|  | **Wydział Finansów i Zarządzania Kierunek: Informatyka** |
| --- | --- |

**Jan Wieprow**

(numer albumu: 81597)

**Aplikacja mobilna pozwalająca na zapisanie informacji o nowo poznanych ludzi w celu przypomnienia sobie o nich przy ponownym spotkaniu**

Inżynierska praca projektowa

**Opiekun merytoryczny:**

**mgr inż. Barczyński Wojciech**

Wrocław 2025

Spis treści

[1.](#_heading=h.1fob9te) Wstęp 4

[1.2.](#_heading=h.3znysh7) Wprowadzenie, cel, zakres pracy 4

[2.](#_heading=h.2et92p0) Analiza konkurencyjności 4

[3.](#_heading=h.tyjcwt) Analiza SWOT 4

[3.1.](#_heading=h.3dy6vkm) Strengths (Mocne strony) 5

[3.2.](#_heading=h.1t3h5sf) Weaknesses (słabe strony) 5

[3.3.](#_heading=h.4d34og8) Opportunities (szanse) 5

[3.4.](#_heading=h.2s8eyo1) Threats 6

[4.](#_heading=h.17dp8vu) Założenia projektowe 6

[4.1.](#_heading=h.3rdcrjn) Wymagania funkcjonalne według zasady MoSCoW 6

[4.2.](#_heading=h.26in1rg) Wymagania niefunkcjonalne według zasady MoSCoW 7

[4.3.](#_heading=h.lnxbz9) Diagramy UML 7

[4.4.](#_heading=h.35nkun2) UX - prototypowanie 7

[5.](#_heading=h.1ksv4uv) Technologie opisać też do czego służyły w mojej aplikacji 7

[5.1.](#_heading=h.44sinio) Android 7

[5.2.](#_heading=h.2jxsxqh) Java jest to obiektowy język programowania. Może być używany do tworzenia niezależnych aplikacji a także działających na serwerze WWW. 7

[5.3.](#_heading=h.z337ya) Android Studio 7

[5.4.](#_heading=h.3j2qqm3) Firebase - napisać o bazie danych oraz logowaniu 8

[5.5.](#_heading=h.1y810tw) GitHub 8

[5.6.](#_heading=h.4i7ojhp) Git 9

[6.](#_heading=h.2xcytpi) Implementacja i wdrożenie 9

[6.1.](#_heading=h.1ci93xb) Opis projektu 9

[6.2.](#_heading=h.2bn6wsx) Ikona aplikacji, 13

[6.3.](#_heading=h.qsh70q) Logowanie do aplikacji. 13

[6.4.](#_heading=h.3as4poj) Rejestracja do aplikacji 18

[6.5.](#_heading=h.1pxezwc) Wylogowanie z aplikacji 21

[6.6.](#_heading=h.49x2ik5) Tworzenie nowych znajomych do listy znajomych 23

[6.7.](#_heading=h.2p2csry) Mowa na tekst w tworzeniu nowego znajomego 27

[6.8.](#_heading=h.147n2zr) Lista znajomych 30

[6.9.](#_heading=h.3o7alnk) Wyszukiwarka w liście przyjaciół 34

[6.10.](#_heading=h.23ckvvd) Usuwanie wierszy w liście znajomych 36

[6.11.](#_heading=h.ihv636) Mapa z pinezkami dla przyjaciół 38

[6.12.](#_heading=h.32hioqz) Ustawienia 41

[6.13.](#_heading=h.41mghml) Zmiana trybu na dzienny lub nocny 46

[7.](#_heading=h.2grqrue) Testy i wdrożenie 50

[7.1.](#_heading=h.vx1227) Testy jednostkowe 50

[7.2.](#_heading=h.3fwokq0) Testy UI 50

[7.3.](#_heading=h.1v1yuxt) Bezpieczeństwo i dostępność aplikacji 51

[7.4.](#_heading=h.4f1mdlm) Metody dystrybucji aplikacji - napisać co najmniej jedną stronę 51

[8.](#_heading=h.2u6wntf) Podsumowanie i wnioski 51

[8.1.](#_heading=h.19c6y18) Wnioski 51

[8.2.](#_heading=h.3tbugp1) Przyszłość aplikacji (następne kroki) 51

[Przyszłość aplikacji (następne kroki) 51](#_heading=h.28h4qwu)

[Bibliografia 53](#_heading=h.nmf14n)

# Wstęp

# Wprowadzenie, cel, zakres pracy

W aktualnym świecie wraz z rozwojem technologii oraz trendami, życie ludzkie zaczyna płynąć coraz szybciej. Rzadko skupiamy się na obecnej chwili przez co coraz trudniej zapamiętać konkretne wydarzenia czy ludzi. Szczególnie trudne może dla niektórych dokładne zapamiętanie czyjegoś imienia czy miejsce poznania, przy jednoczesnej świadomości, że skądś kojarzy się tą osobę. Taka sytuacja często jest niezwykle frustrująca czy nawet zawstydzająca. W celu rozwiązania właśnie takich problemów powstała aplikacja Humanon. Jej celem jest przełożenie potrzeby zapamiętania niezliczonej ilości osób właśnie na tą aplikację. Pozwoli ona na zapisanie informacji o nowo poznanych ludziach takich jak np. imię, miejsce spotkania, charakter, wygląd czy zdjęcie. Daje to możliwość dyskretnego sprawdzenia spotkanej osoby czy już wcześniej się ją poznało. Istnieje również biznesowe zastosowanie dla Humanon. Możliwe jest zapisywanie nowo poznanych partnerów biznesowych w celu późniejszego skontaktowania się lub po prostu żeby ich zapamiętać.

Celem projektu jest stworzenie oraz opisanie aplikacji Humanon. Aplikacja została stworzona na system Android ze względu duży rynek, dostępność narzędzi do jej tworzenia oraz doświadczenie w użyciu Android studio.

Praca składa się ze wstępu, ośmiu rozdziałów oraz zakończenia. W pierwszej części pracy opisane zostały czym ma być i jakie cele ma tworzona aplikacja. Następnie opisano jej konkurencyjność, zalety, wady, sposób działania, co ma zawierać oraz użyte technologie. Dalej opisano funkcjonalności aplikacji i w jaki sposób działają. Przed końcem napisano jakich testów użyto i ich przebieg. Na koniec napisano podsumowanie, jakie wnioski zostały wyciągnięte w trakcie tworzenia tej pracy oraz plany w jaki sposób aplikacja powinna się rozwijać.

Praca została napisana o literaturę przedmiotu oraz portale internetowe związane z omawianą tematyką w oparciu o literaturę przedmiotu otar portale internetowe związane z omawianą tematyką. Część badawcza pracy projektowej napisana została na podstawie stworzonej przez Autora wersji testowej aplikacji Humano.

# Analiza konkurencyjności

# Analiza SWOT

Analiza SWOT jest to narzędzie analizy strategicznej przedsiębiorstwa. Jest to najpopularniejsza heurystyczna technika analityczna służąca do porządkowania informacji. SWOT ma za zadanie pomóc osobie analizującej poukładać oraz uporządkować myślenie tak, aby informacje były przedstawione w sposób czytelny oraz w łatwy sposób można było wyciągać z nich wnioski.

Nazwa SWOT jest akronimem słów ,,Strenghts” (mocne strony przedsiębiorstwa), „Weakness” (słabe strony przedsiębiorstwa), „Opportunities” (szanse w otoczeniu przedsiębiorstwa), „Threats” (zagrożenia w otoczeniu przedsiębiorstwa)”. Będą one szczegółowo opisane w podpunktach poniżej. Dzięki tej analizie jako przedsiębiorstwo można odpowiedzieć na pytania, jaka jest jego pozycja na rynku, jakie są jego cele czy za pomocą jakich narzędzi będą je realizować w przyszłości.[[1]](#footnote-0)

# Strengths (Mocne strony)

Mocne strony przedsiębiorstwa to czynniki, które wyróżniają jednostkę od konkurencyjnych firm, które działają w podobnej branży czy obszarze.

W stworzonej aplikacji takimi czynnikami mogą być:

* prostota użytkowania,
* możliwe łatwe dodanie nowych funkcjonalności,
* niski koszt produkcji,
* nacisk na ładny design,
* pomoc w zapamiętaniu nowo poznanych osób.

# Weaknesses (słabe strony)

Słabe strony przedsiębiorstwa to takie czynniki, które wynikają z niewystarczających umiejętności i ograniczonych zasobów. W stworzonej aplikacji takimi czynnikami mogą być:

* mała ilość funkcjonalności,
* brak jasno wytyczonej strategii,
* brak badań rynku,
* brak marketingu,
* trudność w dotarciu do nowych klientów.

# Opportunities (szanse)

Szanse to wszystkie czynniki zewnętrzne jeżeli dobrze wykorzystane to staną się bodźcem rozwoju przedsiębiorstwa.

W opisywanej aplikacji takimi czynnikami mogą być:

* utrata zainteresowania konkurencyjnymi platformami,
* tendencje społeczne do krótkiego utrzymania uwagi,
* ciągły wzrost użycia telefonów nad komputerami,
* wzięcie pod uwagę ludzi, dla których taka aplikacja byłaby szczególnie użyteczna.

# Threats

Zagrożenia przedsiębiorstwa to takie czynniki, które mogą pojawić się w otoczeniu organizacji i są postrzegane jak utrudnienia, niebezpieczeństwa oraz bariery w działalności organizacji. W opisywanej aplikacji takimi czynnikami mogą być:

* istnienie podobnych w założeniu aplikacji,
* brak siły przebicia,
* niewystarczająca atrakcyjność aplikacji,
* niewystarczająca ilość testów,
* niewystarczające zauważenie przez ilość wypuszczanych aplikacji.

# Założenia projektowe

# Wymagania funkcjonalne według zasady MoSCoW

Metoda MoSCoW to technika zarządzania priorytetami w projektach, szczególnie popularna w zarządzaniu projektami IT. Nazwa tej metody jest akronimem kategorii priorytetów. Polega na podzieleniu wymagań projektu na cztery kategorie:

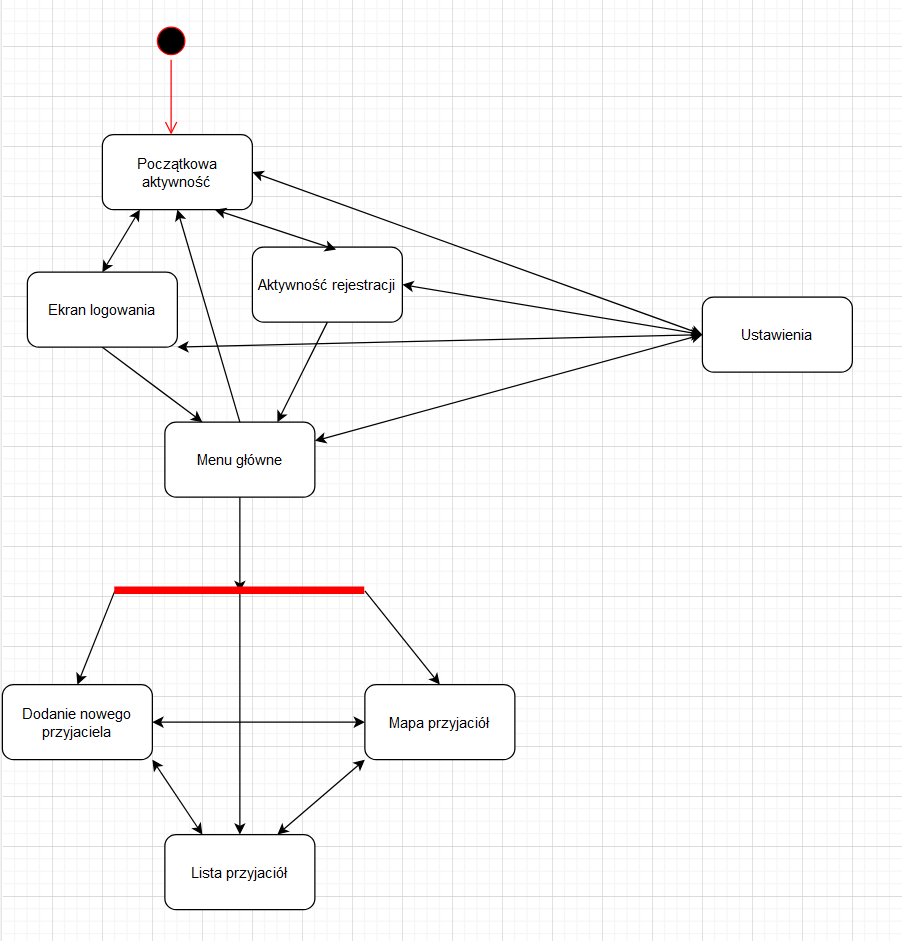
* „Must have”,
* „Should have”,
* „Could have”
* „Won’t have this time”.

Must have – określa wymagania, które są krytyczne dla produktu. Jeżeli nie zostaną one zrealizowane, cały projekt zostanie zablokowany. Z reguły rozpoczęcie produkcji zależy całkowicie od zdefiniowania must have przy użyciu wskaźników takich jak „wymagane do uruchomienia”, „wymagane dla bezpieczeństwa” itp. Should have – ten rodzaj wymagań ma drugorzędne znaczenie. Wymagania typu should have nie mają wpływu na uruchomienie i tradycyjnie są uważane za ważne, ale nie kluczowe. Jeśli budujesz produkt, będzie on nadal nadawał się do użytku, nawet jeśli te wymagania nie zostaną spełnione. Could have – wymagania są pożądane, ale mniej ważne niż dwa poprzednie. To kryterium ma mniejszy wpływ na realizację projektu niż kategoria should have, więc może zostać pominięte. Zadania could have są realizowane, jeśli projekt nie jest mocno ograniczony w czasie. Możemy nazwać je dodatkowymi opcjami produktu. Won’t have – można spotkać się z wymaganiami nazywanymi „nie będziesz mieć”; te wymagania określają najniższy priorytet dla zadań, które nie są możliwe do realizacji z określonym budżetem i terminem. Won’t have nie oznacza całkowitego odrzucenia czegoś, przewiduje ono ponowne wprowadzenie na korzystnych warunkach w przyszłości.[[2]](#footnote-1)

# Wymagania niefunkcjonalne według zasady MoSCoW

# Diagramy UML

UML to notacja umożliwiająca prezentację systemu w formie graficznej, za pomocą różnorodnych diagramów. Modele stworzone w języku UML przedstawiają system w sposób hierarchiczny – od ogółu do szczegółu. Pozwala to na analizę modelu systemu z dowolnie wybranym poziomem szczegółowości, dostosowanym do aktualnych potrzeb.[[3]](#footnote-2)



# UX - prototypowanie

# Technologie opisać też do czego służyły w mojej aplikacji

# Android

# Java jest to obiektowy język programowania. Może być używany do tworzenia niezależnych aplikacji a także działających na serwerze WWW.[[4]](#footnote-3)

# Android Studio

Android Studio to oficjalne zintegrowane środowisko programistyczne (IDE) do tworzenia aplikacji na Androida. Android Studio, oparty na zaawansowanym edytorze kodu i narzędziach dla programistów z platformy [IntelliJ IDEA](https://www.jetbrains.com/idea/), oferuje jeszcze więcej funkcji zwiększających produktywność podczas tworzenia aplikacji na Androida. Oto niektóre z nich:[[5]](#footnote-4)

* elastyczny system kompilacji oparty na Gradle;
* szybki i bogaty w funkcje emulator,
* ujednolicone środowisko, w którym możesz tworzyć aplikacje na wszystkie urządzenia z Androidem, edytować w czasie rzeczywistym komponenty na emulatorach i na fizycznych urządzeniach;
* szablony kodu i integrację z GitHubem, które ułatwiają tworzenie typowych funkcji aplikacji i importowanie przykładowego kodu; kompleksowe narzędzia i ramy testowania;
* narzędzia lint do wykrywania problemów z wydajnością, użytecznością, zgodnością z wersją i innymi problemami;
* Obsługa C++ i NDK,
* Wbudowane wsparcie dla [Google Cloud Platform](https://cloud.google.com/tools/android-studio/docs?hl=pl), które ułatwia integrację Google Cloud Messaging i App Engine.

# Firebase - napisać o bazie danych oraz logowaniu

Firebase to kompleksowy zestaw narzędzi i usług oferowanych jako platforma Backend-as-a-Service (BaaS), umożliwiający programistom łatwe tworzenie, uruchamianie i rozbudowę aplikacji mobilnych i internetowych. Oferuje bazę danych w czasie rzeczywistym, uwierzytelnianie, przechowywanie, hosting i inne funkcje, a wszystko to zarządzane z jednej platformy. W swoim rdzeniu, Firebase zapewnia bazę danych w czasie rzeczywistym, która synchronizuje dane we wszystkich podłączonych urządzeniach w czasie rzeczywistym. Baza danych wykorzystuje model danych zorientowany na dokumenty [NoSQL](https://appmaster.io/pl/blog/bazy-danych-sql-vs-nosql), który pozwala programistom przechowywać dane w sposób elastyczny i skalowalny. Dane są przechowywane w formacie [JSON](https://appmaster.io/pl/blog/json-vs-xml-pl), a baza danych obsługuje transakcje atomowe i powiadomienia o zdarzeniach w czasie rzeczywistym.[[6]](#footnote-5)

# GitHub

GitHub jest to platforma w chmurze, która używa open-source rozwiązania kontroli wersji Git. Pozwala ona na przechowywanie kodu dzięki czemu możliwe jest śledzić zmiany jakie nastąpiły w projekcie, wydzielać zadania, recenzować, udostępniać czy pracować wspólnie nad kodem. Jest to niezwykle przydatna strona internetowa, która niezwykle często przez programistów. Właśnie przy jej użyciu pisana jest aplikacja opisywana w tej pracy inżynierskiej.[[7]](#footnote-6)

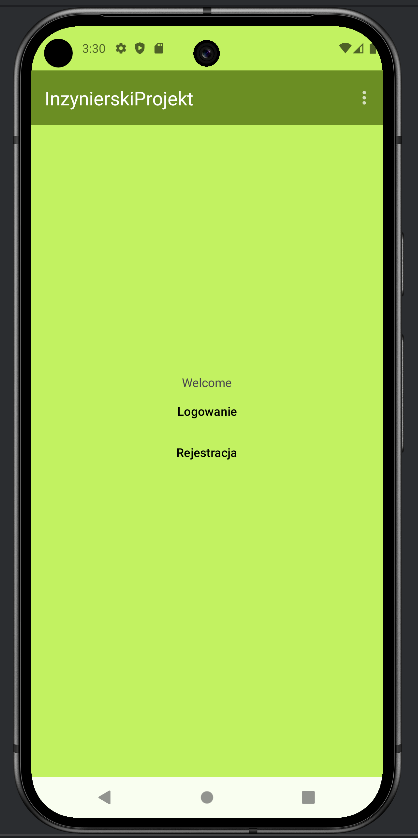
# Git

Git jest to rozwiązanie open-source pozwalające na śledzenie zmian w plikach czyli kontrole wersji albo VCS. Jest to niezwykle użyteczna technologia przy pracy wielu osób nad jednym plikiem lub plikami. Dzięki temu deweloper może pracować na dowolnej wersji projektu oraz odzyskać poprzednią wersje plików. Może on również przeglądać kod w celu sprawdzeniu kiedy i jakie zmiany zostały dokonane, kto ich dokonał oraz przy dobrych praktykach dlaczego zostały one dokonane.[[8]](#footnote-7)

# Implementacja i wdrożenie

# Opis projektu

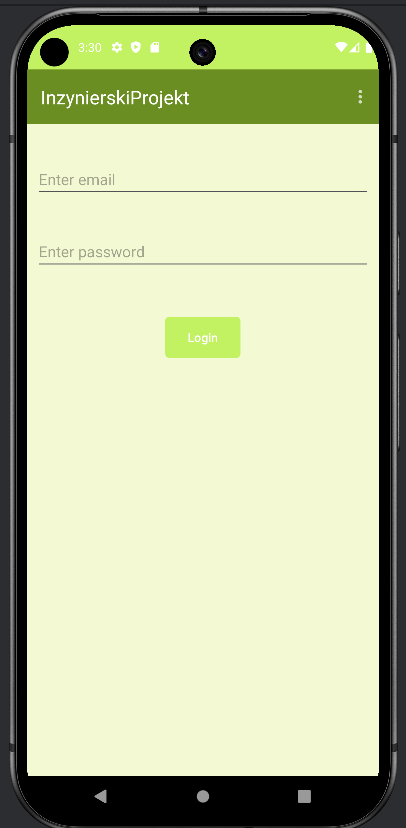
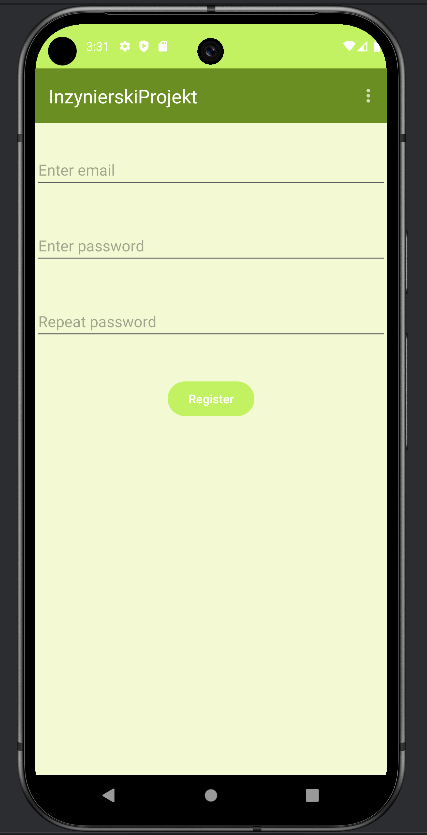
Aplikacja pozwala na zapisywanie ludzi w formie prostego opisu oraz umieszczeniu ich na mapie gdzie się ich spotkało. Do aplikacji konieczne jest rejestracja lub logowanie. Użytkownik ma do wyboru ustawienia, w których jest możliwa zmiana motywu kolorystycznego z jasnego na ciemny i na odwrót. Widok,w którym możliwe jest tworzenie nowych przyjaciół znajduje się na dolnym pasku menu po lewej stronie. Po wpisaniu danych co najmniej jednego pola można wcisnąć przycisk zapisz co zachowa dane i pozwoli je wyświetlić w liście przyjaciół. Znajduje się ona w menu na dole na środku. W tym widoku wyświetlana jest lista zapisanych przyjaciół, możliwe jest wyszukanie przyjaciela po wpisanych danych. W tym celu należy wpisać odpowiednią frazę w pole wyszukiwania na górze ekranu. W celu usunięcia pozycji z danymi danego przyjaciela należy kliknąć przycisk usuń. W ostatnim widoku dostępnym na dole ekranu po prawej stronie widoczna jest mapa, na której zostaną wyświetlone pinezki z imionami dla przyjaciół, dla których uzupełniło się lokalizację.



Rys. 6.1. Widok ekranu

Źródło: opracowanie własne.

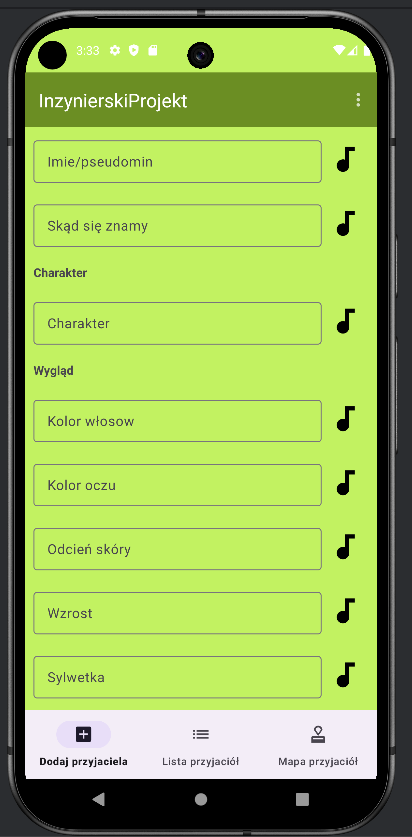
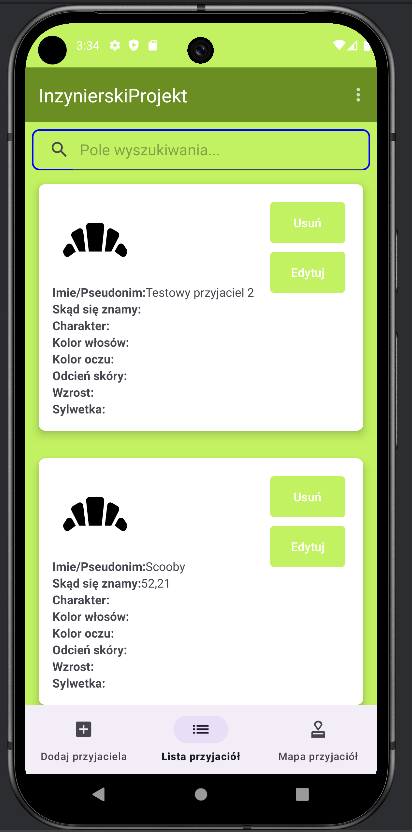
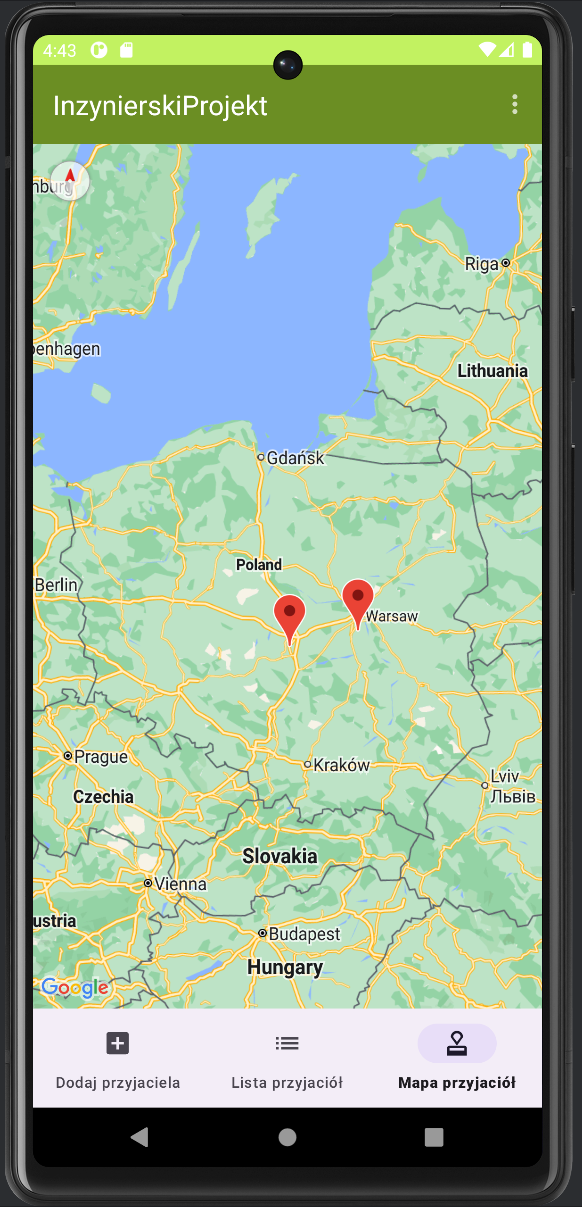
Główna aktywność Posiada możliwość przejścia do ekranu logowania, rejestracji oraz ustawień. Jak nazwy wskazują dzięki tym ekranom użytkownik może się zalogować, zarejestrować lub zmienić ustawienia.

Rys. 6.2.Widok ekranu

Źródło: opracowanie własne.

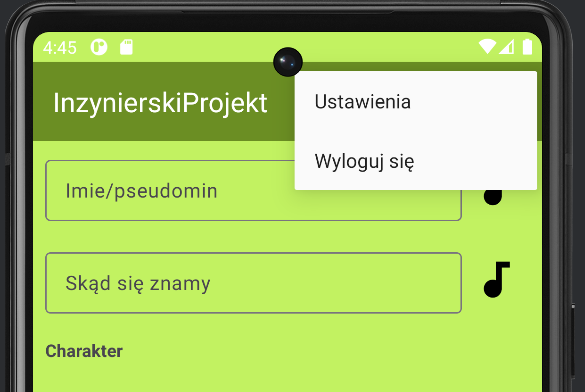
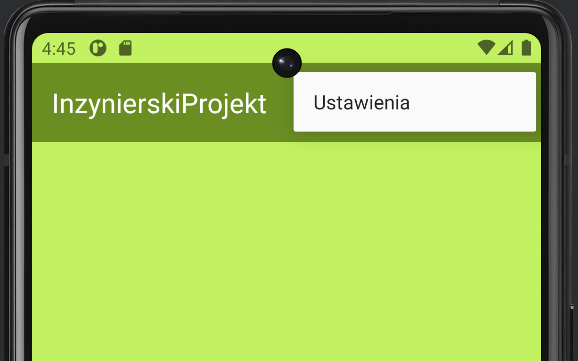
Aktywność logowania pozwala na przejście do głównego menu po poprawnym podaniu danych oraz pozwala na przejście do ustawień.

Rys. 6.3.Widok ekranu

Źródło: opracowanie własne.

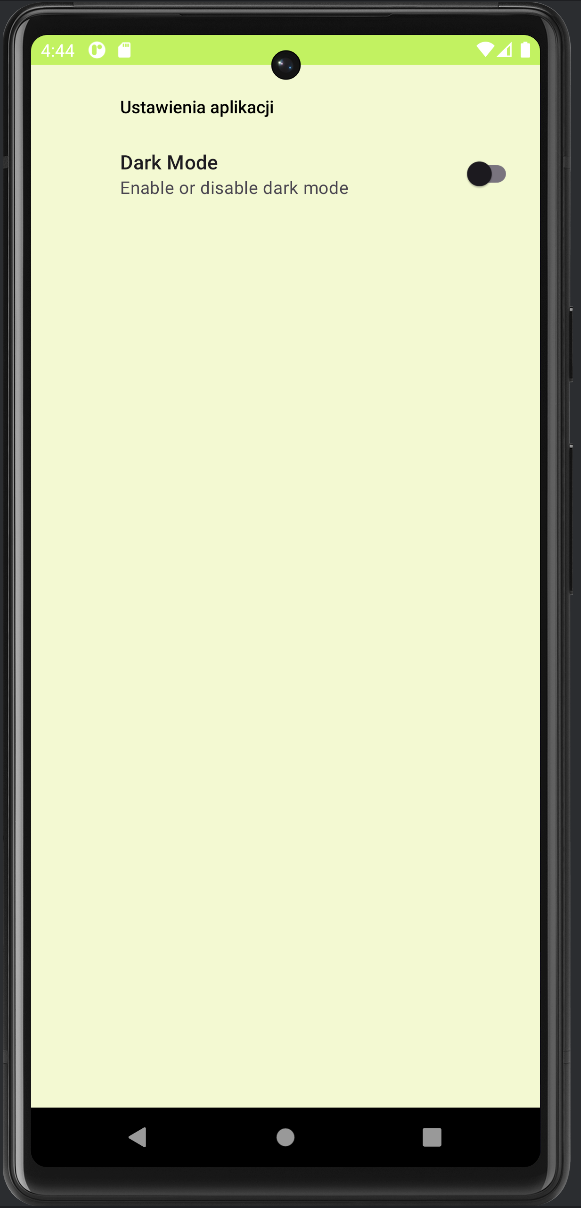
Widok głównego menu pozwala na dostęp do fragmentów Dodaj Przyjaciela, Lista przyjaciół oraz Mapa przyjaciół. Dodatkowo jest możliwość przejścia do ustawień oraz wylogowania się co przekierowują na aktywność wyboru logowania lub rejestracji.

Rys. 6.4. Widok ekranu

Źródło: opracowanie własne.

Dostęp do ustawień znajduje się w prawym górnym rogu gdzie jednocześnie jest opcja wylogowania się.



Rys.6.5. Widok ekranu

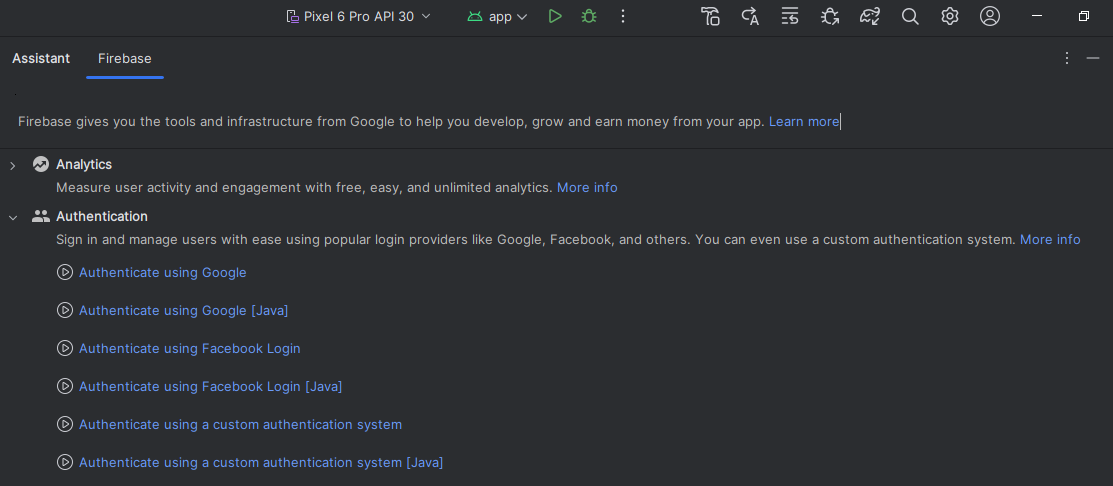
Źródło: opracowanie własne.

Ustawienia oferują zmianę trybu z jasnego na ciemny i na odwrót.

# Ikona aplikacji,

# Logowanie do aplikacji.

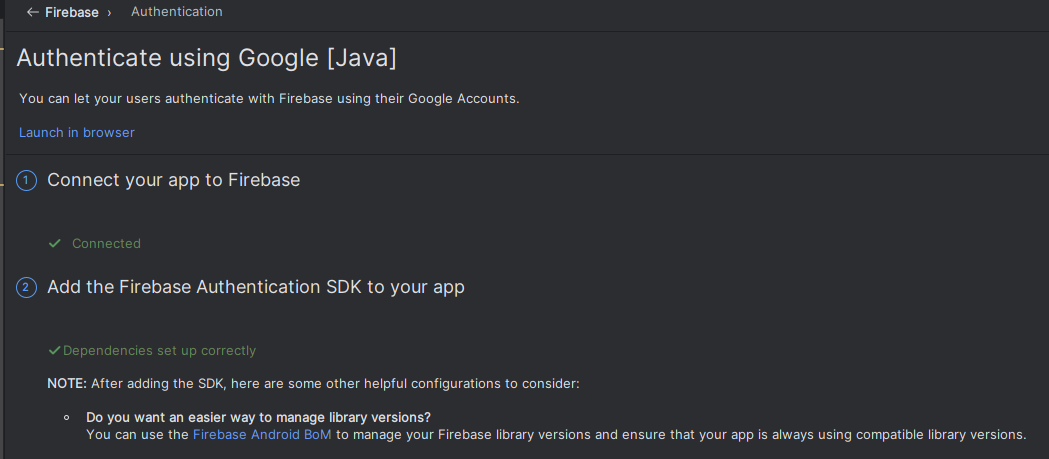
Żeby używać FireBase w aplikacji należy najpierw połączyć Android Studio ze stroną Fire Base na której znajduje się nasz projekt.



Rys. 6.6. Widok kodu

Źródło: opracowanie własne.

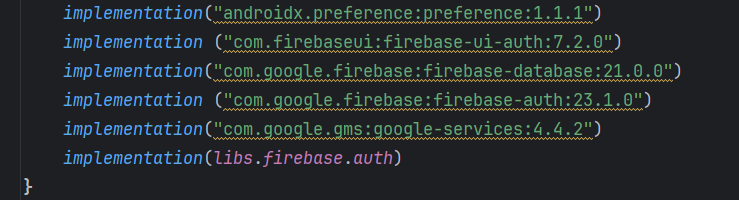
Na załączonym obrazku przedstawione jest okno FireBase w Android Studio. W celu w połączenia projektu ze stroną Fire Base i umożliwić logowanie przez Google należy wybrać Authenticate using Google [Java].



Rys. 6.7. Widok kodu

Źródło: opracowanie własne.

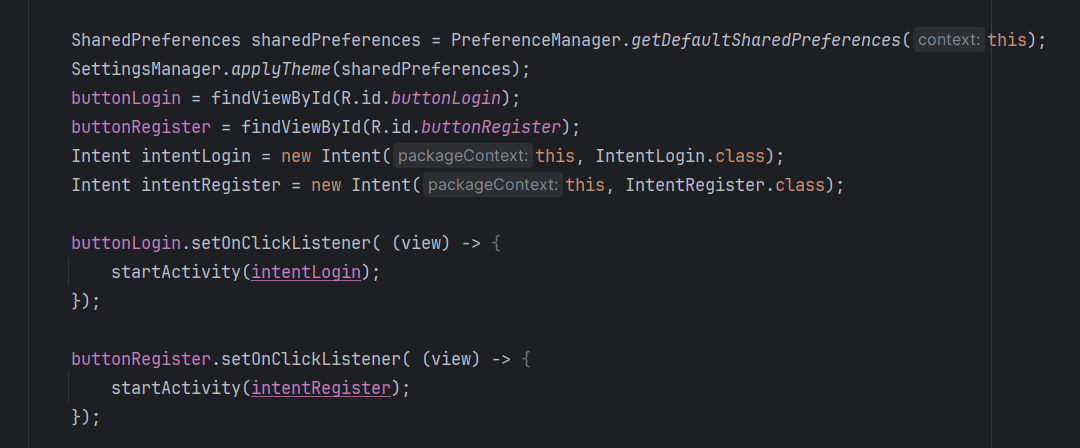
Na obrazku widać, że projekt został już połączony z Fire Base. Przed połączeniem, pod Connect your app to Firebase jest możliwość kliknięcia przycisku, który przeniesie użytkownika na stronę Fire Base, na której będzie możliwe połączenie projektu z Fire Base. Następnie należy kliknąć przycisk, który normalnie pojawia się pod Add Firebase Authentication SDK to your app. Powinny zostać dodane odpowiednie zależności, które umożliwią odwoływanie się do instancji Fire Base.



Rys. 6.8. Widok kodu

Źródło: opracowanie własne.

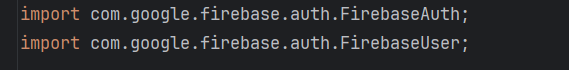
Na obrazku powyżej są przedstawione niezbędne zależności.



Rys. 6.9. Widok kodu

Źródło: opracowanie własne.

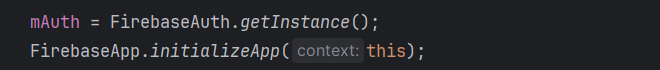
Ze względu na to, że użytkownik musi wybrać czy chce się zalogować czy zarejestrować jest możliwość kliknięcia guzików, które przenoszą do odpowiednich intencji, które pozwalają kolejno się zalogować lub zarejestrować. Po przeniesieniu użytkownika do intencji z logowaniem, udostępnione są mu dwa pola, w których może kolejno wpisać email oraz hasło następnie przycisk, który zatwierdza logowanie.



Rys. 6.10. Widok kodu

Źródło: opracowanie własne.

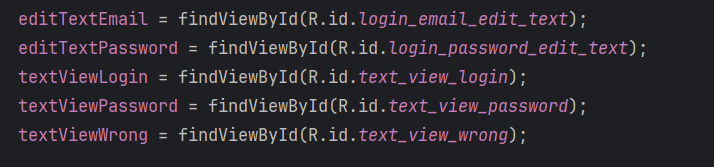
W celu poprawnego działania zaimportowano moduły FirebaseAuth oraz FirebaseUser.



Rys. 6.11. Widok kodu

Źródło: opracowanie własne.

Zainicjalizowano usługę Fire Base w tej intencji co pozwala na dostęp do usługi autoryzującej logowanie.



Rys. 6.12. Widok kodu

Źródło: opracowanie własne.

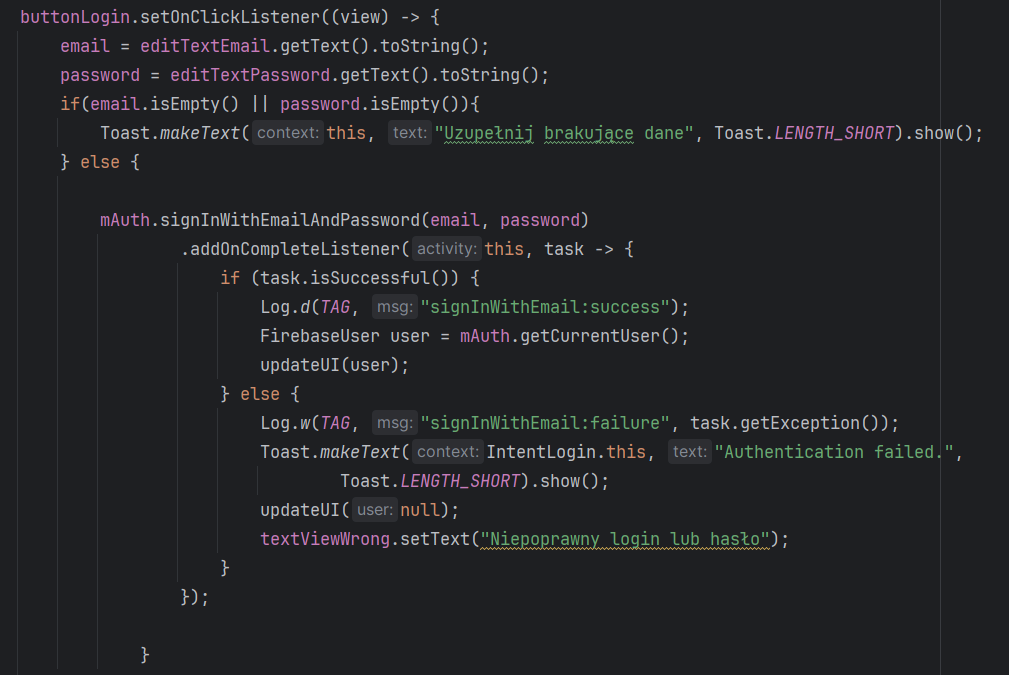
Zainicjalizowano elementy widoku. Pierwsze dwa pozwalają na wpisanie loginu oraz hasła przez użytkownika a następnie sczytanie ich. Następnie trzy pozwalają na powiadomienie użytkownika kolejno o tym, że brakuje loginu, hasła i dane nie są poprawne.



Rys. 6.13. Widok kodu

Źródło: opracowanie własne.

Zainicjalizowano element widoku, który pozwala po wciśnięciu na pobranie danych logowania z odpowiednich pól a następnie przesłanie ich do instancji Fire Base Authorizantion co pozwoli je zweryfikować.



Rys. 6.14. Widok kodu

Źródło: opracowanie własne.

Zainicjalizowano funkcję przycisku, która pozwala na wywołanie kodu po wciśnięciu go. Przedstawiony kod pobiera kolejno wartości dla loginu i hasła po czym zapisuje je jako zmienne. Następnie sprawdza czy, któreś pole nie jest puste a jeśli jest to wyświetla informacje o jego braku co jest wykonywane przez logikę podaną poniżej na obrazku.



Rys. 6.15. Widok kodu

Źródło: opracowanie własne.

Jeśli pola nie są puste zmienne przechowujące login i hasło przesyłane są do instancji mAuth, które z kolei przesyła dane na serwer w celu ich weryfikacji i zwraca informację czy logowanie się powiodło. W przypadku niepowodzenia logowania użytkownik jest informowany o nieprawidłowym loginie lub haśle. W celach bezpieczeństwa nie informuje się użytkownika czy ktoś o takim emailu już ma konto. W sytuacji gdy mAuth zwróci informację o poprawnym zalogowaniu tworzona jest instancja user, która aktualizuje intencje i przenosi do głównego menu.



Rys. 6.16. Widok kodu

Źródło: opracowanie własne.

Przedstawiona funkcja, jeżeli udało się przypisać użytkownika do instancji user, kończy prace aktualnej intencji i aktywuje nową z głównym menu. W głównym menu również zainicjalizowano odpowiednie importu.



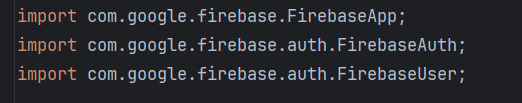
Rys. 6.17. Widok kodu

Źródło: opracowanie własne.

Tworzona jest instancja Fire Base, która pozwala korzystać jej funkcji. Pobierany jest aktualnie zalogowany użytkownik. Jeżeli logowanie powiodło się pomyślnie to nie wywoła się dodatkowa logika ale w sytuacji przeciwnej intencja zostanie zamknięta. To samo stanie się jeśli użytkownik wyjdzie z aplikacji i wróci do niej. Dodatkowo jeżeli, w którymś momencie użytkownik zostanie wylogowany to główne menu zostanie zamknięte.

# Rejestracja do aplikacji

Tak jak w sytuacji, która została przedstawiona w poprzednim podpunkcie gdy użytkownik może wybrać czy chce się zarejestrować czy zalogować, wybierzecie rejestracje zostanie przeniesiony do odpowiedniej intencji, która obsługuje to działanie.



Rys. 6.18. Widok kodu

Źródło: opracowanie własne.

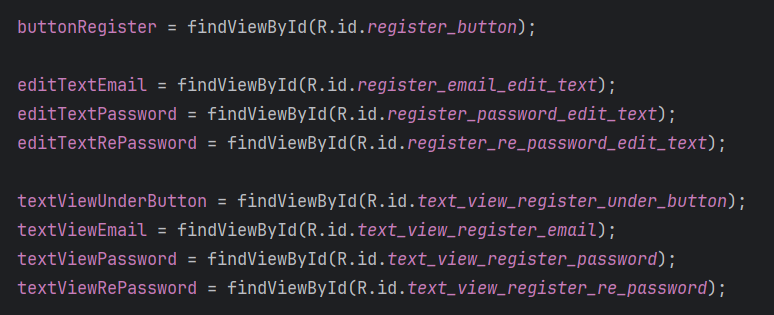
Tak samo jak przy logowaniu zainicjalizowaniu odpowiednie importy.



Rys. 6.19. Widok kodu

Źródło: opracowanie własne.

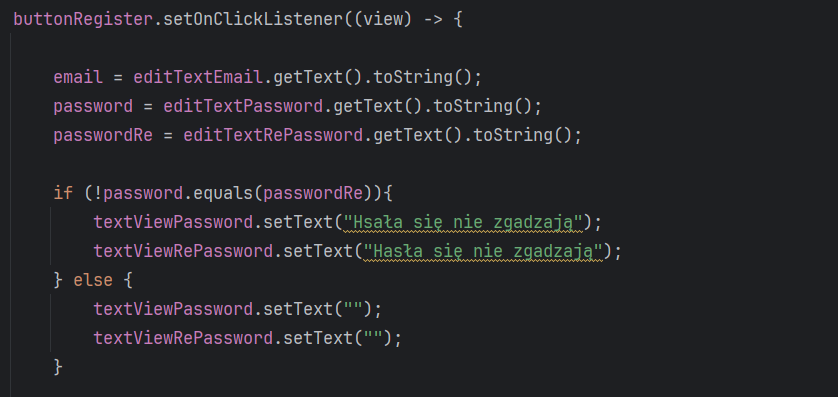
Zainicjalizowano instancje FirebaseAuth oraz zainicjalizowano samo Fire Base.



Rys. 6.20. Widok kodu

Źródło: opracowanie własne.

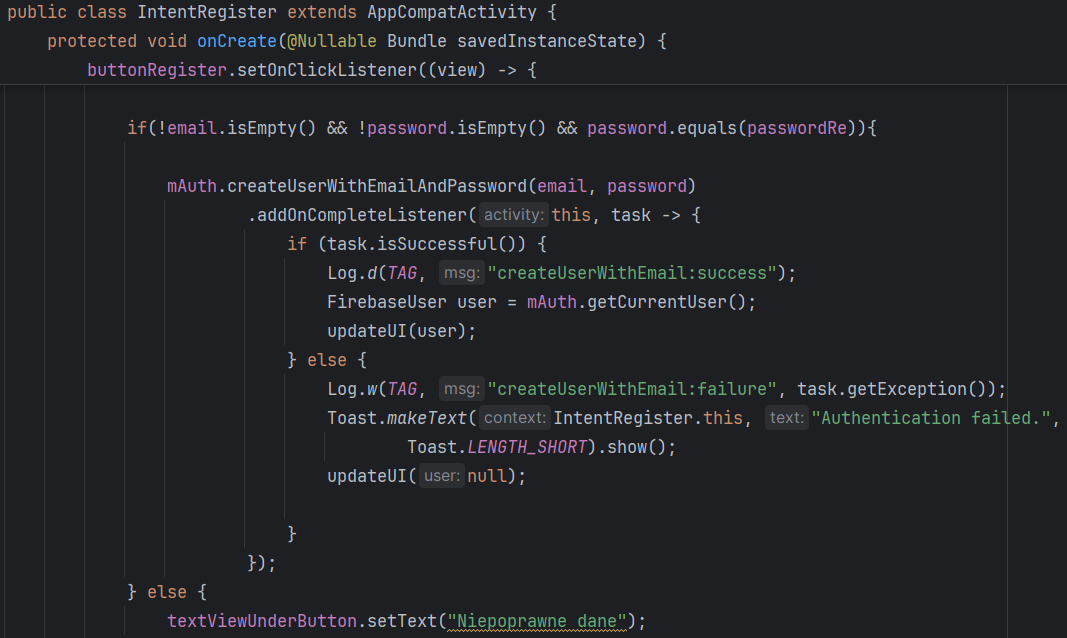
Inicjalizowanie widoków pozwalające na kliknięcie przycisku, wpisanie przez użytkownika emaila, hasła, powtórzonego hasła, poinformowanie użytkownika o niepoprawnych danych, mailu, haśle lub powtórzonym haśle.



Rys. 6.21. Widok kodu

Źródło: opracowanie własne.

Po wciśnięciu przycisku aktywowana jest logika, która pobiera dane z odpowiednich pól i zapisuje je jako zmienne. Jeśli hasła się nie zgadzają użytkownik zostanie o tym poinformowany poprzez pokazanie TextView z odpowiednią informacją. W przypadku jeśli hasła są takie same to tekst w polach na dane zostanie ustawiony na pusty.



Rys. 6.22. Widok kodu

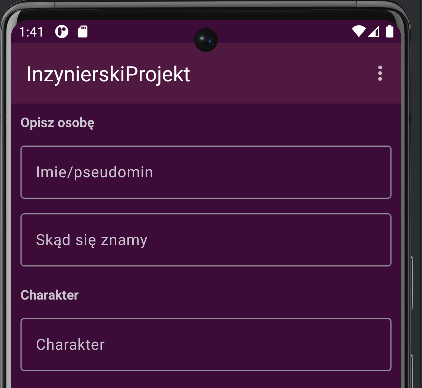
Źródło: opracowanie własne.

Po weryfikacji hasła jeśli są one takie same a login nie jest pusty to te dane zostaną przekazane do instancji FireBase, która prześle je dalej na serwer Fire Base. Jeśli hasła są puste, nie są takie same lub login jest pusty to wyświetli się komunikat w TextView o niepoprawnych danych. W sytuacji uzupełnienia danych ale niespełnienia wymogów odpowiedniej długości znaków, wystarczające ilości cyfr i znaków specjalnych lub email nie będzie poprawny to użytkownik nie zostanie zarejestrowany.

W sytuacji uzupełnienia poprawnych danych użytkownik zostanie zarejestrowany i przeniesiony jako zalogowany użytkownik do głównego menu.

# Wylogowanie z aplikacji

Wylogowanie z aplikacji jest możliwe tylko z intencji głównego menu po kliknięciu w trzy kropki w prawym górnych rogu co otworzy małe okienko.



Rys. 6.23. Widok ekranu

Źródło: opracowanie własne.

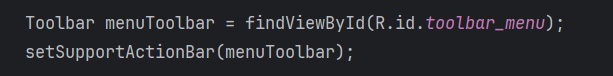
Będzie tam pozycja „Wyloguj się”. Po jej kliknięciu użytkownik zostanie wylogowany, główne menu zostanie zamknięte a użytkownik wróci do wyboru czy chce się zalogować czy założyć nowe konto.



Rys. 6.24. Widok ekranu

Źródło: opracowanie własne.

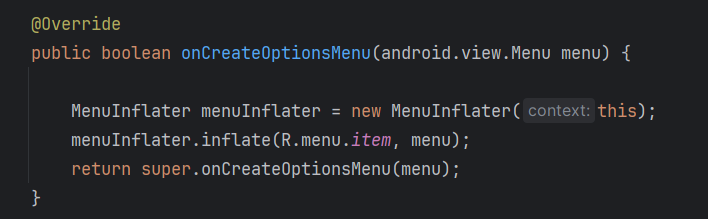
Logikę menu w której pojawiają trzy kropki opcji oraz okienko z opcjami obsługuje widok toolbar menu.



Rys. 6.25. Widok kodu

Źródło: opracowanie własne.

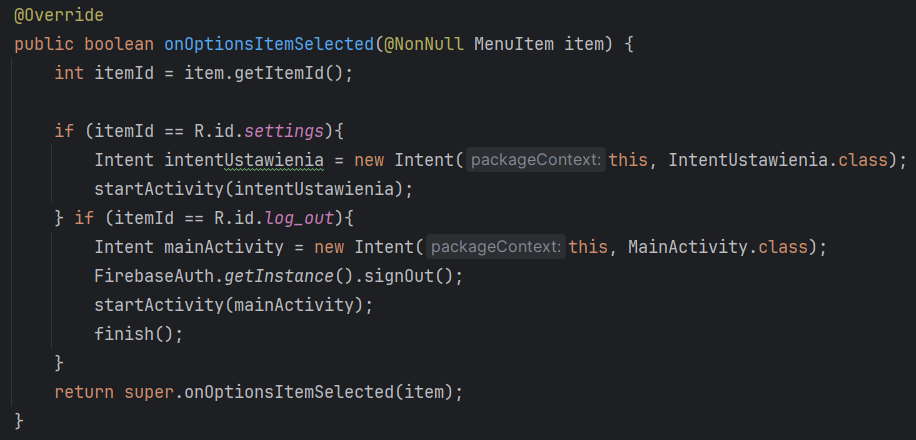
Przypisuje on zmiennej menuToolbar widok, który wywołuje właściwy widok co z kolei wywołuje funkcję onCreateOptionMenu.



Rys. 6.26. Widok kodu

Źródło: opracowanie własne.

Funkcja ta tworzy item z wcześniej przygotowane layout, które to są rzeczonymi opcjami.



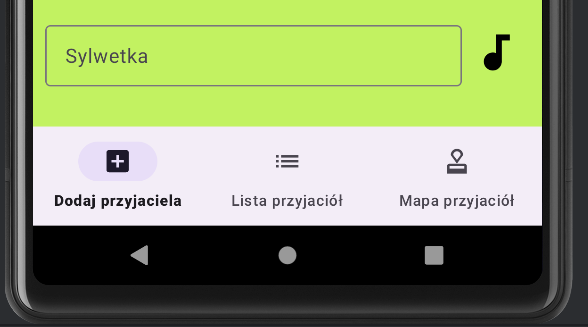
Rys. 6.27. Widok kodu

Źródło: opracowanie własne.

Obsługą logiki, która się aktywuje po kliknięciu, którejś z opcji zajmuje się onOptionItemSelected. Po wybraniu pozycji pobiera on jej wartość a następnie porównuje ją z id item. W naszej sytuacji wybrane zostaje log\_out. Wywołuje on funkcje signOut na instancji Fire Base co powoduje wylogowanie się aktualnie zalogowanego użytkownika , zamyka aktualnie otwartą intencje i otwiera tą z wyborem logowania lub rejestracji.

# Tworzenie nowych znajomych do listy znajomych

Dodawanie wyświetla się domyślnie w widoku po zalogowaniu ale również dostępne jest z menu znajdującym się na dole ekranu.

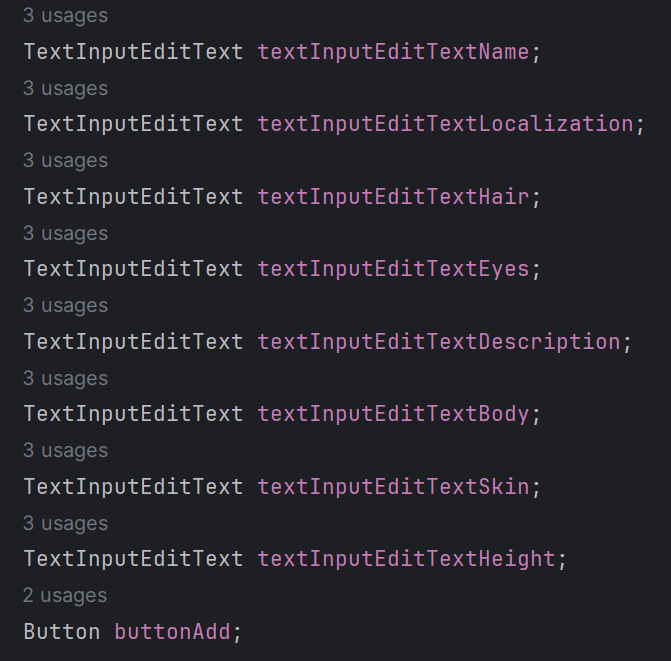


Rys.6.28. Widok ekranu

Źródło: opracowanie własne.

W celu stworzenia nowej pozycji przyjaciela należy uzupełnić co najmniej jedną pole tekstowe i kliknąć przycisk zapisz. Po tej czynności nowy przyjaciel wyświetli się w liście przyjaciół, która dostępna jest z dolnego menu na środku.

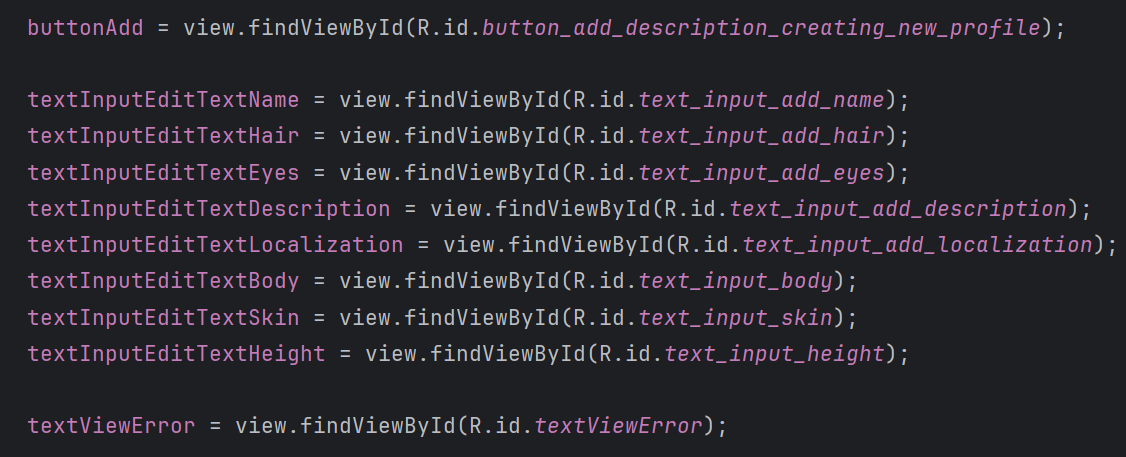
W celu implementacji tej funkcjonalności najpierw zainicjalizowano pola tekstowe , przycisk i pole tekstu .



Rys. 6.29. Widok kodu

Źródło: opracowanie własne.

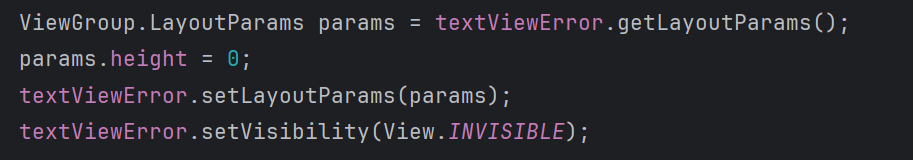
Do każdej zainicjalizowanej pozycji przypisano istniejący widok.



Rys. 6.30. Widok kodu

Źródło: opracowanie własne.

Do widoku tekstu informującego użytkownika o błędzie przypisano dodatkowy parametr, który pozwala na jego schowanie. Ma to na celu odkrycie go kiedy użytkownik spróbuje dodać pustą pozycję do listy.



Rys. 6.31. Widok kodu

Źródło: opracowanie własne.

Następnie przypisano do przycisku funkcję setOnClickListener co pozwala wywołać odpowiedni kod dopiero kiedy użytkownik go kliknie.



Rys. 6.32. Widok kodu

Źródło: opracowanie własne.

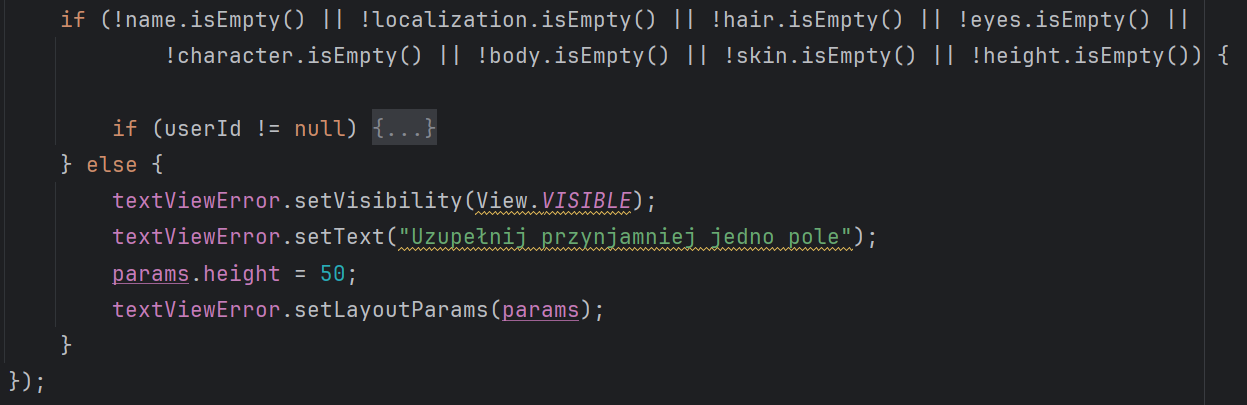
Wywołany wtedy kod pobiera tekst z każdego pola tekstowego i przypisuje je do zmiennych. Tworzy zmienną przechowującą unikalny klucz dla węzła w bazie danych i nowy obiekt typu friendData z pobranymi danymi.



Rys. 6.33. Widok kodu

Źródło: opracowanie własne.

Pobrane dane sprawdzane są czy nie zawierają pustego stringa. Jeśli każde pole tekstowe było puste to zostanie wywołany kod odpowiedzialny za wyświetlanie informacji o tym żeby uzupełnić co najmniej jedno pole.



Rys. 6.34. Widok kodu

Źródło: opracowanie własne.

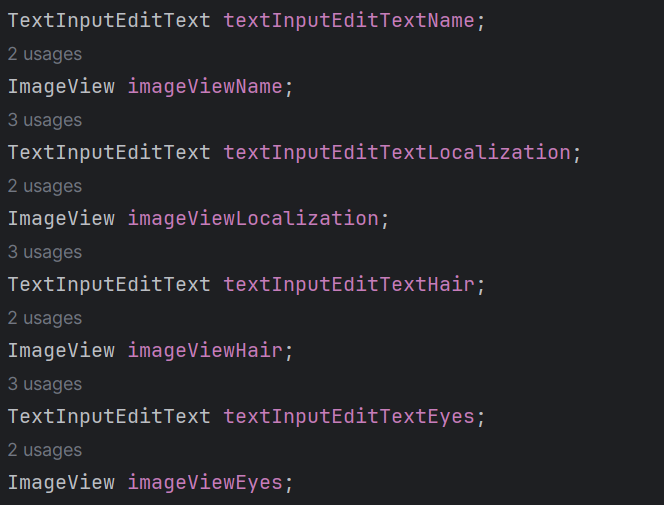
Gdy co najmniej jedno pole zostało wypełnione to na obiekcie dataBase zostanie wywołana próba dostania się do danych przypisanych dla userId. Jako, że userId jest unikalnym kodem dla węzła to zostanie stworzona nowa pozycja o jego nazwie zawierająca dane przechowywane aktualnie w friendData. Stworzona pozycja będzie dostępna w formacie Json w bazie danych FireBase na serwerze.



Rys. 6.35. Widok kodu

Źródło: opracowanie własne.

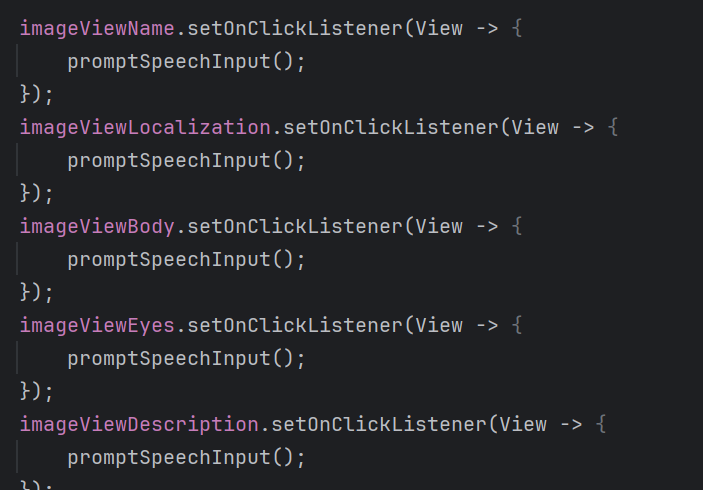
# Mowa na tekst w tworzeniu nowego znajomego



Rys. 6.36. Widok kodu

Źródło: opracowanie własne.

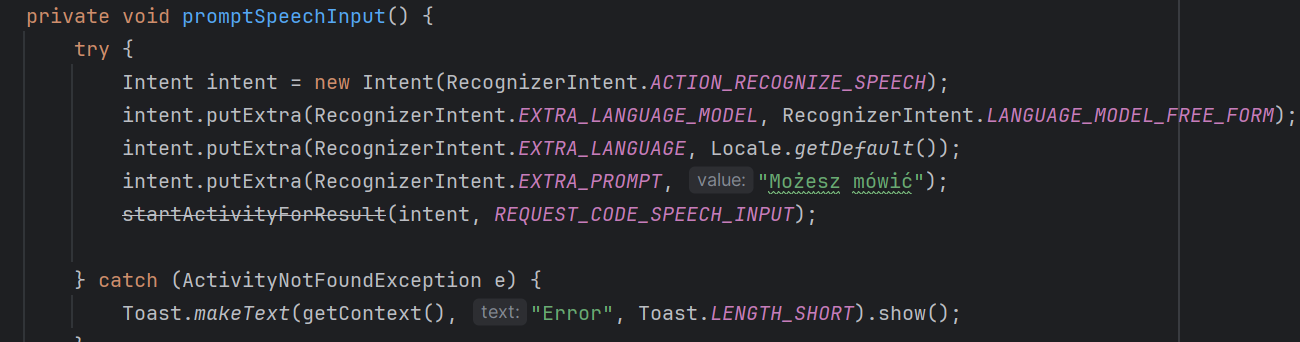
Pierwszym krokiem w celu implementacji tej funkcjonalności było zainicjalizowanie odpowiednich ImageView.



Rys. 6.37. Widok kodu

Źródło: opracowanie własne.

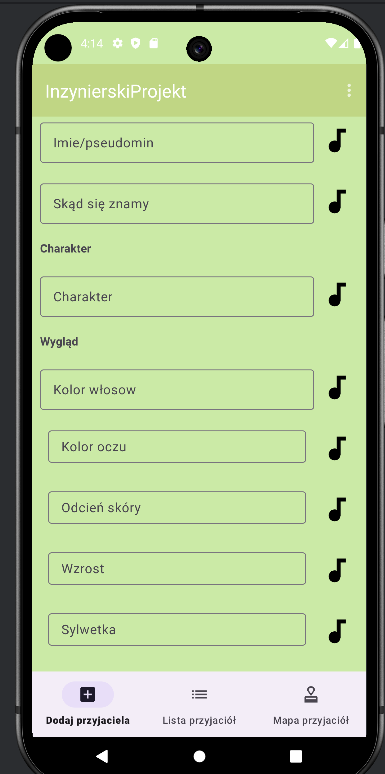
Następnie wywołana jest metoda setOnClickListener na każdym z nich co pozwala wykonać promptSpeechInput. Dzięki temu cały czas nasłuchiwane jest czy użytkownik kliknął w obrazek.



Rys. 6.38. Widok kodu

Źródło: opracowanie własne.

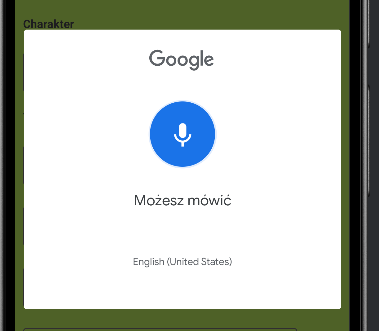
Funkcja przedstawiona powyżej tworzy nowy. W celu przetwarzania mowy na tekst używane jest SpeechRecognizer czyli wbudowana w system Android opcja przetwarzania mowy.



Rys. 6.39. Widok ekranu

Źródło: opracowanie własne.

Pierwszym krokiem było dodanie dodatkowego widoku obok pól tekstowych. Użyto ikonki nutki w celu zasugerowania użytkownikowi, że po kliknięciu w ten obrazek stanie się coś związanego z dźwiękiem. Po kliknięciu w nutkę otworzy się osobny intent, w którym głoś użytkownika będzie zapisywany.



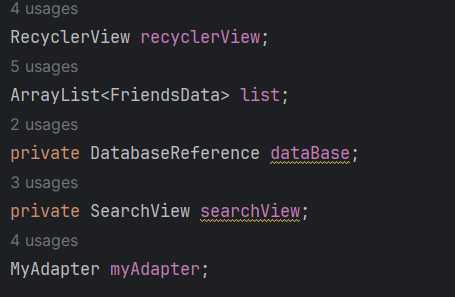
Rys. 6.40. Widok ekranu

Źródło: opracowanie własne.

Po zakończeniu wypowiedzi przez użytkownika i kliknięciu przez niego ikonę mikrofonu, to to co powiedział zostanie zamienione na tekst i przekazane do asystenta chatu GPT, który przetworzy dany tekst. Pozwoli to użytkownikowi na opowiedzenie mniej więcej co chcę napisać a chat to sformatuje i wyciągnie tylko ważne informacje. Następnie przetworzony tekst wstawiany jest w odpowiednie pole tekstowe. Alternatywnie użytkownik zamiast uzupełniać mową pojedyncze pola może wybrać opcje opisania całej osoby naraz. Tekst zostaje przekazany do sztucznej inteligencji a zwrócone informacje są wstawiane w odpowiednie pola.

# Lista znajomych

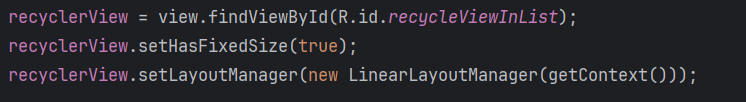
Lista znajomych generowana jest poprzez pobranie danych z bazy danych firebase dla aktualnie zalogowanego użytkownika. Dzieję się to poprzez poprzez dodanie osobnego rekordu z unikalnym Uid każdego użytkownika. W tym rekordzie zapisują się nowo dodani przyjaciele jako kolejne rekordy. Dzięki temu zapisane dane nie wyświetlą się dla innych użytkowników.



Rys. 6.50. Widok kodu

Źródło: opracowanie własne.

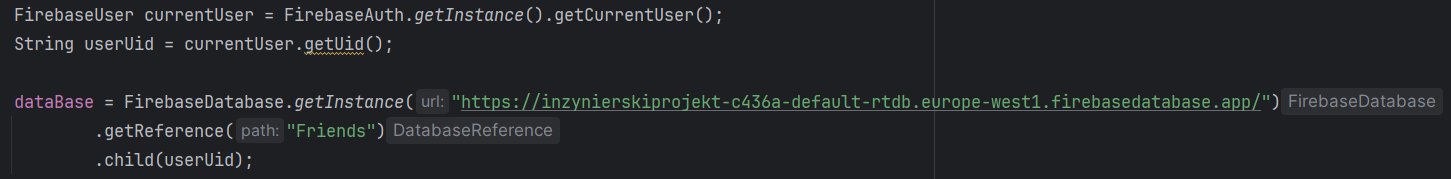
Na początku inicjalizujemy niezbędne obiekty przed wywołaniem onViewCreated.



Rys. 6.51. Widok kodu

Źródło: opracowanie własne.

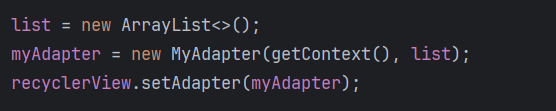
Przypisany zostaje recycleView do odpowiadającemu mu widoku w pliku xml. Ustawiona zostaje również stała ilość elementów w trakcie działania aplikacji oraz przypisuje do niego LinearLayoutManager.



Rys. 6.52. Widok kodu

Źródło: opracowanie własne.

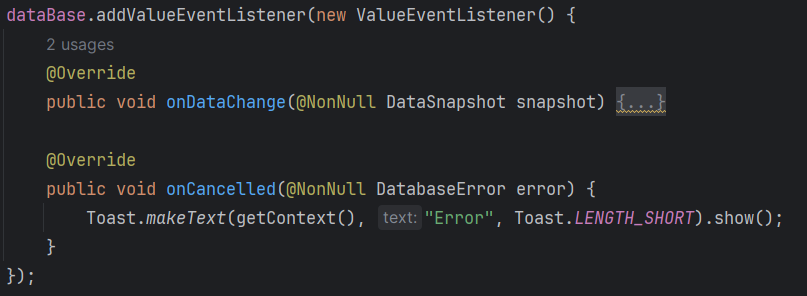
Stworzona zostaje instancja FirebaseAuth, z której pobierany jest aktualnie zalogowany użytkownik oraz jego Uid. Następnie tworzona jest instancja FirebaseDatabase, do której przekazujemy referencje Friends, która zawiera wszystkich użytkowników, na której wywołujemy uzyskanie rekordu z nazwą zależną od aktualnego Uid. Dzięki temu uzyskujemy dostęp do wszystkich zapisanych przyjaciół.



Rys. 6.53. Widok kodu

Źródło: opracowanie własne.

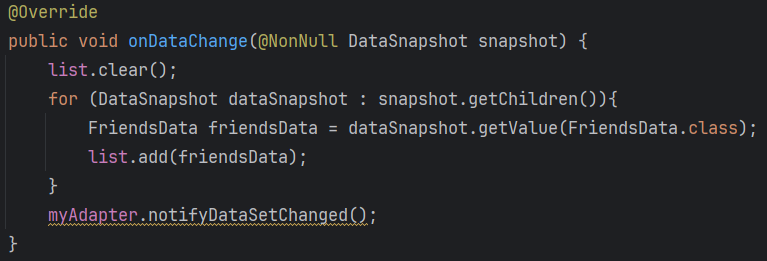
Stworzona zostanie lista, która od razu jest przypisywana do myAdapter, który z kolei przypisywany jest do recycleView. To właśnie adapter odpowiedzialny jest za zarządzanie danymi i przypisanie ich poprawnie do odpowiedniego widoku.



Rys. 6.54. Widok kodu

Źródło: opracowanie własne.

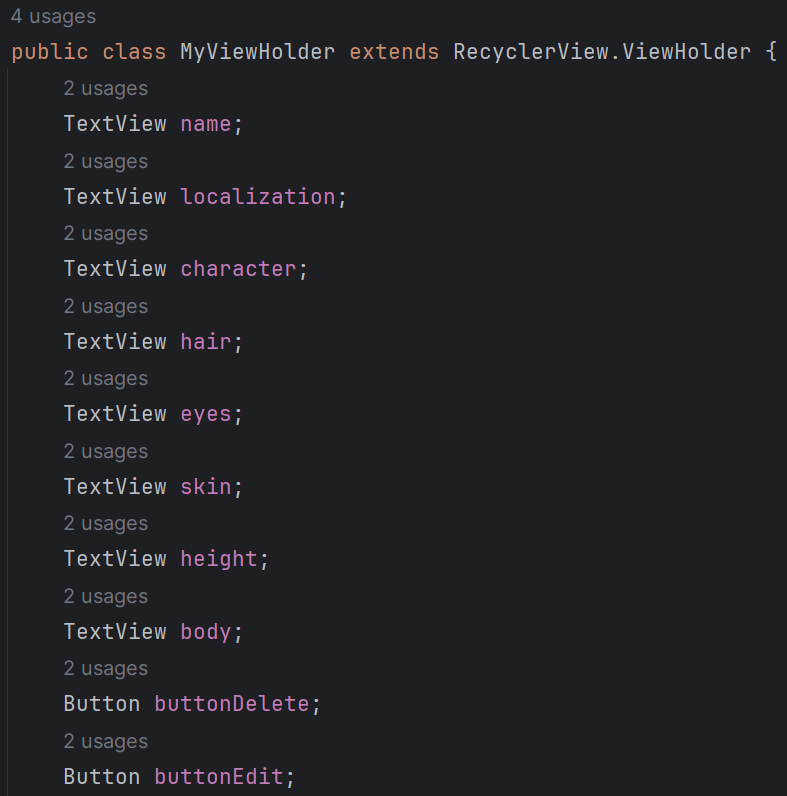
W celu pobrania danych przypisany na obiekcie dataBase wywoływana zostaje metoda addValueEventListener. Dzięki temu obiekt zacznie nasłuchiwać zmian na węźle oraz dostaniemy dostęp do onDataChange i onCanceled. OnCanceled pozwala wywołać kod gdy odczytanie danych zakończy się niepowodzeniem. Gdy jednak się to uda, zostanie wykonany poniższy kod.



Rys. 6.55. Widok kodu

Źródło: opracowanie własne.

Dla pewności by żadne dodatkowe dane nie zostały wyświetlone, to lista zostaje wyczyszczona po czym następuje iteracja po dzieciach aktualnego węzła. Uzyskujemy w ten sposób dostęp do każdego zapisanego przyjaciela, który zostaje zapisany na liście. Na koniec wywoływana jest metoda notifyDataSetChanged na adapterze co informuje go o zmianie danych na liście, która wcześniej została do niego przypisana.



Rys. 6.56. Widok kodu

Źródło: opracowanie własne.

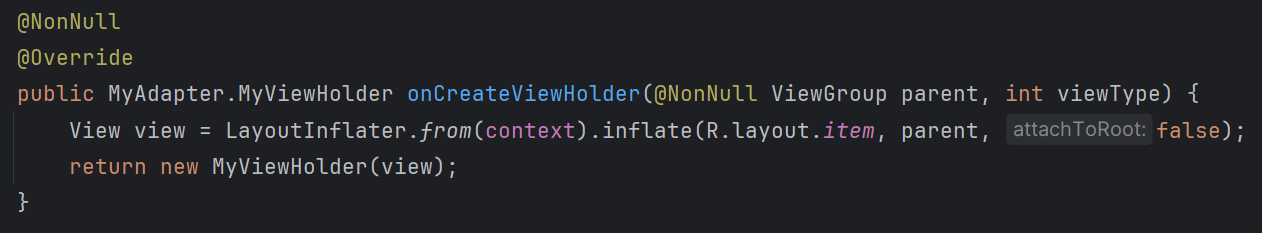
W celu przypisania wartości do ich odpowiedniej pozycji definiowane są pola na tekst i przyciski.



Rys. 6.57. Widok kodu

Źródło: opracowanie własne.

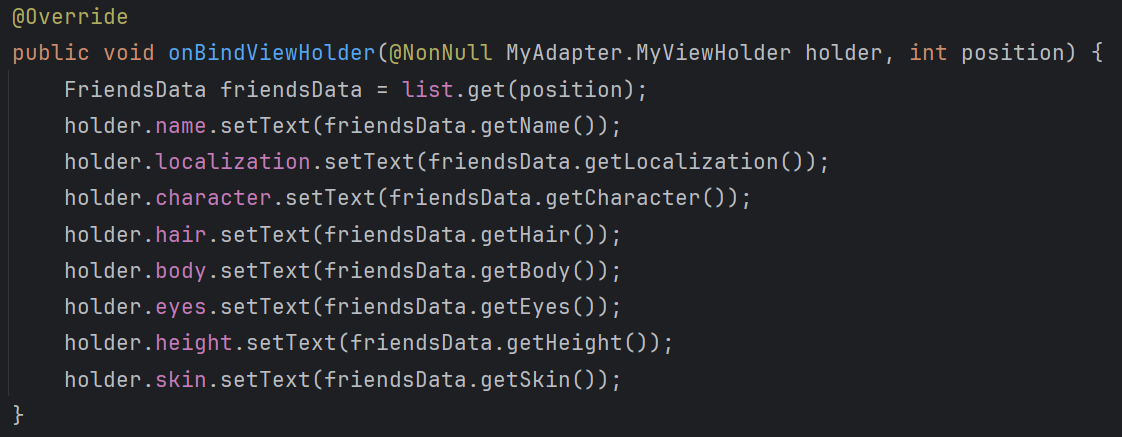
Do pól tekstowych i przycisków przypisywane są istniejące widoku z pliku xml. Do guzików przypisuje się funkcje wywołujące kod w momencie wciśnięcia.



Rys. 6.58. Widok kodu

Źródło: opracowanie własne.

Adapter tworzy nowy widok dla przyjaciela wzorując się na na pliku xml item.



Rys. 6.59. Widok kodu

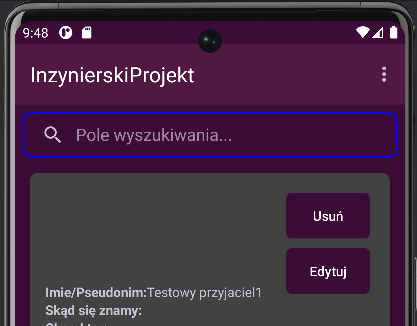
Źródło: opracowanie własne.

Na końcu pobrane dane zostają przypisane do odpowiedniego pola poprzez odwołanie się do niego na obiekcie holder, który jest tymczasowym pojemnikiem na widoki.

Wszystkie czynności w adapterze powtórzą się dla każdego rekordu w bazie danych w FireBase dla aktualnego węzła.

# Wyszukiwarka w liście przyjaciół

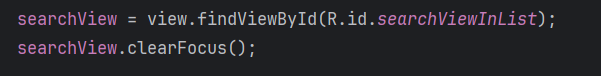
Wyszukiwanie odbywa się poprzez wpisanie interesującej użytkownika frazy do pola wyszukiwania w zakładce z listą przyjaciół.



Rys. 6.60. Widok ekranu

Źródło: opracowanie własne.

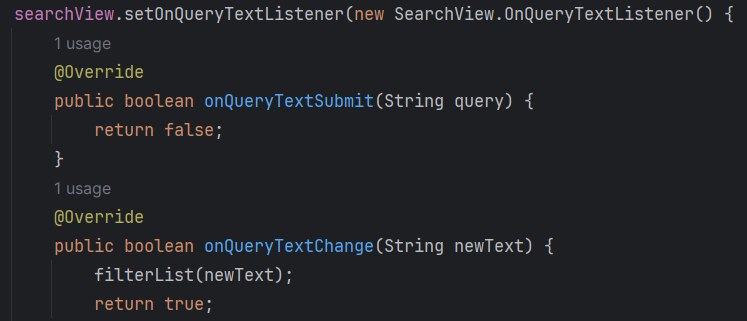
Znalezieniem odpowiedniego widoku zajmuje się searchView.



Rys. 6.61.

Źródło: opracowanie własne.

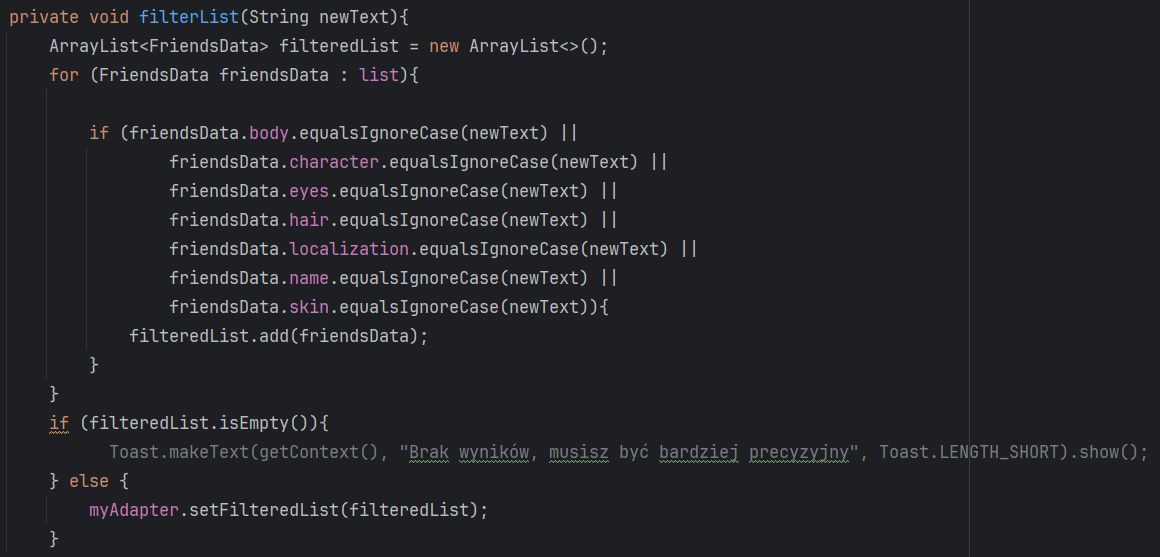
Jest on przypisywany do odpowiedniego widoku a następnie wywoływana jest na nim funkcja clearFocus, która zamknie klawiaturę jeżeli aplikacji wybierze domyślnie searchView w celu wpisania tam tekstu.



Rys. 6.62.

Źródło: opracowanie własne.

Następnie ustawiany jest nasłuchiwacz, który w momencie wpisania przez użytkownika tekstu do searchView przekaże go do funkcji filterList. Będzie się to działo po wpisaniu lub usunięciu każdej litery dla tego użytkownikowi na bieżąco będą się zmieniały widoki.



Rys.

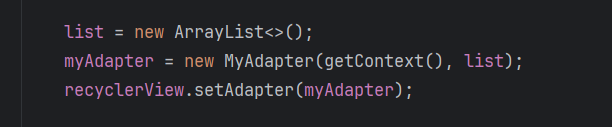
Źródło: opracowanie własne.

Funkcja filterList tworzy nową listę, do której zostaną przypisane te elementy widoku recycleView, które w polach tekstowy zawierają wyszukiwaną frazę. Dzieje się to w pętli for, która iteruje po aktualnej liście pobranej z bazy danych. Dla każdego obiektu sprawdzane jest czy wyszukiwana fraza odpowiada zapisanym danym. W przypadku gdy frazy są takie same obiekt zostanie przypisany do listy stworzonej na początku.

Na koniec jeżeli lista filteresList jest pusta to nic się nie stanie ale w przeciwnym razie zostanie przekazana do adaptera, gdzie zastąpi aktualnie wywoływaną listę.

# Usuwanie wierszy w liście znajomych

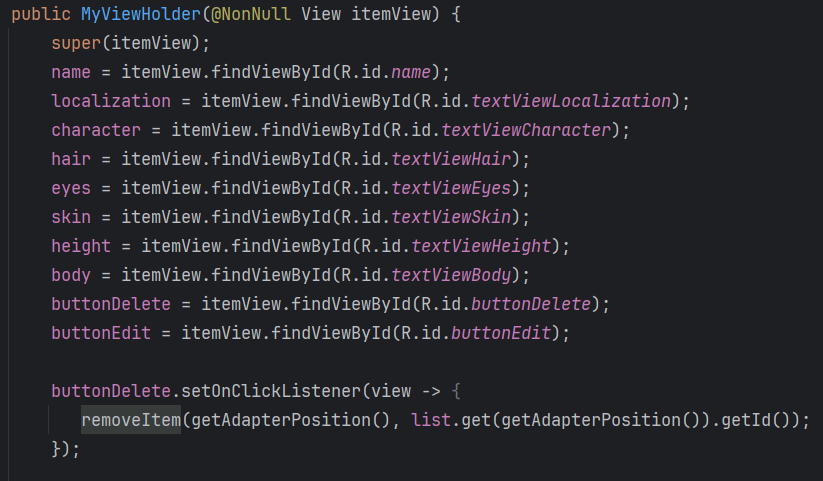
Usuwanie obsługiwane jest przez funkcje removeItem w klasie MyAdapter.



Rys.

Źródło: opracowanie własne.

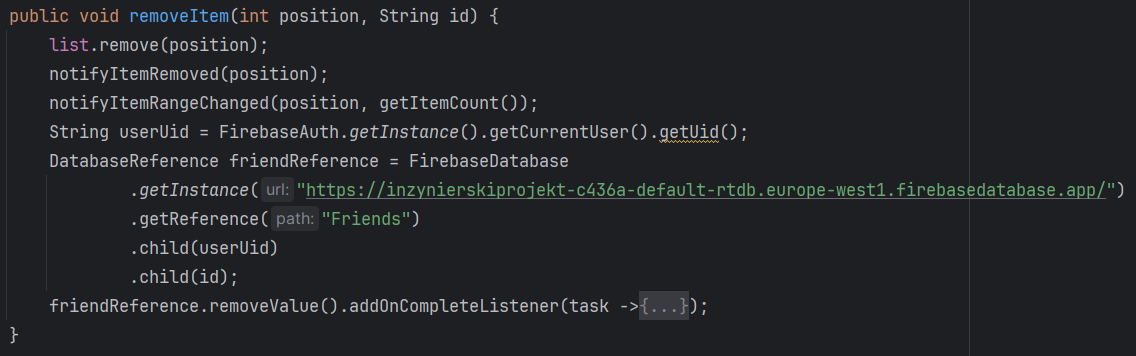
Jak na załączonym obrazku widać, trzeba przypisać obiekt adaptera do obiektu recycleView. Dzięki temu elementy recycleView będą miały dostęp do funkcji adaptera.



Rys.

Źródło: opracowanie własne.

Wpierw przypisany zostaje przycisk, który obsługuję usunięcie pozycji. A następnie przypisanie do niego funkcji, która nasłuchuje moment wciśnięcia. Po kliknięciu wywoływana jest funkcja removeItem, do której przekazana jest pozycja elementu w recycleView oraz id konkretnego przyjaciela w bazie danych.



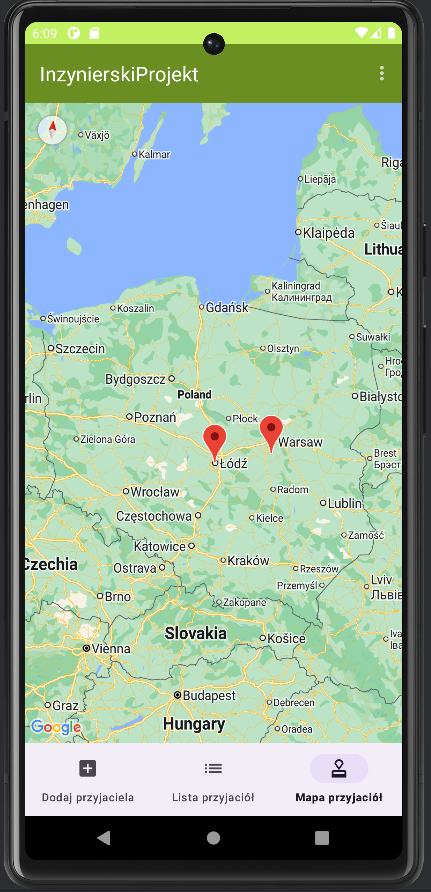
Rys.

Źródło: opracowanie własne.

Następnie na liście, która aktualnie przechowuje dane dla recycleView usuwana jest odpowiednia pozycja. Po tym kroku powiadamiany jest recycle view o usunięciu pozycji oraz o zmianie długości listy. Pobierany jest Uid aktualnie zalogowanego użytkownika i tworzony jest obiekt friendReference, który przechowuje jakie dane mają zostać usunięte. Na koniec na tym obiekcie wywoływana jest funkcja usuwająca te dane.

# Mapa z pinezkami dla przyjaciół

Mapa dostępna jest z dolnego menu po prawej. Pozwala ona zobaczyć mapę Google z dodanymi pinezkami dla przyjaciół dla, których uzupełniło się lokalizację w formie współrzędnych rozdzielonych przecinkiem.



Rys.

Źródło: opracowanie własne.

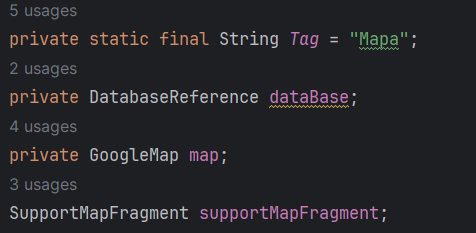
Po kliknięciu pinezki wyświetli się imię przyjaciel, którego poznało się w tym miejscu. Oczywiście jeżeli uzupełniło się pole imię przy tworzeniu przyjaciela.



Rys.

Źródło: opracowanie własne.

Mapa działa dzięki pakietowi Maps SDK na Androida. W celu jej inicjalizacji i pinezek dodano pozycję przedstawione na obrazku poniżej.



Rys.

Źródło: opracowanie własne.

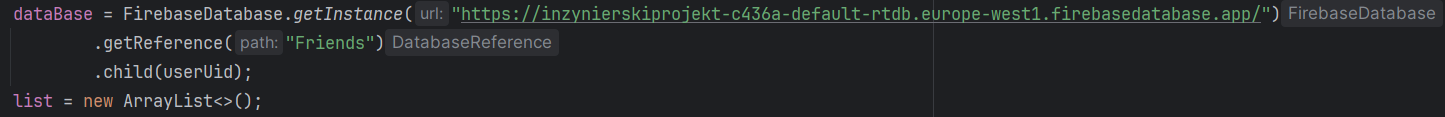
Następnie przypisywany jest supportMapFragmnet do jego widoku w pliku xml oraz sprawdza się czy został zainicjalizowany. Jeśli tak to, mapa zostanie załadowana w sposób asynchroniczni co pozwoli nie blokować głównego wątku aplikacji.



Rys.

Źródło: opracowanie własne.

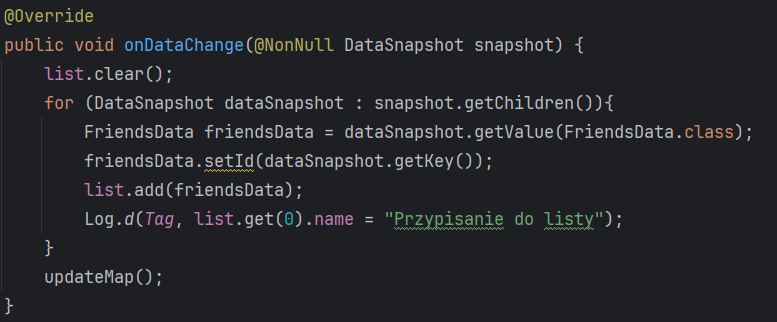
Teraz tworzony jest obiekt typu FirebaseDatabase i pobierane zostają dane dla aktualnie zalogowane użytwkonika oraz tworzona jest nowa pudsta lista.



Rys.

Źródło: opracowanie własne.

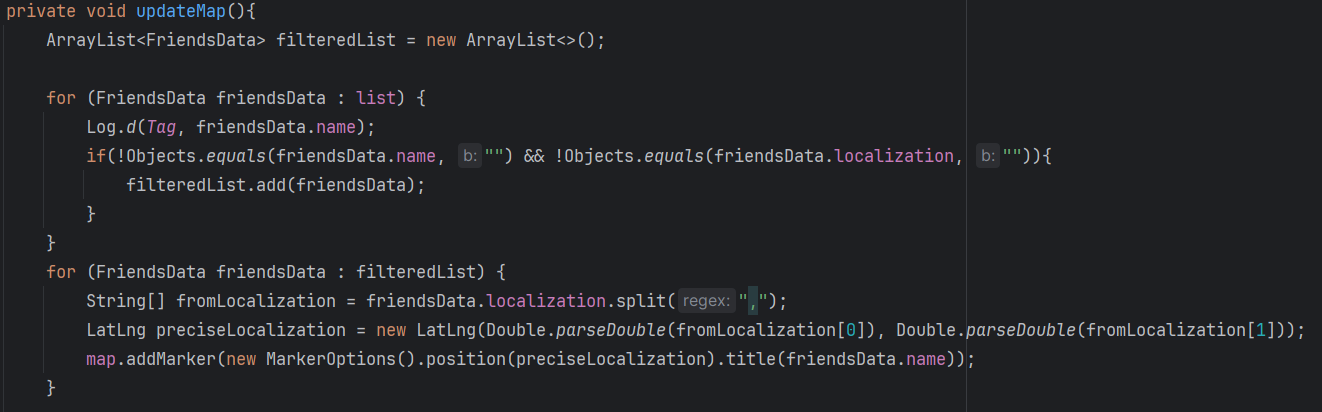
Na powstałym obiekcie dataBase wywołujemy metodę addValueListener i defniowane są metody onDataChange oraz onCancelled. Pierwsza pozwala na pobranie danych z serwera a druga na wywołanie konkretnego kodu jeśli sczytanie danych zakończy się niepowodzeniem.



Rys.

Źródło: opracowanie własne.

Po poprawnym pobraniu danych następuje przypisanie każdego przyjaciela do listy do czego służy pętla for, która iteruje po dzieciach węzła dataBase. A na koniec wywoływana jest metoda updateMap.



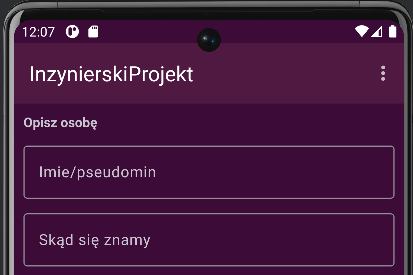
Rys.

Źródło: opracowanie własne.

To w tym miejscu następuje utworzenie pinezek. Najpierw lista jest filtrowana tak by w liście filteredList zostali tylko Ci przyjaciele, którzy mają uzupełnione pole z imieniem i lokalizacją. Dzięki temu możemy iterować po każdym elemencie przefiltrowanej listy i pobrać potrzebne dane. Najpierw rozdzielane są współrzędne na długość i szerokość geograficzna. Potem tworzony jest obiekt typu LatLng, który razem z imieniem przekazywany jest do metody addMarker wywoływanej na obiekcie map co powoduje stworzenie pinezki z imieniem.

# Ustawienia

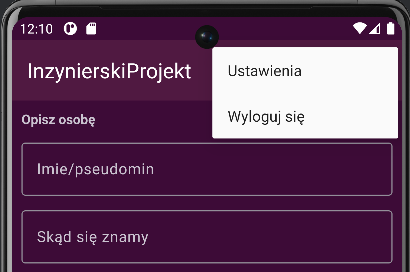
Ustawienia są zarządzane poprzez shared preferences. Przechowują one zbiór wartości w postaci klucz-wartość co pozwala na zapisywanie w nich ustawień, które mają być zastosowane w całej aplikacji.



Rys.

Źródło: opracowanie własne.

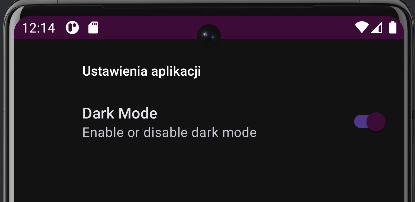
W celu otworzenia ustawień należy najechać i kliknąć na 3 kropki w prawym górnym rogu aplikacji. Są one dostępne w głównym menu, logowaniu, rejestrowaniu czy wyborze czy użytkownik chce się zalogować czy zarejestrować.



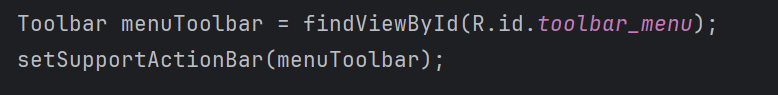
Rys.

Źródło: opracowanie własne.

Po kliknięciu w trzy kropki należy wybrać pozycję ustawienia. Przeniesie ona użytkownika do nowej intencji, w której znajdują się rzeczone ustawienia.



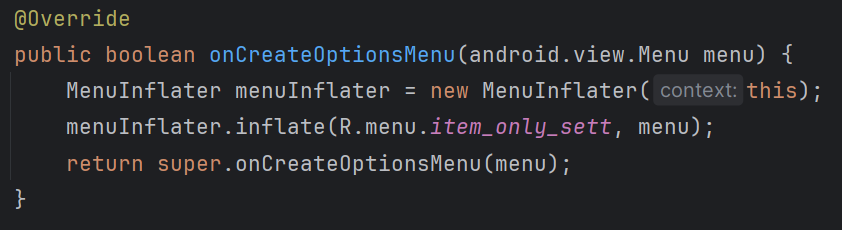
Z tego poziomu można wybrać interesujące nas ustawienie oraz powrócić do poprzedniego ekranu.



Rys.

Źródło: opracowanie własne.

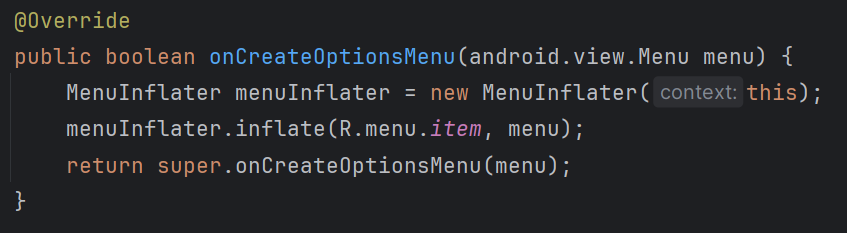
Aby ustawienia mogły zostać wyświetlone, w każdej intencji tworzony jest obiekt menuToolbar, do którego przypisywany zostaje odpowiadający mu widok. Obsługuje on wyświetlanie się górnego paska, w którym znajduje się dostęp do ustawień.



Rys.

Źródło: opracowanie własne.

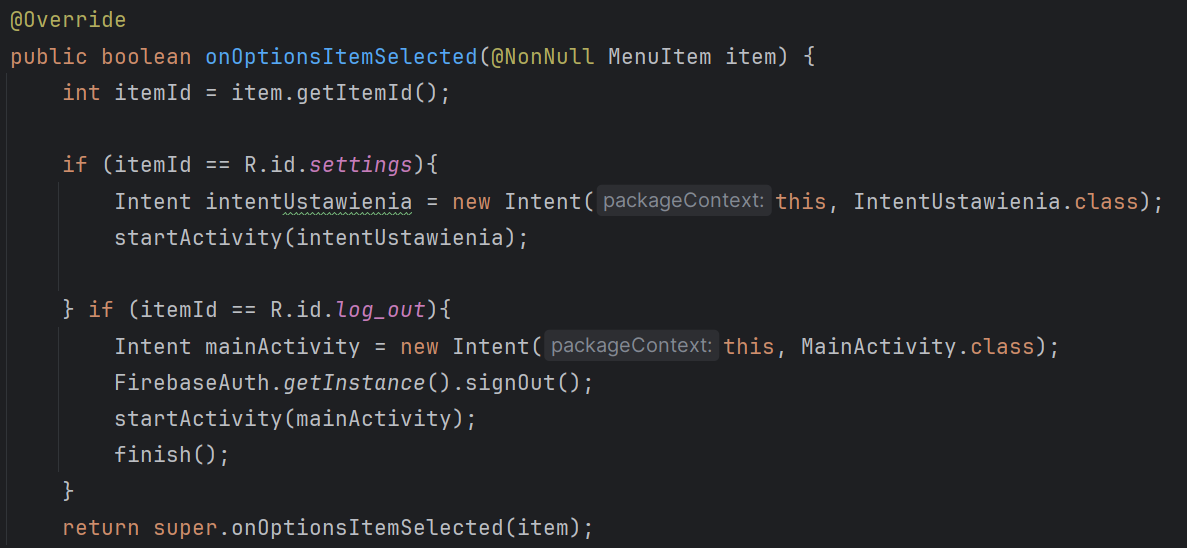
Niezbędne jest również napisanie metody onCreateOptionMenu. Przypisuje się w niej plik xml odpowiadający za wyświetlanie menu z ustawieniami.



Rys.

Źródło: opracowanie własne.

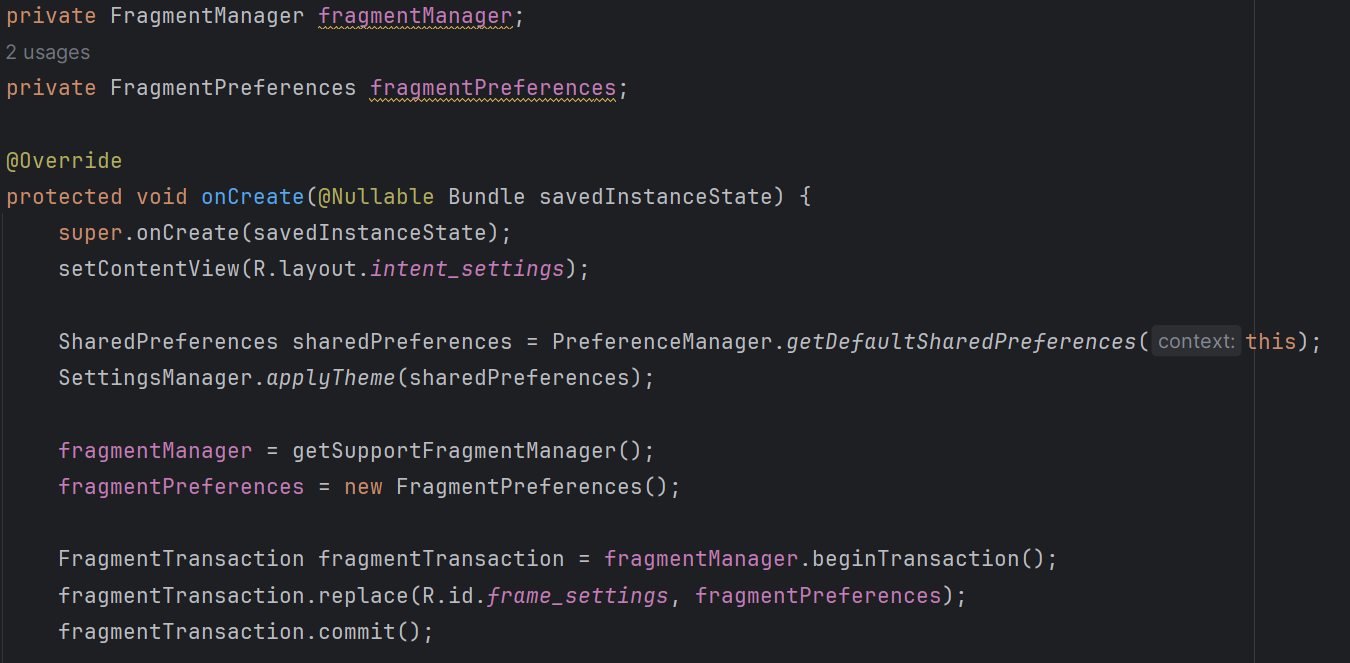
W intencjach dostępnych po zalogowaniu przypisywany zostaje inny plik xml, który wyświetla również opcje wylogowania się.



Rys.

Źródło: opracowanie własne.

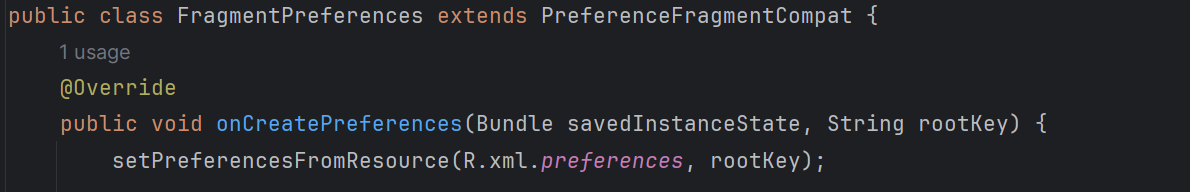
Kliknięcie w ustawienia lub w opcje wylogowania się obsługuje onOptionsItemSelected. W momencie kliknięcia, id aktualnie klikniętej opcji zostaje pobrane i przyrównane do nazw w pliku xml. Jeżeli kliknięty zostanie z id settings to zostanie uruchomiony kod odpowiedzialny za otworzenie nowej intencji, w której będą ustawienia.



Rys.

Źródło: opracowanie własne.

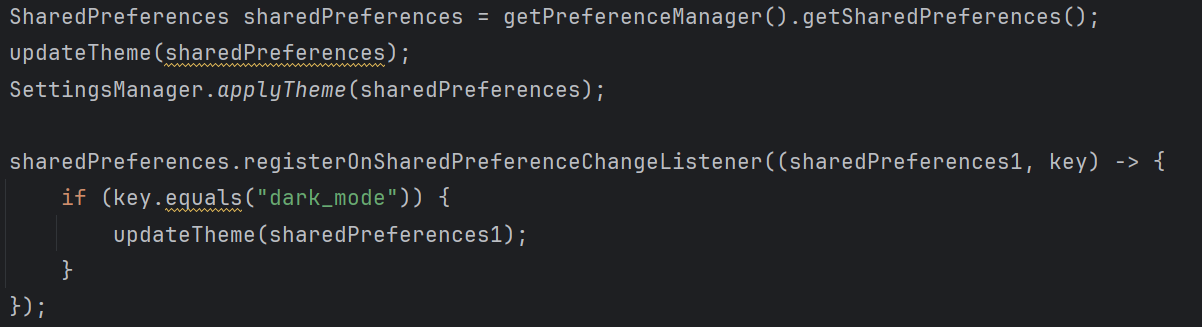
W nowo otwartej aktywności znajduje się jedynie widok pozwalający na wyświetlenie się fragmentu. To w nim będą wyświetlane ustawienia.



Rys.

Źródło: opracowanie własne.

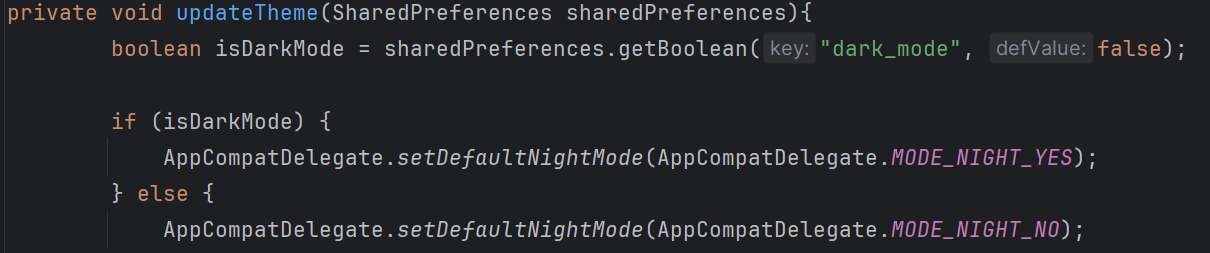
W tej klasie przypisujemy widok fragmentu do odpowiedniego widoku w xml. Po czym definiujemy kod potrzebny do działania ustawień.



Rys.

Źródło: opracowanie własne.

W tym celu stworzony zostaje obiekt sharedPreferences, na którym zostaje wywołana metoda registerOnSharedPreferenceChangeListener. Nasłuchuje ona zmian na przełączniku w ustawieniach. Gdy zostanie on kliknięty metodzie updateTheme przekazany zostanie sharedPreferences1, który zawiera aktualny motyw.

\*\*

Rys.

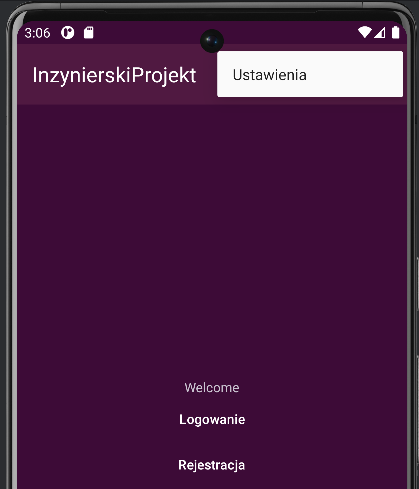
Źródło: opracowanie własne.

UpdateTheme gdy zostanie wywołany pobiera on wartość prawda lub fałsz dla przekazanego mu motywu. Domyślnie wartość ustawiona jest na false co aktywuje kod odpowiedzialny za ustawienie motywu dzinnego. Gdy wartość ustawiona jest na prawdę to aktywowany zostaje kos odpowiedzialny za ustawienie motywu ciemnego.

\*\* (Chyba niepotrzebne? Bo opisuje podobną rzecz w następnym podpunkcie?)

# Zmiana trybu na dzienny lub nocny

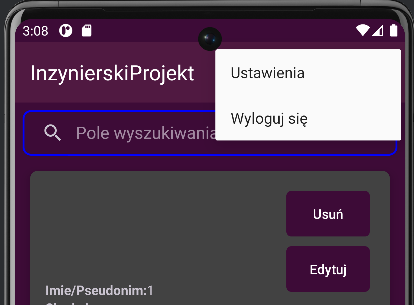
Zmiana trybu dziennego na nocny i odwrotnie odbywa się w ustawieniach, do których dostęp jest w każdej aktywności.



Rys.

Źródło: opracowanie własne.

Aktywność gdy użytkownik otwiera aplikacje i nie jest zalogowany.

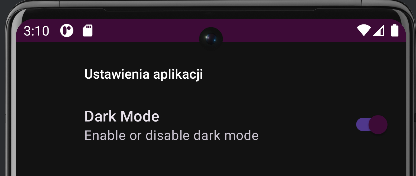


Rys.

Źródło: opracowanie własne.

Aktywność gdy użytkownik jest zalogowany.

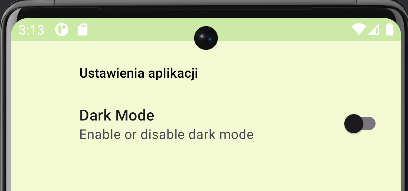
Po kliknięciu w Ustawienia pojawia się nowa intencja, w której mamy dostęp do zmiany trybu.



Rys.

Źródło: opracowanie własne.

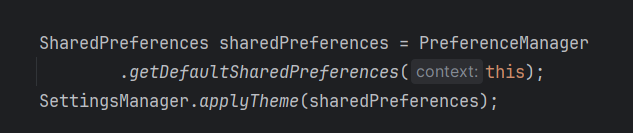
Zmiana trybu następuje po wciśnięciu przełącznika.



Rys.

Źródło: opracowanie własne.

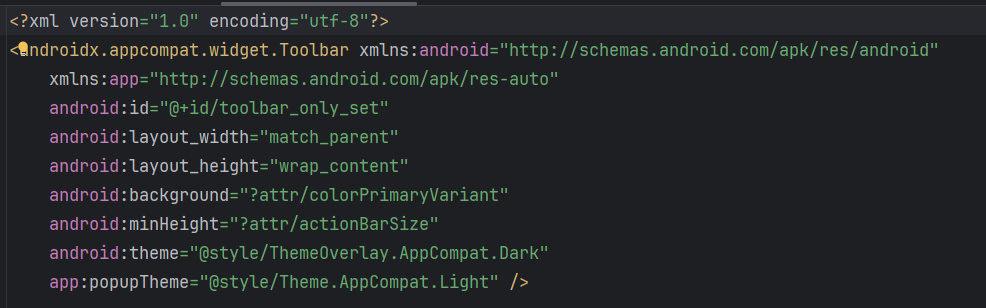
Działa to poprzez zmianę themes za pomocą shared preferences. Pozwalają one zapisać wartości w parach klucz-wartość.



Rys.

Źródło: opracowanie własne.

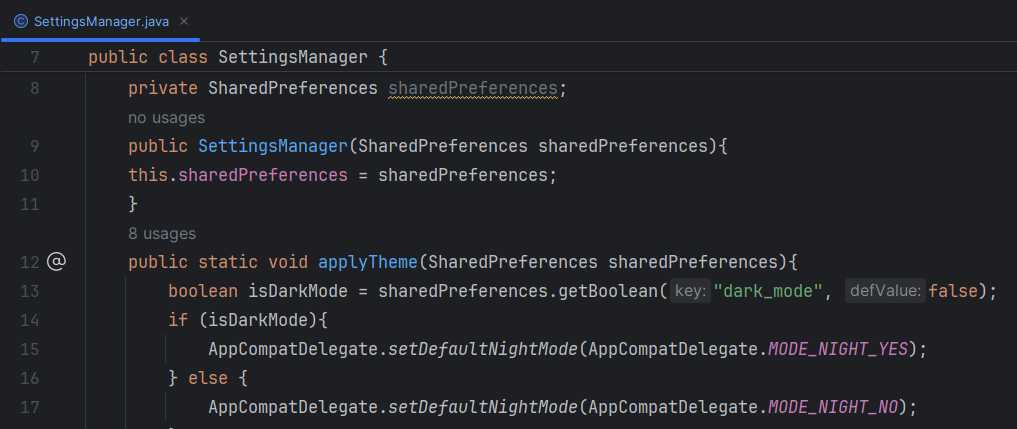
Przedstawiony kod tworzy instancję sharedPreferences, z której pobieramy domyślne ustawienia. Następnie wywołujemy funkcję applyTheme, która systemowo zmienia styl.



Rys.

Źródło: opracowanie własne.

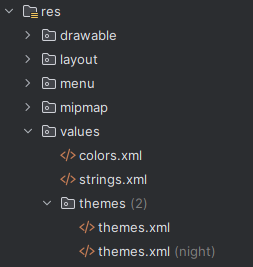
Na obrazku powyżej widać użycie colorPrimaryVariant, które znajduje się w obu stylach. Aplikacja przypisuje kolor z odpowiedniego stylu na podstawie, które ustawienia zostały zastosowane.



Rys.

Źródło: opracowanie własne.

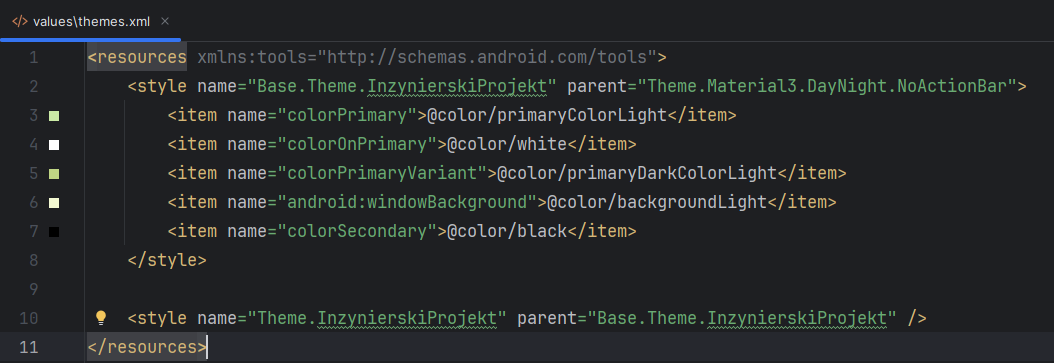
Dzieje się to w klasie SettingsManager. ApplyTheme sprawdza czy aktualnie ustawiony jest tryb nocny. Jeśli nie to aplikacja jest informowana o ustawieniu stylu dziennego i odwrotnie jeśli tak to ustawiany jest styl nocny.



Rys.

Źródło: opracowanie własne.

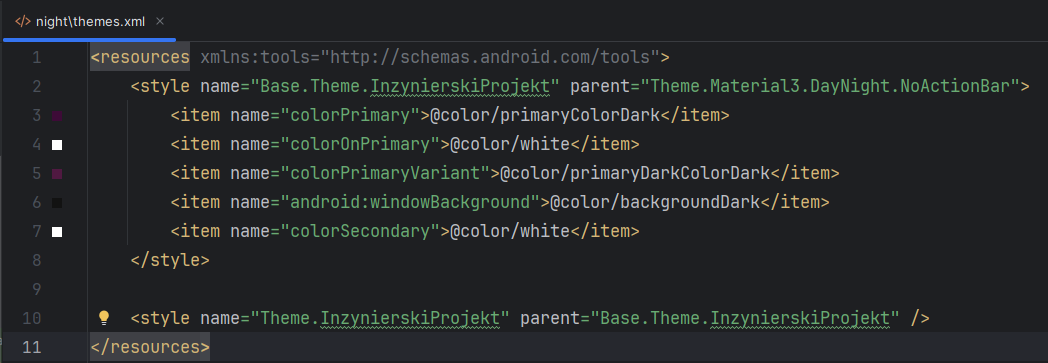
Style deklarowane są w katalogu res/values/themes. Istnieją tam dwa pliki themes.xml. Jeden przechowuje wartości kolorów dla stylu dziennego a drugi dla nocnego.



Rys.

Źródło: opracowanie własne.

W stylu opisane są imiona dla kolorów, których wartości są opisane w res/values/colors.xml. Na obrazku powyżej widać styl zawierający kolory dla trybu jasnego.



Rys.

Źródło: opracowanie własne.

Analogicznie inne kolory są dodane w stylu dla trybu nocnego.

# Testy i wdrożenie

# Testy jednostkowe

# Testy UI

# Bezpieczeństwo i dostępność aplikacji

# Metody dystrybucji aplikacji - napisać co najmniej jedną stronę

# Podsumowanie i wnioski

# Wnioski

# Przyszłość aplikacji (następne kroki)

Wnioski - opiszę dokładnie na końcu

Stworzono aplikację a przede wszystkim nauczono się wielu nowych rzeczy przydatnych przy jej tworzeniu.

# Przyszłość aplikacji (następne kroki)

Następnymi krokami w rozwoju aplikacji jest dalszy rozwój polegający na:

* dodanie nowych funkcjonalności
* poprawa istniejących już funkcjonalności
* poprawienie ogólnego wyglądu aplikacji
* analiza dostępności aplikacji i wprowadzenie potrzebnych zmian w tym zakresie
* stworzenie wersji na system IOS

Do dodatkowych funkcjonalności planowane jest dodanie gry lub lobby, w którym można odwiedzić zapisane osoby, innej gry, która pozwala ćwiczyć imiona poznanych osób, opcji wybrania czy chce się zapisać nowego przyjaciela czy partnera biznesowego oraz możliwość stworzenia awatara w sytuacji kiedy nie posiadamy czyjegoś zdjęcia.

W istniejących funkcjonalność kiedy zapisywany jest nowy przyjaciel powinny być pytania pomocnicze, które mogłyby opisać daną osobę. Po zapisaniu danych puste TextView powinny się chować. Lista przyjaciół powinna mieć różne podgrupy na przyjaciół. Ekran wyboru logowania lub rejestracji jest niepotrzebny i powinien być zastąpiony przez ekran logowania z możliwością przejścia do rejestracji.

Ustawienia powinny zawierać więcej opcji takich jak zmiana stylu czcionki, zmiana języka czy zarządzanie powiadomieniami.

Shared Preferences, które przechowuje ustawienia będzie zamieniona na Data Store, które jest nowszą wersją w pakiecie Android Jetpack.[[9]](#footnote-8)

Przy wyglądzie aplikacji na pewno trzeba zastanowić się nad kolorami czy są zachęcające i przyjemne dla oka oraz czy nie sprawiają ludziom problemów z czytelnością. Zastanowienia wymaga wygląd elementów widoków, kształt i odległości między nimi. Wzorem tego na pewno powinien być design material m3.

Bardzo ważne by aplikacje były dostosowane dla ludzi, którzy różne problemy motoryczne czy poznawcze. Dlatego by nikogo nie wykluczać należy dokładnie przeanalizować aplikację a następnie wprowadzić potrzebne zmiany. Dzięki takim zmianom aplikacja sama w sobie może stać się czytelniejsza, czy przyjaźniejsza dla każdego użytkownika.

Ze względu na cel dotarcia do jak największej grupy odbiorczej nie można zapomnieć o użytkownikach iphone'a. Są oni dużą częścią rynku przez co naturalne jest to, że powstanie wersja aplikacji na IOS.

# Bibliografia

Źródła z literaturowe:

Gaddis T, *Java dla zupełnie początkujących. Owoce programowania. Wydanie VII*, Wydawnictwo Helion, 2019.

Gostkowska-Dźwig S. , E. Kempa, M. Mrozik, R. Królik, Wybrane metody analizy w procesie zarządzania strategicznego, Wydawca Politechnika Częstochowska, 2023, Częstochowa.

Wodo W. ,Kuźma N., Przedsiębiorczość akademicka. Narzędzia i techniki start-upu, Uniwersytet Przyrodniczy we Wrocławiu, Wrocław 2020, s. 27.

Źródła cyfrowe:

<https://wolski.pro/diagramy-uml/> (09.12.2024)

<https://developer.android.com/topic/libraries/architecture/datastore?hl=pl> (24.11.2024)

<https://docs.github.com/en/get-started/start-your-journey/about-github-and-git> (21.11.2024).

<https://docs.github.com/en/get-started/using-git/about-git> (24.11.2024).

<https://developer.android.com/studio/intro?hl=pl> (09.12.2024).

<https://appmaster.io/pl/blog/czym-jest-firebase> (09.12.2024)

Do napisania może:

Napisać o archiwizacji też od razu można o apk napisać

Dokumentacja prawna

Opisać uprawnienia np do internetu czy do zdjęć

Użyć ProGuard może

Może użyć dex2jar

MobFs może użyć - chyba sprawdza jak bezpieczna jest aplikacja

Github action do testowania

Dobre praktyki programowania

Dlaczego jakie api

1. S. Gostkowska-Dźwig, E. Kempa, M. Mrozik, R. Królik, Wybrane metody analizy w procesie zarządzania strategicznego, Wydawca Politechnika Częstochowska, 2023, Częstochowa. [↑](#footnote-ref-0)
2. W. Wodo, N. Kuźma, Przedsiębiorczość akademicka. Narzędzia i techniki start-upu, Uniwersytet Przyrodniczy we Wrocławiu, Wrocław 2020, s. 27. [↑](#footnote-ref-1)
3. <https://wolski.pro/diagramy-uml/> (09.12.2024) [↑](#footnote-ref-2)
4. T. Gaddis, *Java dla zupełnie początkujących. Owoce programowania. Wydanie VII*, Wydawnictwo Helion, 2019. [↑](#footnote-ref-3)
5. <https://developer.android.com/studio/intro?hl=pl> (09.12.2024). [↑](#footnote-ref-4)
6. <https://appmaster.io/pl/blog/czym-jest-firebase> (09.12.2024) [↑](#footnote-ref-5)
7. <https://docs.github.com/en/get-started/start-your-journey/about-github-and-git> (21.11.2024). [↑](#footnote-ref-6)
8. <https://docs.github.com/en/get-started/using-git/about-git> (24.11.2024). [↑](#footnote-ref-7)
9. <https://developer.android.com/topic/libraries/architecture/datastore?hl=pl> 24.11.2024 [↑](#footnote-ref-8)