
    showDatePickerFragment(time);
}

private int[] getTime() {
    TimePicker time=(TimePicker) findViewById(R.id.timePicker);
    int hour, minute;
    if(Build.VERSION.SDK_INT < 23) {
        hour = time.getCurrentHour();
        minute = time.getCurrentMinute();
    } else{
        hour = time.getHour();
        minute = time.getMinute();
    }
    return new int[] {hour, minute};
}

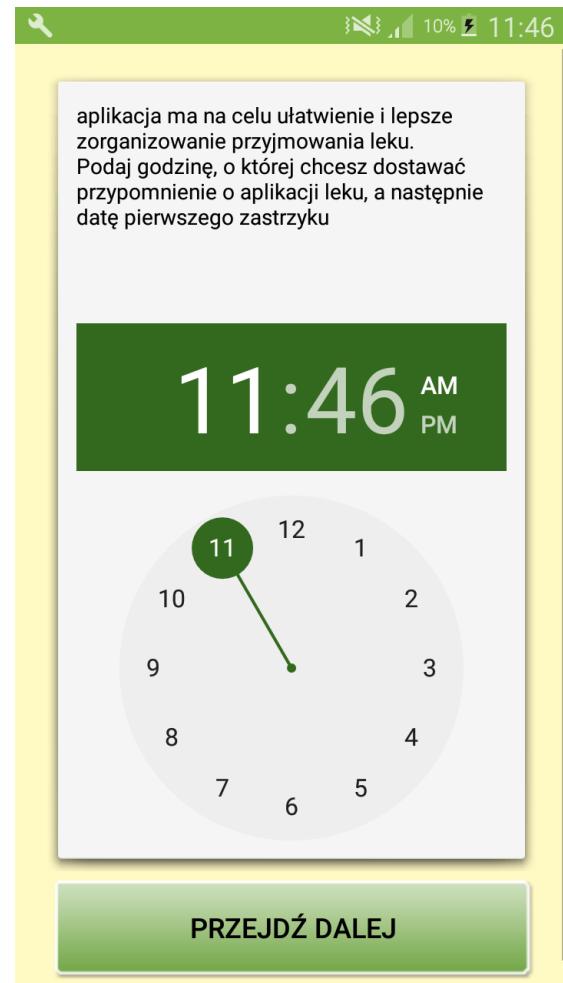
private void showDatePickerFragment(int[] time) {
    Bundle bundle=new Bundle();
    bundle.putInt(FirstInjectionTimeActivity.HOUR, time[0]);
    bundle.putInt(FirstInjectionTimeActivity.MINUTE, time[1]);
    DialogFragment newFragment = new DatePickerFragment();
    newFragment.setArguments(bundle);
    newFragment.show(getFragmentManager(), "datePicker");
}
```

Na listingu 1 przedstawiono pojedynczą aktywność odpowiedzialną za wyświetlenie okienka dialogowego z zegarem i pobranie godziny, w której użytkownik chce wykonywać zastrzyki. Po wciśnięciu przez użytkownika przycisku, odczytywany jest podany czas

notyfikacji, zapisywany w celu przekazania go kolejnej aktywności, a następnie uruchamiany jest kod odpowiedzialny za ustawienie daty pierwszego zastrzyku. Ekran wskazywania czasu zastrzyku oraz podawania informacji o użytkowniku pokazano na rysunkach 9 i 10.



Rys. 9. Ekran informacji początkowych [opracowanie własne]



Rys. 10. Ekran podawania czasu zastrzyku [opracowanie własne]

5.2. Wizualizacja miejsca zastrzyku

Bardzo ważnym elementem programu jest czytelne wskazywanie miejsca kolejnego zastrzyku. Zdecydowano, że najlepszym wyborem będzie model 3D człowieka, który poprzez odpowiednie ustawienie i przybliżenie kamery będzie wskazywał miejsce danej dawki. Jak wspomniano w podrozdziale dotyczącym projektu oprogramowania, zdecydowano się na skorzystanie z biblioteki JPCT AE, która udostępnia prosty i przejrzysty interfejs do pracy z niskopoziomową biblioteką graficzną OpenGL ES. Model człowieka osadzono na standardowym komponencie GLSurfaceView służącym do wyświetlania treści

wygenerowanej przy pomocy OpenGL ES. Poniżej przedstawiono fragmenty kodu związane z wyświetlaniem modelu oraz omówiono najważniejsze ich fragmenty.

Aktywnością odpowiedzialną za pokazanie ekranu z modelem pokazującym kolejne miejsce zastrzyku jest klasa `InjectionActivity`. Poniżej na listingu 2 pokazano metodę wspomnianej klasy, w której ustawiany jest nowy model z odpowiednio dobraną kamerą w zależności od miejsca zastrzyku.

Listing. 2. Metoda `setRenderer()` klasy `InjectionActivity`

```
private void setRenderer() {
    mGLView = (GLSurfaceView) findViewById(R.id.glSurfaceView);
    mGLView.setEGLConfigChooser(new GLSurfaceView.EGLConfigChooser() {
        public EGLConfig chooseConfig(EGL10 egl, EGLDisplay display) {
            int[] attributes = new int[]{EGL10.EGL_DEPTH_SIZE, 16, EGL10.EGL_NONE};
            EGLConfig[] configs = new EGLConfig[1];
            int[] result = new int[1];
            egl.eglChooseConfig(display, attributes, configs, 1, result);
            return configs[0];
        }
    });
    View view=findViewById(R.id.layout_injection);
    int[] injectionPoint=PointFinder.findPoint(this);
    renderer = new ModelRenderer(this, view, 35, "model.3DS", injectionPoint[0],
        injectionPoint[1]);
    mGLView.setRenderer(renderer);
}
```

Pokazana wyżej metoda jest wywoływana przy starcie aktywności i jest odpowiedzialna za utworzenie obiektu klasy `ModelRenderer` i ustawienie go we wspomnianym wcześniej komponencie wyświetlającym grafikę. Klasa odpowiedzialna za stworzenie modelu człowieka to `ModelRenderer`. Jest to klasa, w której generowana jest cała scena tzn. ustawiane jest odpowiednie światło, kolory, dodawana jest kamera oraz rysowany jest model człowieka. Poniżej na listingu 3 przedstawiono fragment omawianej klasy wraz z jej głównymi polami.

Listing. 3. Deklaracja klasy `ModelRenderer`

```
public class ModelRenderer implements GLSurfaceView.Renderer {

    private FrameBuffer fb = null;
    private RGBColor back = new RGBColor(245, 245, 245, 0);
    private float touchTurn = 0;
    private float touchTurnUp = 0;
    private World world;
    private float xpos = -1;
    private float ypos = -1;
    private Object3D cube = null;
    private int fps = 0;
    private Light sun = null;
    private String modelName;
    private int modelScale;
    private Context context;
    private Context master;
    private int area;
```

```

private int point;
int scaleFactor=40;
private long time = System.currentTimeMillis();

```

Jak widać na powyższym listingu klasa implementuje interfejs Renderer, który jest ogólnym interfejsem implementowanym przez klasy odpowiedzialne za generowanie grafiki. Interfejs Renderer deklaruje 3 abstrakcyjne metody, które zostały zaimplementowane w omawianej klasie. Jako właściwości klasy przechowywane są obiekty związane z wyświetlaniem grafiki oraz zmienne pomocnicze, takie jak obecny czas czy ilość klatek na sekundę.

Ładowanie modelu człowieka z pliku odbywa się wewnątrz metody loadModel, która została zaprezentowana na listingu 4.

Listing. 4. Metoda loadModel klasy ModelRenderer

```

private Object3D loadModel(String filename, float scale) throws IOException {
    InputStream stream = context.getResources().getAssets().open(filename);
    Object3D[] model = Loader.load3DS(stream, scale);
    Object3D o3d = new Object3D(0);
    Object3D temp = null;
    for (int i = 0; i < model.length; i++) {
        temp = model[i];
        temp.setCenter(SimpleVector.ORIGIN);
        temp.rotateZ((float) (0.5 * Math.PI));
        temp.rotateY((float) (0.5 * Math.PI));
        temp.rotateMesh();
        temp.setRotationMatrix(new Matrix());
        o3d = Object3D.mergeObjects(o3d, temp);
    }
    return o3d;
}

```

Wewnątrz metody pokazanej na powyższym listingu z pliku ładowany jest wskazany model, a następnie następuje odpowiednie ustawienie każdego z jego wierzchołków. Metoda loadModel wywoływana jest podczas konstrukcji nowego obiektu.

Konstrukcja nowego obiektu ModelRenderer pokazana została na poniższym listingu 5.

Listing. 5. Konstruktor klasy ModelRenderer

```

public ModelRenderer(Context activity, final View view, int scale, String name, int
area,
                     int point) {
    super();
    context=activity;
    this.modelScale =scale;
    this.modelName =name;
    this.area=area;
    this.point=point;
    int appStyle = DatabaseQueries.getApplicationStyle(activity);
    if(appStyle == 2) {
        back = new RGBColor(40, 40, 40, 0);
    } try {

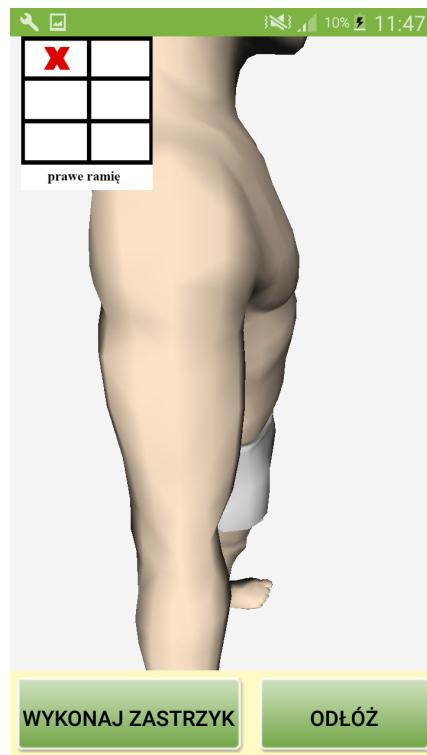
```

```

        cube = loadModel(modelName, modelScale);
    } catch (IOException e) {
        Log.e(this.getClass().toString(), e.getMessage());
    }
    view.setOnTouchListener(new View.OnTouchListener() {
        public boolean onTouch(View v, MotionEvent me) {
            if (me.getAction() == MotionEvent.ACTION_DOWN) {
                xpos = me.getX();
                ypos = me.getY();
                return true;
            } if (me.getAction() == MotionEvent.ACTION_UP) {
                xpos = -1;
                ypos = -1;
                touchTurn = 0;
                touchTurnUp = 0;
                return true;
            } if (me.getAction() == MotionEvent.ACTION_MOVE) {
                float xd = me.getX() - xpos;
                xpos = me.getX();
                touchTurn = xd / -100f;
                return true;
            }
            return view.onTouchEvent(me);
        }
    });
}
}

```

Wewnątrz konstruktora na podstawie podanego miejsca zastrzyku dodawane jest odpowiednie tło sceny dla używanego stylu oraz wywoływana jest metoda loadModel. Następnie utworzony model dodawany jest do przesłanego kontenera i ustawiana jest metoda odpowiedzialna za możliwość ruchu kamery. Ekran wykonywania zastrzyku z widocznym modelem człowieka pokazano na rysunku 11.



Rys. 11. Ekran wykonywania zastrzyku [opracowanie własne]

5.3. Generator raportów

Jak wspomniano w rozdziale dotyczącym wykorzystanych bibliotek, do generowania raportów wykorzystano bibliotekę iText. W aplikacjach tworzonych na urządzenia z zainstalowanym systemem Android 4.4 lub nowszym istnieje wbudowana klasa PrintedPdfDocument, umożliwiająca wygenerowanie pliku pdf. Zdecydowano się jednak na użycie dodatkowej biblioteki ze względu na dużo większą czytelność kodu oraz możliwość budowania bardziej złożonych dokumentów.

Wygenerowanie nowego raportu ze wskazanych zastrzyków odbywa się w klasie PdfGenerator. Do stworzenia nowego raportu służy metoda generate, którą pokazano na poniższym listingu 6.

Listing. 6. Metoda statyczna generate klasy PdfGenerator

```
public static void generate(Context context, List<Injection> list, String fileName){  
    try {  
  
        File f = new File(Environment.getExternalStorageDirectory().getAbsolutePath() +  
        "/" +  
        + fileName);  
        FileOutputStream output = new FileOutputStream(f);  
        Document document = new Document(PageSize.A4);  
        PdfAWriter.getInstance(document, output);  
        document.open();  
  
        User user = DatabaseQueries.getUser(context);  
        String text = user.getName() + "- Raport z zastrzyków: \n \n";  
  
        document.addHeader("nagłówek", text);  
        document.addTitle(text);  
        document.addAuthor(user.getName());  
        document.add(new Paragraph(text));  
  
        for (Injection currentInj : list) {  
            PdfPTable nextTable = addTable(currentInj, context);  
            document.add(nextTable);  
        }  
        document.close();  
    }  
    catch (FileNotFoundException| DocumentException e){  
        Log.e(PdfGenerator.class.toString(), "blad przy generowaniu pdf");  
    }  
}
```

Wewnątrz pokazanej wyżej metody następuje cały proces budowania nowego raportu. Tworzony jest nowy plik o wskazanej nazwie, a następnie dla każdego zastrzyku z podanej listy dodawana jest nowa tabela, zawierająca informacje o objawach, czasie zastrzyku oraz jego miejscu [7].

Za utworzenie pojedynczej tabeli dla konkretnego zastrzyku odpowiada metoda addTable, pokazana na poniższym listingu 7.

Listing. 7. Metoda statyczna addTable klasy PdfGenerator

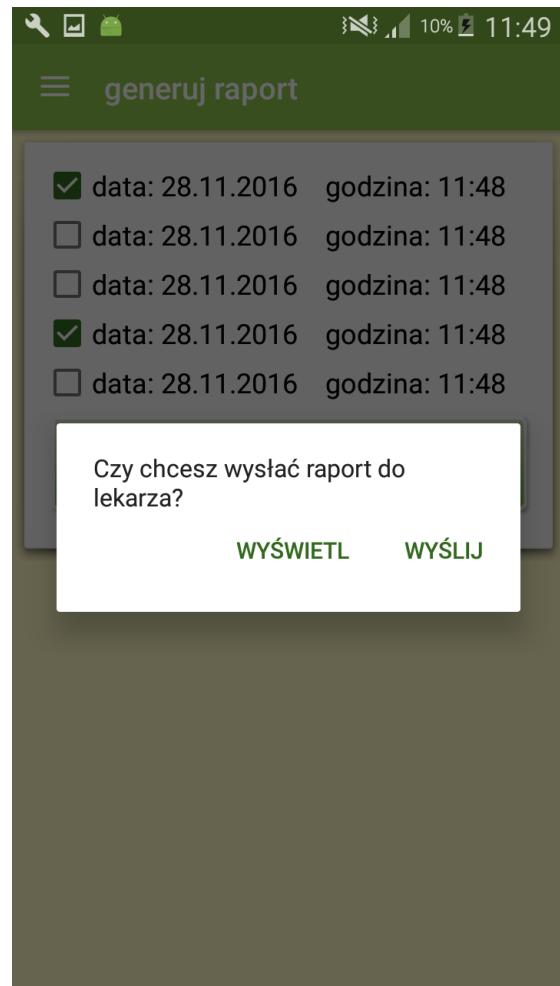
```
private static PdfPTable addTable(Injection currentInj, Context context){

    PdfPTable table = new PdfPTable(2);
    Calendar calendar = Calendar.getInstance();
    calendar.setTimeInMillis(currentInj.getTimeInMillis());
    String injectionDate = calendar.get(Calendar.DAY_OF_MONTH) + "." +
        calendar.get(Calendar.MONTH) + "." + calendar.get(Calendar.YEAR) +
        " " + calendar.get(Calendar.HOUR) + ":" + calendar.get(Calendar.MINUTE);
    PdfPCell cell=new PdfPCell(new Phrase("Data zastrzyku: "+injectionDate));
    table.addCell(cell);
    cell=new PdfPCell(new Phrase("Miejsce zastrzyku:"));
    table.addCell(cell);
    table.addCell("Objawy:");
    try {
        String field="f"+ String.valueOf(currentInj.getArea()) +
        String.valueOf(currentInj.
            getPoint());
        Drawable d = ContextCompat.getDrawable(context, context.getResources().
            getIdentifier(field, "drawable", context.getPackageName()));
        BitmapDrawable bitDw = ((BitmapDrawable) d);
        Bitmap bmp = bitDw.getBitmap();
        bmp=getResizedBitmap(bmp,200,200);
        ByteArrayOutputStream stream = new ByteArrayOutputStream();
        bmp.compress(Bitmap.CompressFormat.PNG, 100, stream);
        Image image = Image.getInstance(stream.toByteArray());
        PdfPCell imageCell = new PdfPCell(image);
        imageCell.setRowspan(2);
        table.addCell(imageCell);
    } catch (BadElementException | IOException e) {
        e.printStackTrace();
    }
    com.itextpdf.text.List list=new
    com.itextpdf.text.List(com.itextpdf.text.List.UNORDERED);
    if(currentInj.isTemperature()) {
        ListItem item = new ListItem(" -temperatura\n");
        list.add(item);
    } if(currentInj.isAche()) {
        ListItem item = new ListItem(" -ból miesni\n");
        list.add(item);
    } if(currentInj.isTrembles()) {
        ListItem item = new ListItem(" -dreszcze\n");
        list.add(item);
    }
    Phrase phrase=new Phrase();
    phrase.add(list);
    cell=new PdfPCell(phrase);
    table.addCell(cell);
    table.setSpacingAfter(10f);
    return table;
}
```

Wewnątrz metody addTable następuje konstrukcja tabeli dla jednego zastrzyku. Dodawane są wiersze z czasem zastrzyku, objawami w formie wypunktowania oraz miejscem wykonania zastrzyku. Miejsce zastrzyku pokazane jest w formie czytelnego obrazka identycznego z tym, który widać podczas wykonywania zastrzyku [7]. Proces tworzenia i wyświetlania raportu pokazano na rysunkach 12 oraz 13.



Rys. 12. Wybór zastrzyków do raportu [opracowanie własne]



Rys. 13. Wyświetlanie i wysyłanie raportu [opracowanie własne]

5.4. Notyfikacje

Kluczową funkcją w omawianej aplikacji jest informowanie pacjenta o zbliżającym się czasie leku. Poniżej na listingu 8 zaprezentowano metodę scheduleNotification klasy NotificationOrganizer, wewnątrz której ustawiana jest nowa powtarzająca się co 48 godzin notyfikacja.

Listing. 8. Metoda scheduleNotification klasy NotificationOrganizer

```
public static void scheduleNotification(long injectionTime, Activity ac) {  
    Intent broadcastIntent = new Intent(ac, InjectionTimeAlarmReceiver.class);  
  
    Calendar calendar= Calendar.getInstance();  
    calendar.setTimeInMillis(injectionTime);  
    PendingIntent pendingIntent = PendingIntent.getBroadcast(ac, 0, broadcastIntent,  
            PendingIntent.FLAG_UPDATE_CURRENT);  
    AlarmManager alarmManager = (AlarmManager)ac.getSystemService(ac.ALARM_SERVICE);  
    alarmManager.setRepeating(AlarmManager.RTC_WAKEUP, injectionTime,  
            AlarmManager.INTERVAL_DAY * 2, pendingIntent);  
}
```

Wewnątrz pokazanej wyżej metody na podstawie podanego czasu pierwszego zatrzyku, ustawiany jest powtarzający się alarm uruchamiający co 48 metodę onReceive we wskazanej klasie InjectionTimeAlarmReceiver. Dzięki temu w określonym czasie zostanie uruchomiona funkcja odpowiedzialna za wyświetlenie nowej notyfikacji o zbliżającym się zatrzyku.

Na poniższym listingu 9 pokazano klasy odpowiedzialne za wyświetlenie informacji o zatrzyku.

Listing. 9. Klasy InjectionTimeAlarmReceiver oraz NotificationService

```
public class InjectionTimeAlarmReceiver extends BroadcastReceiver {

    public void onReceive(Context context, Intent intent) {
        Intent service = new Intent(context, NotificationService.class);
        context.startService(service);
    }
}

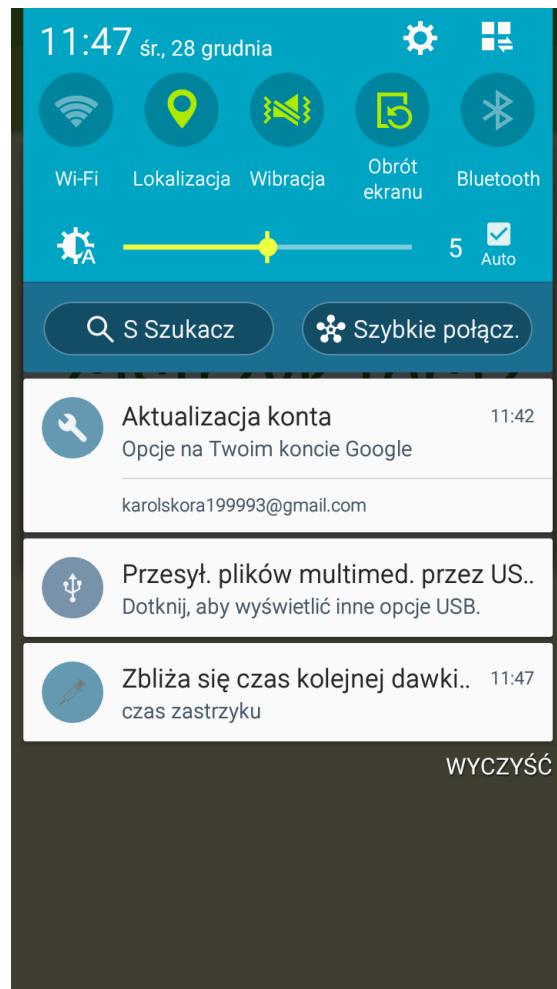
public class NotificationService extends Service {
    public final static int NOTIFICATION_ID=1;
    @Override
    public IBinder onBind(Intent intent) {

        return null;
    }
    @Override
    public int onStartCommand(Intent intent,int flags, int startId) {
        super.onStartCommand(intent, flags, startId);
        Context context = this.getApplicationContext();
        NotificationManager notificationManager = (NotificationManager)context.getSystemService(context.NOTIFICATION_SERVICE);
        Intent mIntent = new Intent(this, InjectionActivity.class);
        PendingIntent pendingIntent = PendingIntent.getActivity(context, 0, mIntent, PendingIntent.FLAG_UPDATE_CURRENT);
        NotificationCompat.Builder builder = this.getNotificationBuilder();
        builder.setContentIntent(pendingIntent);
        notificationManager.notify(NOTIFICATION_ID, builder.build());

        return START_NOT_STICKY;
    }
    private NotificationCompat.Builder getNotificationBuilder() {
        NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
        builder.setContentTitle(getString(R.string.notification_text));
        builder.setContentText(getString(R.string.notification_title));
        builder.setSmallIcon(R.mipmap.icon);
        builder.setVibrate(new long[]{1000, 1000, 1000, 1000, 1000});
        builder.setAutoCancel(true);
        Uri alarmSound =
        RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
        builder.setSound(alarmSound);
        return builder;
    }
}
```

Opisany wyżej alarm uruchamia metodę onReceive klasy InjectionTimeAlarmReceiver, która jest odpowiedzialna tylko za uruchomienie nowego serwisu pokazującego notyfikację [1]. Cały proces tworzenia notyfikacji odbywa się w klasie NotificationService. Wewnątrz metody getNotificationBuilder ustawiany jest wygląd notyfikacji tzn. sposób vibracji, tytuł, treść oraz ikona widoczna w lewym roku notyfikacji.

Metoda `onStartCommand` wyświetla utworzoną notyfikację na ekranie urządzenia [1]. Sposób wyświetlania notyfikacji pokazano na rysunku 14.



Rys. 14. Notyfikacja o zastrzyku [opracowanie własne]

6. Testy

Jak wspomniano we wstępie aplikacja, która ma służyć do celów medycznych musi działać niezawodnie, dlatego wykonano szereg testów funkcjonalnych, sprawdzających czy wszystkie najważniejsze funkcje programu działają poprawnie. Testy były uruchamiane po każdej zmianie w kodzie aplikacji, dzięki temu można było mieć pewność, że nowy kod nie wprowadził żadnego niepożądanego zachowania do aplikacji.

6.1. Implementacja testów

Do implementacji testów wykorzystano wcześniej omówioną bibliotekę Espresso. Każda klasa testowa jest oznaczona adnotacją `@LargeTest`, która informuje, że test powinien być uruchomiony jako część większych testów. W każdej klasie testowej dodano dodatkowo adnotację `@RunWith()` z podanym aliasem wersji klasy uruchamiającej testy(`AndroidJUnit4`). Dzięki temu zabezpieczono się od sytuacji w przyszłości, kiedy zmieni się domyślna wersja tej klasy [9].

Wykonano testy wszystkich najważniejszych funkcji aplikacji. Ostatecznie powstały testy sprawdzające poprawność:

- ustawień początkowych,
- wykonania zastrzyku i dodania objawów ubocznych,
- sprawdzenia historii zastrzyków oraz szczegółów jednego z nich,
- zarządzania zapasami leku: zmiana ilości leku, aktualizacja notyfikacji o kończących się zasobach,
- wygenerowania raportu dla lekarza i wyświetlenia powstałego pliku,
- zmiany ustawień:
 - informacji o użytkowniku,
 - czasu powiadomień o zastrzyku.
- wyglądu aplikacji,
- skorzystania z ekranów pomocy.

Poniżej na listingu 10 pokazano klasę testową odpowiedzialną za sprawdzenie poprawności działania wszystkich ekranów początkowych oraz prawidłowego przechodzenia między nimi.

Listing. 10. Klasa testowa FirstRunTest

```
@LargeTest
@RunWith(AndroidJUnit4.class)
public class FirstRunTest {
    @Rule
    public ActivityTestRule<SplashActivity> mActivityTestRule =
        new ActivityTestRule<>(SplashActivity.class);
    @Before
    public void clearAppInfo() {
        Activity mActivity = mActivityTestRule.getActivity();
        SharedPrefs prefs =
            PreferenceManager.getDefaultSharedPreferences(mActivity);
        prefs.edit().clear().commit();
        mActivity.deleteDatabase("ms_organizer.db");
    }
    @Test
    public void firstRunTest() {
        SystemClock.sleep(6000); //wait for splashscreen finish
        onView(withId(R.id.image_holo_light)).perform(click());
        onView(allOf(withId(R.id.initialSettingsButtonNext))).perform(click());
        onView(allOf(withId(R.id.nameTextEdit),
                    withParent(withId(R.id.aboutUserFragment)),
                    isDisplayed())).perform(replaceText("testName"));
        onView(allOf(withId(R.id.doctorNameTextEdit),
                    withParent(withId(R.id.aboutUserFragment)),
                    isDisplayed())).perform(replaceText("doctortest"));
        onView(allOf(withId(R.id.nurseNameTextEdit),
                    withParent(withId(R.id.aboutUserFragment)),
                    isDisplayed())).perform(replaceText("nurseTest"));
        onView(allOf(withId(R.id.emailTextEdit),
                    withParent(withId(R.id.aboutUserFragment)),
                    isDisplayed())).perform(replaceText("email@test.com"));
        onView(allOf(withId(R.id.initialSettingsButtonNext))).perform(click());
        onView(allOf(withId(R.id.firstInjectionButtonNext))).perform(scrollTo(),
        click());
        onView(allOf(withId(android.R.id.button1))).perform(click());
        onView(allOf(withId(R.id.dosesEditText),
                    withParent(withId(R.id.aboutUserFragment)),
                    isDisplayed())).perform(replaceText("20"));
        onView(allOf(withId(R.id.notificationDosesEditText),
                    withParent(withId(R.id.aboutUserFragment)),
                    isDisplayed())).perform(replaceText("5"));
        onView(allOf(withId(R.id.button2),
                    withParent(withId(R.id.aboutUserFragment)),
                    isDisplayed())).perform(click());
        onView(allOf(withId(R.id.initialSettingsButtonNext))).perform(scrollTo(),
        click());
    }
}
```

Pokazana wyżej klasa implementuje dodatkową metodę clearAppInfo, oznaczoną adnotacją @Before. Dzięki temu przed uruchomieniem właściwej metody testowej wewnętrz metody clearAppInfo czyszczone są wszystkie wcześniej zapisane dane aplikacji. Gwarantuje to każdorazowe przejście po ekranach początkowych. Wewnątrz głównej metody testowej oznaczonej adnotacją @Test wypełniane są po kolejne wszystkie ekrany początkowe i sprawdzane jest poprawne przejście między nimi.

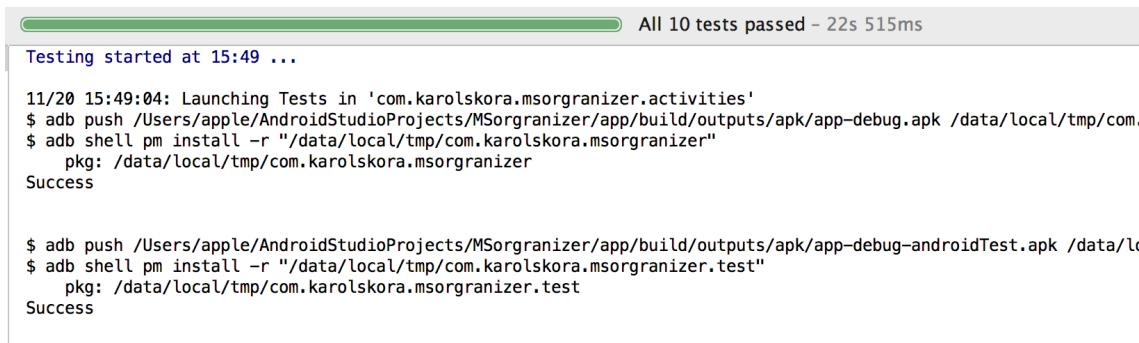
6.2. Uruchomienie testów

Aby uruchomić testy, w pliku konfiguracyjnym build.gradle należy dodatkowo zdefiniować parametr testInstrumentationRunner wskazujący domyślną klasę uruchamiającą. Fragment tego pliku pokazano niżej na listingu 10.

Listing. 11. Fragment pliku konfiguracyjnego build.gradle

```
defaultConfig {  
    applicationId "com.karolskora.msorganizer"  
    minSdkVersion 19  
    targetSdkVersion 23  
    versionCode 1  
    versionName "1.0"  
    testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"  
}
```

Po poprawnym wykonaniu wszystkich testów środowisko Android studio pokazuje na ekranie informację o sukcesie wraz z czasem trwania testów. Poprawne wykonanie testów pokazuje poniższy rysunek 15.



The screenshot shows the terminal output of an Android test run. At the top, it says 'All 10 tests passed - 22s 515ms'. Below that, it shows the start time 'Testing started at 15:49 ...'. The log then lists several adb commands used to push the app and test APKs to the device, followed by the message 'Success'.

```
All 10 tests passed - 22s 515ms  
Testing started at 15:49 ...  
  
11/20 15:49:04: Launching Tests in 'com.karolskora.msorganizer.activities'  
$ adb push /Users/apple/AndroidStudioProjects/MSorganizer/app/build/outputs/apk/app-debug.apk /data/local/tmp/com.  
$ adb shell pm install -r "/data/local/tmp/com.karolskora.msorganizer"  
    pkg: /data/local/tmp/com.karolskora.msorganizer  
Success  
  
$ adb push /Users/apple/AndroidStudioProjects/MSorganizer/app/build/outputs/apk/app-debug-androidTest.apk /data/l  
$ adb shell pm install -r "/data/local/tmp/com.karolskora.msorganizer.test"  
    pkg: /data/local/tmp/com.karolskora.msorganizer.test  
Success
```

Rys. 15. Komunikat o poprawnym wykonaniu testów [opracowanie własne]

6.3. Testy wydajności generowania modelu

Dodatkowo sprawdzono także czy słabsze urządzenia poradzą sobie ze wskazywaniem zastrzyku w formie modelu 3D. W tym celu w klasie ModelRenderer dodano fragment kodu, pokazany na listingu 12, umożliwiający monitorowanie ilości klatek na sekundę w czasie rzeczywistym podczas działania programu.

Listing. 12. Fragment metody onDrawFrame klasy ModelRenderer

```
if (System.currentTimeMillis() - time >= 1000) {  
    Logger.log(fps + "fps");  
    fps = 0;  
    time = System.currentTimeMillis();  
}  
fps++;
```

Dzięki temu po uruchomieniu ekranu wyświetlającego miejsce zastrzyku widać szybkość generowania grafiki podczas działania aplikacji. Na poniższym rysunku 16 pokazano przykładowy zapis ilości klatek na sekundę w terminalu środowiska Android studio.

```
I/jPCT-AE: 61fps  
I/jPCT-AE: 61fps  
I/jPCT-AE: 61fps  
I/jPCT-AE: 60fps  
I/jPCT-AE: 61fps  
I/jPCT-AE: 61fps  
I/jPCT-AE: 61fps  
I/jPCT-AE: 60fps  
I/jPCT-AE: 60fps
```

Rys. 16. Przykładowy zapis ilości klatek na sekundę [opracowanie własne]

Następnie uruchomiono program na kilku urządzeniach testowych w celu sprawdzenia ich wydajności podczas generowania modelu. Zebrane dane przedstawiono poniżej w formie tabeli 1.

Tabela 1. Zestawienie wydajności testowanych telefonów [opracowanie własne]

model	procesor	układ graficzny	średnia ilość klatek na sekundę
LG Optimus L9 II	Qualcomm Snapdragon 400, Zegar procesora: 1,40 GHz, Liczba rdzeni: 2	Adreno 305	55
Samsung Galaxy A5	Qualcomm Snapdragon 410, Zegar procesora: 1,20 GHz, Liczba rdzeni: 4	Adreno 306	57
LG Nexus 4	Qualcomm Snapdragon S4 Pro, Zegar procesora: 1,50 GHz, Liczba rdzeni: 4	Adreno 320	59
Samsung Galaxy s5	Qualcomm Snapdragon 801, Zegar procesora: 2,50 GHz, Liczba rdzeni: 4	Adreno 330	60
Samsung Galaxy s6	Samsung Exynos 7420, Zegar procesora: 2,10 GHz, Liczba rdzeni: 8	Mali-T760 MP8	61

Jak widać na powyższej tabeli każde z testowanych urządzeń poradziło sobie bez problemu z generowaniem modelu 3d widocznego na ekranie wykonywania zastrzyku. Biorąc pod uwagę powyższe zestawienie można przypuszczać, że każde urządzenie, na którym zainstalowano system Android o numerze przynajmniej 4.4 powinno działać wydajnie podczas pracy prezentowanej aplikacji.

7. Podsumowanie

W ostatnich latach powstało wiele programów dedykowanych na urządzenia przenośne, wspierających osoby chore lub niepełnosprawne. Grupą osób, która do tej pory nie była zauważana przez twórców takiego oprogramowania, byli chorzy na stwardnienie rozsiane. Dlatego w pracy tej za cel postawiono sobie stworzenie oprogramowania na smartfony, które wspomagałyby tą grupę osób.

Aplikacja powstała w celu ułatwienia osobom chorym na stwardnienie rozsiane codziennego zmagania się z chorobą. Dzięki programowi użytkownik informowany jest o zbliżającym się zastrzyku oraz wskazywane jest dokładne miejsce kolejnej dawki. Dzięki temu osoba korzystająca z tego ułatwienia nie musi pamiętać o godzinie aplikacji leku oraz o miejscu, w którym będzie ją wykonywała. Program ułatwia także prowadzenie historii zastrzyków, umożliwia zapis objawów ubocznych oraz wspomaga zarządzanie zapasami leku. Podczas prac nad programem powstał także wygodny oraz czytelny dla osób mających problemy ze wzrokiem interfejs graficzny aplikacji. Wszystkie główne założenia stawiane oprogramowaniu w fazie projektowania zostały spełnione, podczas prac udało się także rozbudować projekt o dodatkowe możliwości, takie jak: zmiana wyglądu aplikacji na wysoki kontrast czy uruchomienie pomocy, pokazującej główne elementy programu.

Po zakończeniu prac aplikacja trafiła do osoby chorej na stwardnienie rozsiane w celu dalszego testowania i ulepszania. W momencie pisania pracy program jest cały czas używany przez zainteresowaną osobę oraz planowane są dalsze udoskonalenia i rozbudowa całego projektu. W przyszłości powstać ma dodatkowo strona internetowa, na której użytkownicy będą mieli dostęp do całej historii swoich zastrzyków. Historia zastrzyków pacjenta oraz jego informacje personalne zostaną przeniesione z lokalnej bazy danych na telefonie komórkowym do bazy danych po stronie serwera. Dzięki temu zmniejszy się ryzyko utraty danych o poprzednich zastrzykach. Powstanie strony internetowej umożliwi także dodatkową formę kontaktu z lekarzem prowadzącym osobę chorą poprzez udostępnienie możliwości przeglądania raportów z zastrzyków online.

8. Literatura

- [1]. Android Developer. [Online]. Dostępny w Internecie:
<http://www.developer.android.com>
- [2]. Dawn Griffiths, David Griffiths, *Head First Android Development A Brain-Friendly Guide*, wydawnictwo O'Reilly Media, 2016
- [3]. Andrzej Stasiewicz, *Android Studio. Podstawy tworzenia aplikacji*, wydawnictwo Helion, 2015
- [4]. Oficjalna dokumentacja biblioteki JPCT [Online]. Dostępny w Internecie:
<http://www.jpct.net/jpct-ae/doc/>
- [5]. Dokumentacja biblioteki Material Drawer [Online]. Dostępny w Internecie:
<https://github.com/mikepenz/MaterialDrawer>
- [6]. Oficjalna strona biblioteki iText [Online]. Dostępny w Internecie:
<http://itextpdf.com/>
- [7]. Oficjalna strona biblioteki ORMLite [Online]. Dostępny w Internecie:
<http://ormlite.com/>
- [8]. Dokumentacja biblioteki Espresso [Online]. Dostępny w Internecie:
<https://google.github.io/android-testing-support-library/docs/espresso/>
- [9]. Dokumentacja oraz opis biblioteki MPAndroidChart [Online]. Dostępny w Internecie: <https://github.com/PhilJay/MPAndroidChart/wiki>
- [10]. Material design guidelines [Online]. Dostępny w Internecie:
<https://material.io/guidelines/>