

Politechnika Wrocławska

Wydział Elektroniki, Fotoniki i Mikrosystemów

---

KIERUNEK: Automatyka i Robotyka

PRACA DYPLOMOWA  
INŻYNIERSKA

Tytuł pracy:

Aplikacja mobilna ułatwiająca seniorom  
nawiązywanie kontaktów towarzyskich

AUTOR:

Roksana Gulewska

PROMOTOR:

Dr inż. Krzysztof Halawa

# Spis treści

1. Wstęp .....	3
1.1 Cele i motywacje pracy .....	3
1.2 Założenia .....	5
1.3 Zakres pracy .....	5
2. Projekt .....	6
2.1 Funkcjonalności .....	6
2.2 Wybór technologii .....	7
2.3 Dostosowanie aplikacji do potrzeb grupy docelowej .....	8
3. Implementacja .....	9
3.1 Rejestracja nowego użytkownika .....	9
3.2 Logowanie i wylogowanie .....	13
3.3 Sprawdzanie połączenia z Internetem .....	17
3.4 Pobieranie informacji o użytkowniku .....	21
3.5 Dodawanie zdjęcia .....	30
3.6 Baza danych .....	36
3.7 Menu .....	38
3.8 Wyświetlanie profilu zalogowanego użytkownika .....	41
3.9 Wyświetlanie potencjalnych znajomych .....	43
3.10 Mechanizm polubień i odrzuceń .....	47
3.11 Wyświetlanie listy dopasowanych użytkowników .....	49
3.12 Wysyłanie i wyświetlanie wiadomości .....	53
3.13 Wyświetlanie profilu dopasowanego użytkownika .....	58
3.14 Usunięcie użytkownika z listy dopasowań .....	60
3.15 Aktualizacja informacji o użytkowniku .....	62
3.16 Usuwanie konta .....	65
4. Podsumowanie .....	68
4.1 Problemy implementacyjne .....	68
4.2 Możliwości rozwoju aplikacji .....	68
4.3 Wnioski końcowe .....	69
4.4 Spis listingów .....	69
4.5 Literatura .....	71

# 1. Wstęp

W niniejszym rozdziale zawarto charakterystykę problemu, który ma za zadanie rozwiązać aplikacja, a także opis motywacji kierujących wyborem tematu pracy. Znajduje się tu również lista założeń jakie powinna spełniać aplikacja, aby z powodzeniem mogła służyć użytkownikom wraz z uzasadnieniem. Na końcu rozdziału zamieszczono zakres pracy.

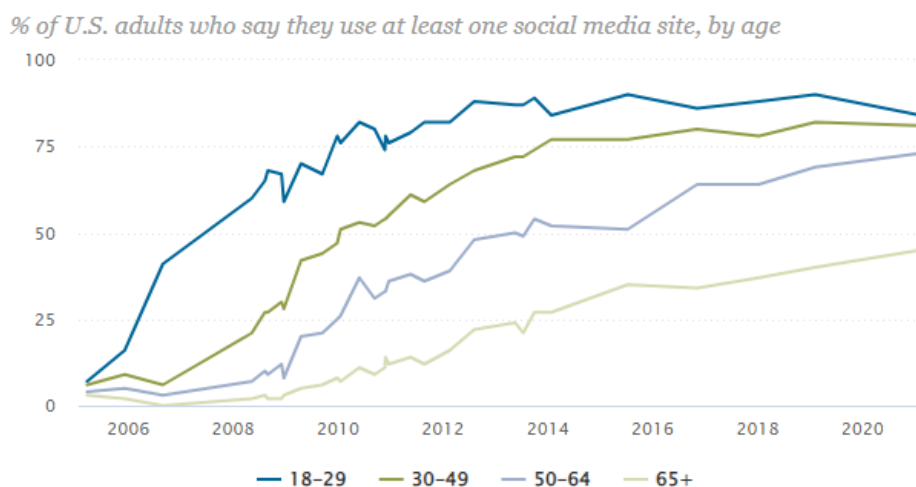
## 1.1 Cele i motywacje pracy

Celem pracy inżynierskiej jest zaprojektowanie i implementacja aplikacji mobilnej o nazwie „SeniorsApp” ułatwiającej seniorom nawiązywanie kontaktów towarzyskich. Główną motywacją do wyboru takiego tematu była chęć stworzenia skutecznego, lecz nie skomplikowanego narzędzia, pozwalającego osobom starszym na poznawanie osób w podobnym wieku i nawiązywanie relacji w bezpiecznych i komfortowych warunkach.

Samotność niestety jest powszechnym zjawiskiem w grupie wiekowej obejmującej osoby starsze. Ma ona niebagatelny wpływ na pogorszenie stanu zdrowia, zarówno fizycznego jak i psychicznego, oraz spadek ogólnego poziomu jakości życia. W ostatnich latach w wyniku trwającej obecnie pandemii COVID-19 zjawisko to szczególnie zyskało na sile. Ryzyko zakażenia i powikłań po chorobie wzrasta wraz z wiekiem, w związku z czym seniorzy, będąc najbardziej narażoną grupą są często izolowani od otoczenia. Wiele kiedyś oczywistych miejsc i okazji do spotkań nie jest już dostępnych ze względów bezpieczeństwa. Wśród nich można wymienić kluby seniora, koła gospodyń wiejskich, spotkania organizacji na rzecz kultury i sztuki, kluby sportowe, a nawet zwykłe spotkania sąsiedzkie. Pandemia skutecznie utrudnia utrzymywanie i nawiązywanie kontaktów towarzyskich. Sytuacja nie jest jednak beznadziejna, bowiem remedium na samotność może okazać się Internet.

Rozwój technologii oraz upowszechnienie się smartfonów otworzyło przed ludźmi ogrom nowych możliwości związanych z zawieraniem oraz utrzymywaniem znajomości. Jeżeli tylko mamy ochotę, możemy być w stałym kontakcie ze znajomymi i rodziną, niezależnie od odległości jaka nas dzieli. Oprócz rozwijania już istniejących relacji, za pośrednictwem Internetu możemy zawierać zupełnie nowe poznając ludzi, z którymi łączą nas wyznawane wartości, zainteresowania lub ulubione formy spędzania wolnego czasu. Obecnie, aby poszerzać kręgi towarzyskie nie ma nawet konieczności wychodzenia z domu.

W ostatnich latach spora część osób starszych aktywnie korzysta z możliwości, które oferuje Internet oraz chętnie udziela się w mediach społecznościowych.



Note: Respondents who did not give an answer are not shown.

Source: Surveys of U.S. adults conducted 2005-2021.

PEW RESEARCH CENTER

*Rysunek 1 Wartość procentowa dorosłych użytkowników mediów społecznościowych w Stanach Zjednoczonych ze względu na wiek.*

Ilość użytkowników mediów społecznościowych wśród seniorów z biegiem lat wciąż wzrasta, co można zaobserwować na powyższym wykresie. Mimo to przeglądając najpopularniejsze serwisy i aplikacje można odnieść wrażenie, że ich główną grupą docelową są młodzi ludzie, od dawna obcy z elektroniczną formą społecznych interakcji. Mały rozmiar czcionki, krzykliwe kolory oraz wyskakujące zewsząd powiadomienia nie stwarzają komfortowego środowiska dla seniorów nieprzywykłych do takiej dynamiki korzystania z Internetu, który dla wielu może być stosunkowo nowym doświadczeniem. Kolejnym mało przyjaznym aspektem może być rozbudowany i skomplikowany interfejs oraz fakt, że w dalszym ciągu dominującą grupą w większości mediów społecznościowych są młodzi ludzie. Może to sprawiać, że osoby starsze nie będą czuć się tam wystarczająco komfortowo. Na rynku wciąż brakuje oprogramowania dostosowanego do potrzeb tej coraz liczniejszej grupy użytkowników.

## 1.2 Założenia

Aby prawidłowo spełniać swoje zadanie aplikacja powinna umożliwiać założenie unikalnego konta użytkownika zabezpieczonego hasłem. Dzięki temu po zalogowaniu użytkownik będzie miał dostęp między innymi do wiadomości, które wymienił z innymi użytkownikami. Równie istotna jest możliwość wprowadzenia informacji o użytkowniku, tak, aby pozostali użytkownicy mogli się czegoś o nim dowiedzieć zanim zdecydują czy są zainteresowani kontaktem. Podczas użytkowania aplikacji część danych wprowadzonych przez użytkownika może ulec zmianie, w związku z czym należy stworzyć mu możliwość modyfikacji informacji wprowadzonych podczas rejestracji. Ponieważ każda osoba korzystająca z aplikacji ma prawo decydować o tym czy chce, aby jej dane były w dalszym ciągu przechowywane w bazie, należy stworzyć użytkownikowi możliwość usunięcia konta, wraz z wszystkimi przypisanymi do niego danymi. Aplikacja powinna również chronić użytkownika przed otrzymywaniem niepożądanych wiadomości od osób, z którymi nie chce on korespondować, a więc konieczna jest implementacja mechanizmu pozwalającego na wysyłanie wiadomości tylko do osób, które zaakceptowały i zostały zaakceptowane przez zalogowanego użytkownika. Z tego samego powodu przydatna może się okazać funkcja usuwania użytkownika z listy dopasowań, co skutkuje zablokowaniem możliwości dalszego wymieniać wiadomości z usuniętą osobą. W celu weryfikacji poprawności wprowadzonych danych, użytkownik będzie miał również możliwość wyświetlenia swojego profilu z aktualnie wprowadzonymi do bazy danych informacjami. Dla ułatwienia użytkownika zaimplementowany zostanie także algorytm wyświetlania informacji o braku połączenia z Internetem.

## 1.3 Zakres pracy

Zakres pracy obejmuje zaprojektowanie i implementację aplikacji mobilnej w języku Java, na platformę Android, przy pomocy platformy Firebase. Praca zawiera 4 rozdziały. W pierwszym rozdziale znajduje się opis powodów, dla których właśnie ten temat został wybrany, przybliżenie charakterystyki problemu i opis założeń, które powinna spełniać aplikacja. Rozdział drugi zawiera uzasadnienie wyboru technologii, listę niezbędnych funkcjonalności i opis sposobu w jaki aplikacja została dostosowana do potrzeb grupy odbiorców. Rozdział trzeci jest szczegółowym opisem implementacji każdej z najważniejszych funkcjonalności zawierającym zdjęcia interfejsu użytkownika i fragmenty kodu odpowiedzialne za kluczowe elementy logiki aplikacji. W rozdziale czwartym zawarto opis napotkanych problemów implementacyjnych, pomysły na funkcjonalności, które można zaimplementować w aplikacji, wnioski końcowe z pracy nad aplikacją oraz spis rysunków, listingów i literatury.

## 2. Projekt

W tym rozdziale zamieszczono listę niezbędnych funkcjonalności, które należy zaimplementować oraz uzasadnienie wyboru wykorzystanych technologii. Opisano również w jaki sposób aplikacja została dostosowana do potrzeb grupy docelowej.

### 2.1 Funkcjonalności

- Utworzenie nowego konta użytkownika przy pomocy emaila i hasła
- Walidacja długości i zgodności podanych haseł
- Sprawdzenie połączenia z Internetem
- Umieszczenie danych logowania w Firebase Authentication
- Pobranie informacji podanych przez użytkownika
- Walidacja danych podanych przez użytkownika
- Wybór zdjęcia z galerii
- Zrobienie zdjęcia przy pomocy aplikacji aparatu
- Załadowanie zdjęcia do Firebase Storage
- Utworzenie użytkownika w Firebase Realtime Database i zapis do bazy podanych przez niego informacji
- Zalogowanie do istniejącego konta
- Wylogowanie
- Podgląd profilu zalogowanego użytkownika
- Przeglądanie użytkowników pasujących do preferencji zalogowanego użytkownika
- Akceptacja lub odrzucenie wyświetlonego użytkownika
- Weryfikacja czy zaakceptowany użytkownik również zaakceptował zalogowanego użytkownika
- Dopasowanie wzajemnie zaakceptowanych użytkowników i dodanie ich do swoich list dopasowań
- Przeglądanie listy dopasowanych osób
- Przeglądanie profili dopasowanych osób
- Wysyłanie wiadomości do dopasowanych osób
- Odbieranie wiadomości od dopasowanych osób
- Usunięcie użytkownika z listy dopasowanych osób
- Edycja informacji o użytkowniku i jego preferencji
- Usunięcie konta i wszystkich informacji o użytkowniku z bazy danych

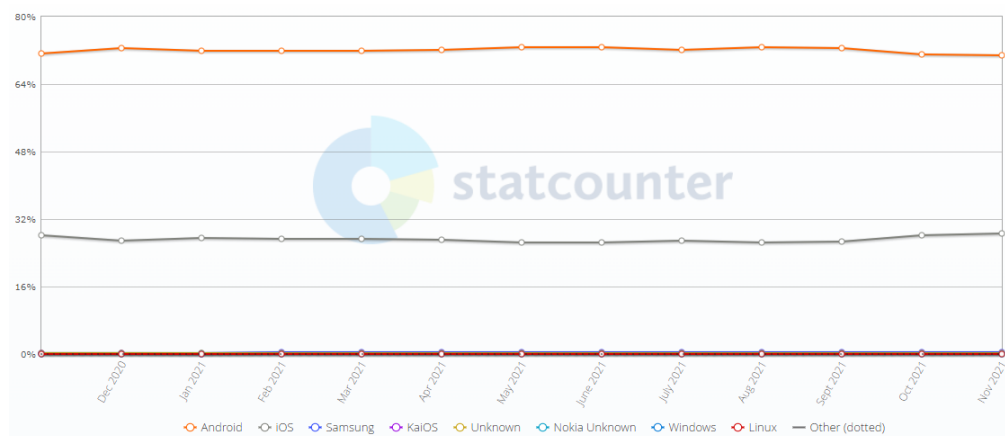
## 2.2 Wybór technologii

- **Java 8**

Przeważająca większość kodu źródłowego aplikacji została napisana w języku Java. Skłoniło mnie do tego osobiste zainteresowanie i chęć nauki tego języka. Jest on bardzo popularny między innymi ze względu na swoją uniwersalność, pozwala na pisanie aplikacji mobilnych, desktopowych, a także webowych.

- **Android**

Obecnie Android jest dominującym systemem operacyjnym na rynku urządzeń mobilnych. Według danych statystycznych z listopada 2021 roku w system ten wyposażono 70,74% przenośnych urządzeń elektronicznych.



Rysunek 2 Procentowy udział w rynku urządzeń mobilnych poszczególnych systemów operacyjnych.

Popularność Androida była jednym z czynników decydujących o wyborze właśnie tego systemu jako platformy do zaimplementowania aplikacji. Przy aplikacjach społecznościowych, w których interakcja polega na poznawaniu nowych osób, ilość użytkowników ma kluczowe znaczenie. Dlatego też ważne było, aby aplikacja była łatwo dostępna dla możliwie jak największej grupy użytkowników. Specjalnie jako minimalną wersję systemu (SDK) wybrano Android 7.0 Nougat (API 24), ponieważ dzięki temu będzie ona działać poprawnie aż na 89% urządzeń wyposażonych w system Android, a jednocześnie daje to dostęp do wielu rozwiązań programistycznych niedostępnych dla starszych wersji SDK. Innym ważnym atutem Androida jest prostota implementacji i testowania aplikacji, oraz łatwy dostęp do sprzętu. Dzięki temu, że zarówno ja jak i spora część mojej rodziny posiada urządzenia z Androidem, miałam możliwość testowania aplikacji na kilku różnych modelach smartfonów i wersjach SDK. Ponadto aplikacje natywne na tę platformę pisane przy użyciu dedykowanego środowiska Android Studio, które jest bardzo wygodnym narzędziem, pozwalającym na tworzenie części frontendowej aplikacji bez znajomości dodatkowych, związanych z tym języków jak np. JavaScript.

Android Studio pozwala również na korzystanie z podstawowych wbudowanych ikon i komponentów, oraz tworzenie własnych oraz całkowitą personalizację przycisków, przełączników i podobnych elementów, czego miałam okazję się nauczyć podczas implementacji aplikacji.

- **Firestore**

Jest to platforma programistyczna należąca do Google od 2014 roku. Pomaga ona tworzyć, ulepszać i rozwijać aplikacje. Jest to również tzw. BaaS czyli usługa Backendu, oferująca wiele usług pozwalających programistom na natychmiastową implementację podstawowych funkcjonalności. Najbardziej istotne w kontekście tej aplikacji to Authentication, Realtime Database i Storage. Zdecydowałam się na wykorzystanie takiego rozwiązania, ponieważ znacznie ułatwia ono pracę z danymi oraz uwierzytelnianie użytkowników. Dzięki takiemu udogodnieniu można bardziej skupić się na implementacji funkcjonalności będących bezpośrednio po stronie aplikacji.

## **2.3 Dostosowanie aplikacji do potrzeb grupy docelowej**

Ze względu na to, że aplikacja została zaprojektowana z myślą o osobach starszych, jej interfejs został dostosowany do specyficznych potrzeb tej grupy użytkowników. Specjalnie wybrana czcionka bezszeryfowa i duży rozmiar tekstu (między 28 a 50dp) zwiększają czytelność komunikatów wyświetlanych na ekranie. Nie bez znaczenia jest również paleta barw, która została dobrana w taki sposób, aby kolory nie męczyły wzroku. Bardzo istotny jest również kontrast pomiędzy kluczowymi elementami takimi jak tekst oraz przyciski, a tłem, który znacznie poprawia widoczność. Każdy przycisk został opatrzony odpowiednim napisem lub ikoną, aby jego funkcja w układzie była jasna dla użytkownika. Dzięki zastosowaniu fragmentów użytkownik ma wygodny dostęp do każdej z podstawowych funkcji aplikacji za pośrednictwem wysuwanego bocznego paska nawigacji, otwieranego za pomocą klasycznej ikony hamburgera.

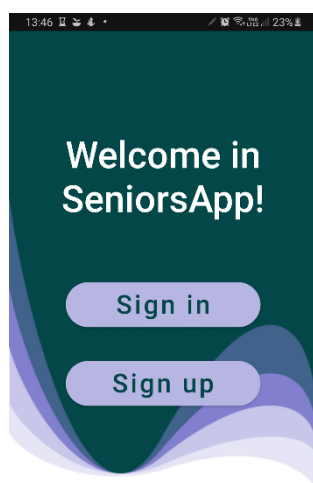


### 3. Implementacja

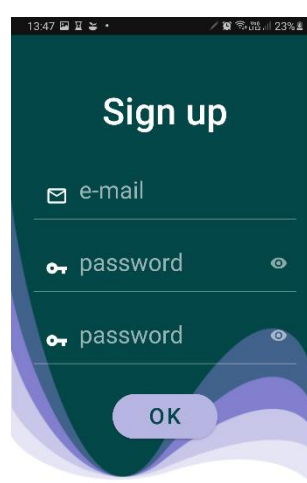
W rozdziale zawarto szczegółowy opis implementacji najważniejszych funkcjonalności. Zamieszczono również wiele zdjęć aktywności działającej aplikacji oraz kluczowych fragmentów kodu, aby jak najlepiej zaprezentować sposób w jaki działają.

#### 3.1 Rejestracja nowego użytkownika

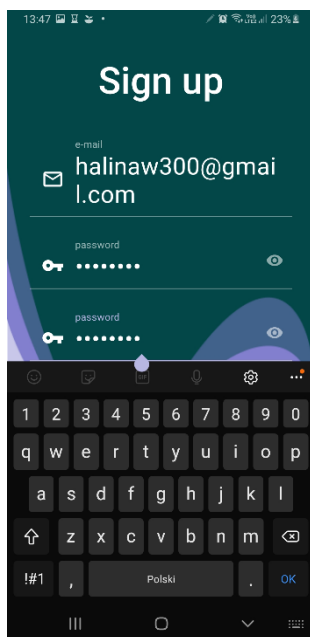
Jedną z najważniejszych funkcjonalności aplikacji jest rejestracja. Dzięki wykorzystaniu Firebase Authentication jest ona zdecydowanie uproszczona. Po włączeniu aplikacji pierwszą uruchamianą aktywnością jest `StartingActivity`. Wyświetla ona komunikat witający użytkownika i dwa przyciski, „Sign in” oraz „Sign up”. Naciśnięcie przycisku „Sign up” powoduje przejście do aktywności `RegistrationActivity`, w której znajdują się trzy edytowalne pola tekstowe (komponenty `EditText` umieszczone w `TextInputLayout`) oraz dwa przyciski (komponenty `Button`). Rejestracja jest wykonywana przy użyciu adresu e-mail oraz hasła, które musi zostać wprowadzone 2 razy w celu zminimalizowania pomyłki i wpisania hasła innego od tego, które użytkownik zapamiętał. Pola zawierające hasło można łatwo odsłonić, naciskając ikonę oka znajdującą się po prawej stronie każdego z nich. Taki efekt osiągnięto dzięki dodaniu do komponentu `EditText` właściwości `android:inputType="textPassword"` w pliku układu aktywności `activity_registration.xml`.



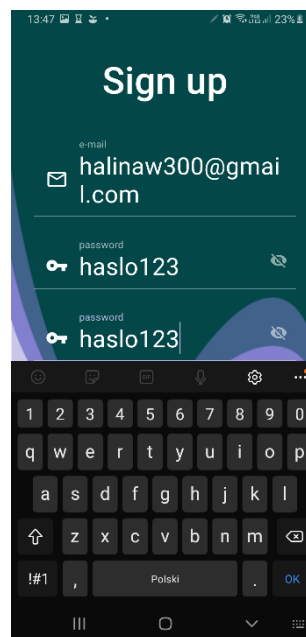
Rysunek 3 Aktywność `StartingActivity` uruchamiana przy włączeniu aplikacji.



Rysunek 4 Nowo uruchomiona aktywność `RegistrationActivity`.



Rysunek 5 Aktywność *RegistrationActivity* z wypełnionymi polami tekstowymi.



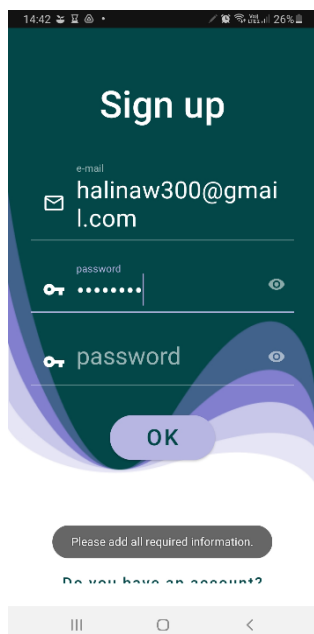
Rysunek 6 Aktywność *RegistrationActivity* po odsłonięciu wpisanego hasła.

Każda z wprowadzanych przez użytkownika informacji jest odpowiednio walidowana poprzez sprawdzenie czy została wprowadzona i czy spełnia postawione jej wymagania. E-mail musi mieć odpowiedni format zawierający znak „@” i znak „.”, a wpisane hasła muszą być identyczne i mieć co najmniej 8 znaków. Po wciśnięciu przycisku „OK” łańcuchy tekstowe są pobierane z komponentów `EditText` i przypisywane do odpowiednich zmiennych, następnie za pomocą instrukcji warunkowej `if...else` sprawdzane jest czy pola nie zostały pozostawione puste, czy hasło ma co najmniej 8 znaków i czy wprowadzone hasła są identyczne, jeżeli każde z wymagań zostało spełnione uruchamiana jest metoda `registerUser()`.

```
String email = emailEditText.getText().toString().trim();
String password = passwordEditText.getText().toString().trim();
String password2 = password2EditText.getText().toString().trim();

if(TextUtils.isEmpty(email) || TextUtils.isEmpty(password) ||
TextUtils.isEmpty(password2)) {
    Toast.makeText(getApplicationContext(), "Please add all required
information.", Toast.LENGTH_SHORT).show();
}else if(password.length() < 8) {
    Toast.makeText(getApplicationContext(), "Your password has to
contain at least 8 characters.", Toast.LENGTH_SHORT).show();
}else if(!password2.equals(password)) {
    Toast.makeText(getApplicationContext(), "Passwords do not match!",
Toast.LENGTH_SHORT).show();
}else {
    registerUser(email, password);
}
```

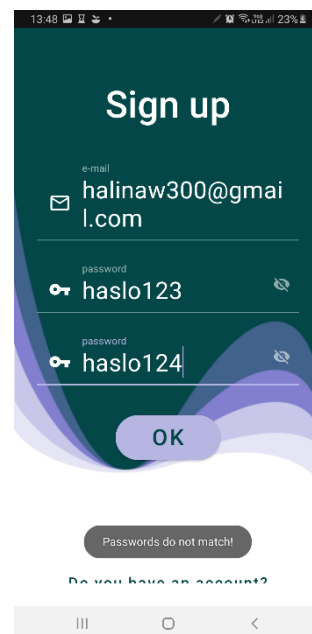
Listing 1 Walidacja informacji wprowadzonych przez użytkownika po naciśnięciu przycisku „OK”.



Rysunek 7 Wyświetlenie komunikatu związanego z niewypełnieniem jednego z pól.



Rysunek 8 Wyświetlenie komunikatu związanego z wprowadzeniem zbyt krótkiego hasła.



Rysunek 9 Wyświetlenie komunikatu związanego z wprowadzeniem różniących się od siebie haseł.

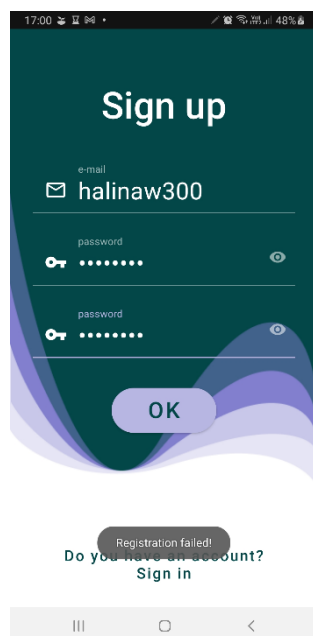
Metoda `registerUser` ma za zadanie zarejestrować użytkownika przy pomocy wprowadzonych przez niego adresu e-mail i hasła. Dzieje się to poprzez wywołanie metody `createUserWithEmailAndPassword()` wywołanej na obiekcie `auth` klasy `FirebaseAuth`. W przypadku niepowodzenia rejestracji z powodu na przykład próby rejestracji konta na już wykorzystany adres mailowy lub w przypadku wpisania adresu o niewłaściwym formacie, zostanie wyświetlony odpowiedni komunikat. Jeżeli operacja rejestracji użytkownika zostanie zakończona sukcesem, wówczas wprowadzone przez niego dane zostaną zapakowane w obiekt klasy `Bundle` i wysłane wraz z intencją do nowej aktywności `UserInfoActivity`.

```

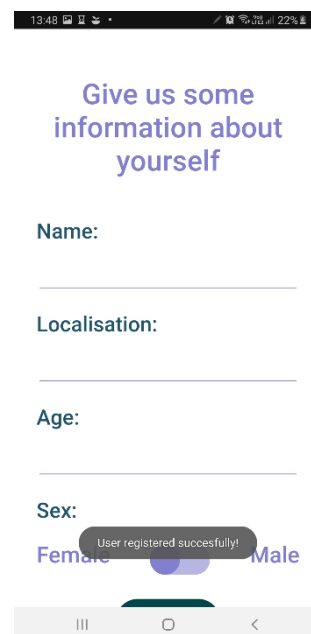
private void registerUser(String email, String password) {
    auth.createUserWithEmailAndPassword(email,
password).addOnCompleteListener(RegistrationActivity.this, new
OnCompleteListener<AuthResult>() {
    @Override
    public void onComplete(@NonNull Task<AuthResult> task) {
        if(task.isSuccessful()) {
            Toast.makeText(getApplicationContext(), "User registered
succesfully!", Toast.LENGTH_SHORT).show();
            Intent registerIntent = new
Intent(getApplicationContext(), UsersInfoActivity.class);
            Bundle registerBundle = new Bundle();
            registerBundle.putString("email", email);
            registerBundle.putString("password", password);
            registerIntent.putExtras(registerBundle);
            startActivity(registerIntent);
            finish();
        }else{
            Toast.makeText(getApplicationContext(), "Registration
failed!", Toast.LENGTH_SHORT).show();
        }
    }
});
}

```

Listing 2 Definicja funkcji registerUser().

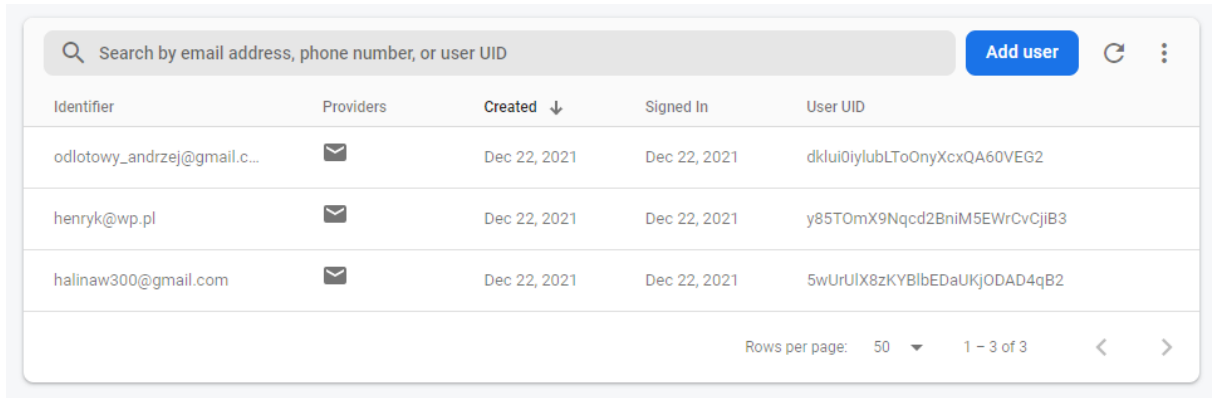


Rysunek 10 Niepowodzenie rejestracji z powodu nieprawidłowego formatu adresu e-mail.



Rysunek 11 Efekt udanej rejestracji użytkownika, po której nastąpiło wyświetlenie odpowiedniego komunikatu i przejście do aktywności UsersInfoActivity.

Po prawidłowym zarejestrowaniu użytkownika pojawia się on w tabeli w Firebase Authentication. Jako jego identyfikator zostaje wykorzystany podany przez niego adres email. W kolumnie Providers wyświetlany jest sposób rejestracji, w tym przypadku przy pomocy adresu e-mail i hasła. W tabeli znajdują się również takie informacje jak data utworzenia konta, data ostatniego zalogowania i nadany użytkownikowi wygenerowany przez Firebase UID.

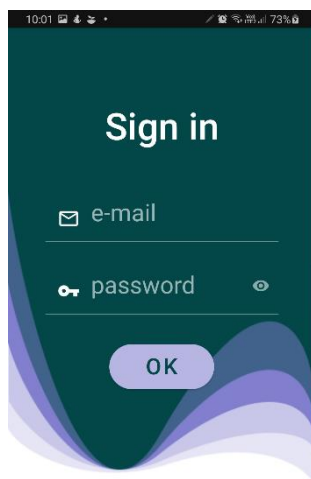


Identifier	Providers	Created ↓	Signed In	User UID
odlotowy_andrzej@gmail.c...	✉	Dec 22, 2021	Dec 22, 2021	dklui0iylubLToOnyXcxQA60VEG2
henryk@wp.pl	✉	Dec 22, 2021	Dec 22, 2021	y85T0mX9Nqcd2BniM5EWrcvCjiB3
halinaw300@gmail.com	✉	Dec 22, 2021	Dec 22, 2021	5wUrUIX8zKYBlbEDaUKjODAD4qB2

Rysunek 12 Zrzut ekranu interfejsu użytkownika Firebase Authentication.

## 3.2 Logowanie i wylogowanie

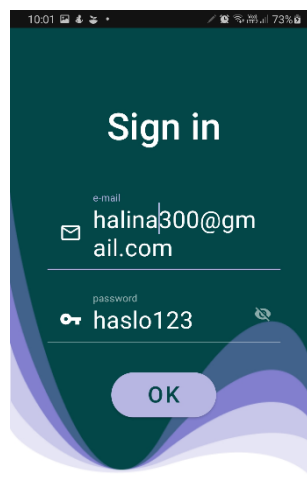
Logowanie do konta użytkownika odbywa się w aktywności LogInActivity, do której można się dostać poprzez naciśnięcie przycisku „Sign in” w aktywności StartingActivity lub naciśnięcie przycisku „Do you have an account? Sign in” w aktywności RegistrationActivity. W analogiczny sposób można dostać się do aktywności RegistrationActivity, poprzez naciśnięcie przycisku „Are you not registered? Sign up” w LogInActivity lub naciśnięcie przycisku „Sign up” znajdującego się w aktywności StartingActivity. Aktywność LogInActivity zawiera dwa edytowalne pola tekstowe, pierwsze z nich jest przeznaczone na adres e-mail, a drugie na hasło. Podobnie jak w aktywności RegistrationActivity, wpisane hasło można łatwo odsłonić, aby zweryfikować jego poprawność, poprzez naciśnięcie ikony oka po prawej stronie pola tekstowego.



Are you not registered?  
Sign up



Rysunek 13 Aktywność  
LogInActivity.



Incorrect email or password!  
Are you not registered?  
Sign up



Rysunek 14 Efekt nieudanej  
rejestracji, z powodu wpisania  
danych, z powodu wpisania  
danych, na które nie zostało  
zarejestrowane żadne konto  
użytkownika.

Po wciśnięciu przycisku „OK” następuje pobranie danych wprowadzonych przez użytkownika i przypisanie ich do odpowiednich zmiennych, a następnie wywołanie metody `loginUser()`.

```
String email = emailEditText.getText().toString().trim();
String password = passwordEditText.getText().toString().trim();
loginUser(email, password);
```

Listing 3 Przypisanie danych wprowadzonych do odpowiednich zmiennych i wywołanie metody `loginUser()`.

Metoda `loginUser()` ma za zadanie zalogować użytkownika do stworzonego przez niego wcześniej konta, przy pomocy wprowadzonych przez niego adresu e-mail i hasła. Dzieje się to poprzez wywołanie metody `signInWithEmailAndPassword()` wywołanej na obiekcie `auth` klasy `FirebaseAuth`. W przypadku niepowodzenia logowania, zostanie wyświetlony odpowiedni komunikat. Jeżeli operacja zalogowania użytkownika zostanie zakończona sukcesem zostanie wyświetlony informujący o tym komunikat i nastąpi przejście do aktywności `NavigationActivity`.

```

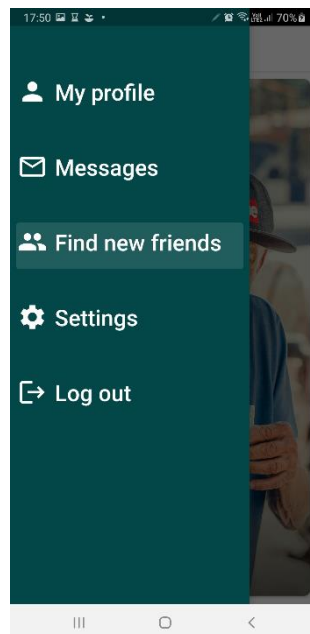
private void loginUser(String email, String password) {
    auth.signInWithEmailAndPassword(email,
password).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
        @Override
        public void onComplete(@NonNull Task<AuthResult> task) {
            if(task.isSuccessful()) {
                Toast.makeText(getApplicationContext(), "Signing in
succesfull!", Toast.LENGTH_SHORT).show();
                Intent intent = new Intent(getApplicationContext(),
NavigationActivity.class);
                startActivity(intent);
                finish();
            }else{
                Toast.makeText(getApplicationContext(), "Incorrect email
or password!", Toast.LENGTH_SHORT).show();
            }
        }
    });
}

```

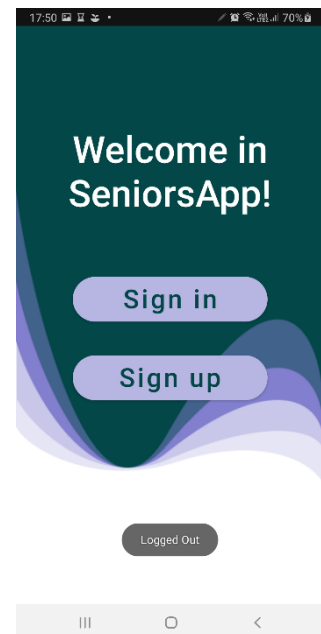
Listing 4 Definicja metody loginUser().



Rysunek 15 Aktywność NavigationActivity wyświetlona po poprawnym zalogowaniu.



Rysunek 16 Pasek boczny nawigacji w aktywności NavigationActivity.



Rysunek 17 Przejście do aktywności StartingActivity po udanym wylogowaniu użytkownika oraz wyświetlenie potwierdzającego to komunikatu.

Aby się wylogować użytkownik musi nacisnąć ikonę hamburgera, znajdującą się w lewym górnym rogu ekranu. Dzięki takiemu działaniu zostanie rozwinięty boczny pasek nawigacji, w którym ostatnia pozycja odpowiada za wylogowanie użytkownika. W momencie wybrania pozycji „Log out” z menu, zostanie wywołana metoda `logout()`, której zarówno wywołanie jak i definicja znajduje się w pliku `NavigationActivity.java`. Wylogowanie następuje poprzez wywołanie metody `signOut()` na instancji klasy `FirebaseAuth`, która dostarczona jest przez `Firebase`. Po wylogowaniu wyświetlony zostaje potwierdzający je komunikat i aplikacja przechodzi do aktywności `StartingActivity`.

```
public void logout () {
    FirebaseAuth.getInstance().signOut();
    Toast.makeText(getApplicationContext(), "Logged Out",
    Toast.LENGTH_SHORT).show();
    Intent intent = new Intent(NavigationActivity.this,
    StartingActivity.class);
    startActivity(intent);
}
```

*Listing 5 Definicja metody `logout()`.*

```
switch (item.getItemId()) {
    case R.id.nav_my_profile:

        getSupportFragmentManager().beginTransaction().replace(R.id.fragment_container, new MyProfileFragment()).commit();
        break;
    case R.id.nav_messages:

        getSupportFragmentManager().beginTransaction().replace(R.id.fragment_container, new MessagesFragment()).commit();
        break;
    case R.id.nav_find_friends:

        getSupportFragmentManager().beginTransaction().replace(R.id.fragment_container, new FindNewFriendsFragment()).commit();
        break;
    case R.id.nav_settings:

        getSupportFragmentManager().beginTransaction().replace(R.id.fragment_container, new SettingsFragment()).commit();
        break;
    case R.id.nav_logout:
        logout();
        break;
}
```

*Listing 6 Wywołanie metody `logout()` w przypadku wybrania elementu o id `nav_logout` w pliku `NavigationActivity.java` wewnątrz metody `onNavigationItemSelectedListener`.*



### 3.3 Sprawdzanie połączenia z Internetem

Aplikacja wymaga połączenia z Internetem do wykonywania wszystkich operacji związanych z przesyłaniem lub wyświetlaniem danych. Takie operacje stanowią elementy niemal wszystkich funkcjonalności aplikacji. W związku z tym, dla wygody użytkownika zaimplementowano mechanizm sprawdzania połączenia, który w przypadku jego braku, natychmiast wyświetla informujący o tym komunikat. Wyświetlanie komunikatu o braku połączenia z Internetem okazało się również bardzo przydatne podczas pracy nad aplikacją i jej testowaniem, ponieważ gdy aplikacja nie działała prawidłowo ze względu na brak połączenia, od razu stawało się to jasne. Zaoszczędziło mi to wiele czasu spędzonego na szukaniu błędu w napisanym kodzie, podczas gdy problemem był brak połączenia z Internetem. W momencie podjęcia przez użytkownika próby rejestracji poprzez naciśnięcie przycisku „OK” zostaje wywołana metoda `isConnected()`, która sprawdza połączenie z Internetem i zwraca wartość `true` w sytuacji gdy urządzenie ma dostęp do sieci lub wartość `false`, kiedy urządzenie takiego dostępu nie posiada. Wywołanie tej metody zostało umieszczone w warunku instrukcji warunkowej `if...else`. Dzięki takiemu rozwiązaniu zaistniała możliwość zróżnicowania działania programu w zależności od połączenia z siecią, czyli wartości zwróconej przez metodę `isConnected()`. Dzięki wywołaniu metody `getActiveNetworkInfo()` na obiekcie klasy `ConnectivityManager` możliwe jest sprawdzenie połączenia urządzenia z Internetem. Jeżeli obiekt klasy `NetworkInfo`, do którego przypisano to wywołanie, jest różny od `null` i wywołanie metody `isConnected()` na tym obiekcie zwraca wartość `true` to oznacza, że urządzenie jest połączone z siecią, w przeciwnym wypadku wiadomo, że nie jest połączone.

```
public boolean isConnected() {
    ConnectivityManager connectivityManager = (ConnectivityManager)
getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo =
connectivityManager.getActiveNetworkInfo();

    if (networkInfo != null) {
        if (networkInfo.isConnected()) {
            return true;
        } else {
            return false;
        }
    } else {
        return false;
    }
}
```

*Listing 7 Definicja metody `isConnected()` w pliku `RegistrationActivity.java`.*

```

registerBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

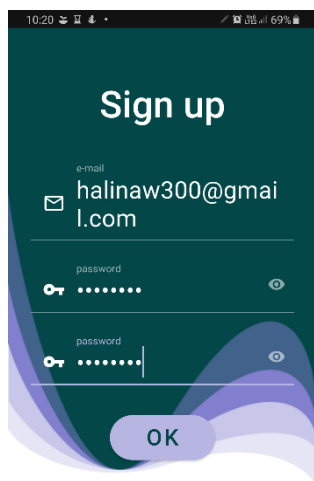
        if (isConnected()) {
            String email = emailEditTxt.getText().toString().trim();
            String password =
passwordEditTxt.getText().toString().trim();
            String password2 =
password2EditTxt.getText().toString().trim();

            if(TextUtils.isEmpty(email) ||
TextUtils.isEmpty(password) || TextUtils.isEmpty(password2)) {
                Toast.makeText(getApplicationContext(), "Please add
all required information.", Toast.LENGTH_SHORT).show();
            }else if(password.length() < 8) {
                Toast.makeText(getApplicationContext(), "Your
password has to contain at least 8 characters.",
Toast.LENGTH_SHORT).show();
            }else if(!password2.equals(password)) {
                Toast.makeText(getApplicationContext(), "Passwords
do not match!", Toast.LENGTH_SHORT).show();
            }else {
                registerUser(email, password);
            }
        } else {
            Intent i = new Intent(getApplicationContext(),
RegisterInternetAccessActivity.class);
            startActivity(i);
            finish();
        }
    }
});

```

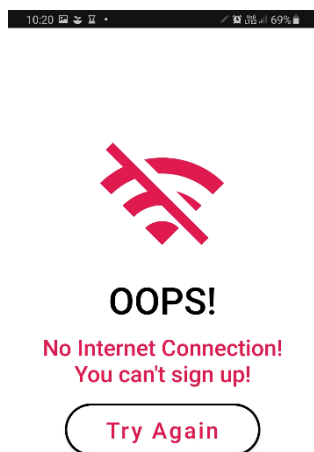
*Listing 8 Kod wykonywany w następstwie naciśnięcia przycisku OK w aktywności RegistrationActivity i wywołanie metody isConnected() w setOnClickListener() wywołanym na przycisku w pliku RegistrationActivity.java*

W sytuacji, w której metoda isConnected() wywołana w warunku instrukcji warunkowej if...else zwraca true program przechodzi do walidacji wprowadzonych przez użytkownika danych, a następnie do rejestracji użytkownika. Natomiast gdy metoda isConnected() zwróci wartość false, tworzona jest nowa intencja, za pomocą której aplikacja przechodzi do aktywności RegisterInternetAccessActivity.

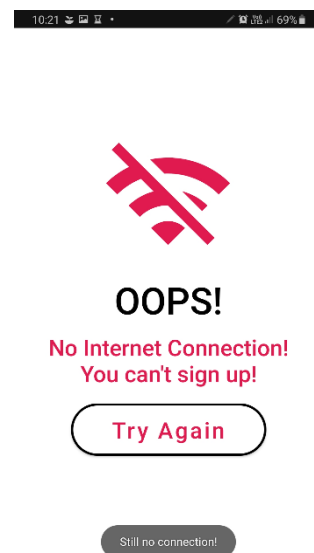


Do you have an account?

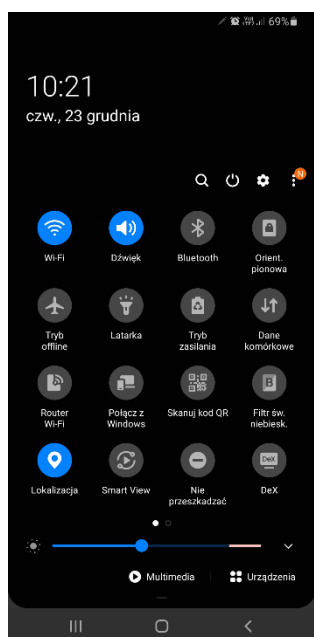
Rysunek 18 Aktywność *RegistrationActivity* przed wciśnięciem przycisku „OK”.



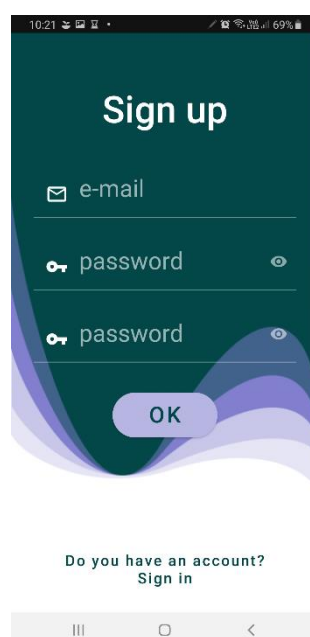
Rysunek 19 Aktywność *RegisterInternetAccessActivity*.



Rysunek 20 Wyświetlenie komunikatu o dalszym braku połączenia po wciśnięciu przycisku „Try Again”.



Rysunek 21 Włączenie Wi-Fi i tym samym połączenie urządzenia z Internetem.



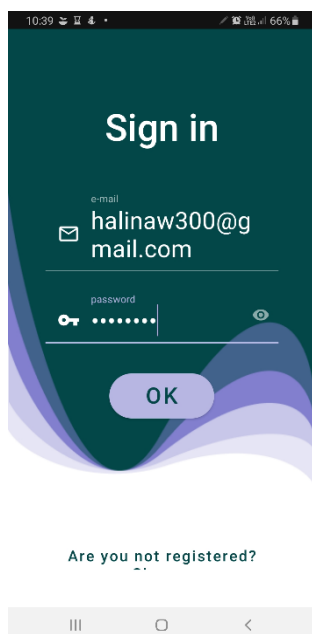
Do you have an account?  
Sign in

Rysunek 22 Przejście do aktywności *RegistrationActivity* po wciśnięciu przycisku Try Again i potwierdzeniu połączenia z siecią.

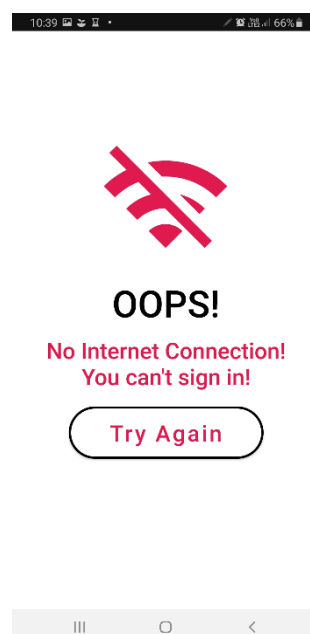
```
tryAgainBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if (isConnected()) {
            Intent intent = new Intent(getApplicationContext(),
RegistrationActivity.class);
            startActivity(intent);
            finish();
        } else {
            Toast.makeText(getApplicationContext(), "Still no
connection!", Toast.LENGTH_SHORT).show();
        }
    }
});
```

Listing 9 Kod wykonywany w następstwie naciśnięcia przycisku Try Again w pliku RegisterInternetAccessActivity.java

Sprawdzanie połączenia w aktywności logowania LogInActivity rozwiązano w sposób analogiczny do powyższego. W przypadku braku połączenia z Internetem aplikacja przechodzi do aktywności LoginInternetAccessActivity, działającej analogicznie do RegisterInternetAccessActivity.



Rysunek 23 Aktywność LogInActivity.



Rysunek 24 Aktywność LoginInternetAccessActivity.

### 3.4 Pobieranie informacji o użytkowniku

Zbieranie informacji o użytkowniku odbywa się w trzech różnych aktywnościach: `UserInfoActivity`, `PreferencesActivity` i `ProfileInfoActivity`. W każdej z nich zbierane są dane, które później zostaną przesłane do bazy danych `Firebase Realtime Database`. W aktywności `UserInfoActivity` znajdują się trzy komponenty `EditText`, w których należy wprowadzić kolejno imię, lokalizację i wiek, komponent `Switch`, za pomocą którego użytkownik może wybrać swoją płeć oraz przycisk „OK” służący do zatwierdzenia wprowadzonych danych i przejścia do następnej aktywności. Każde z pól tekstowych jest wyposażone w walidację.

W momencie uruchomienia aktywności `UserInfo` przesłane za pomocą `Bundle` i intencji łańcuchy tekstowe zawierające email i hasło są przypisywane do zmiennych w metodzie `onCreate()`. Za pomocą `listenera` sprawdzane jest czy przełącznik zmienił pozycję i na tej podstawie ustawiana jest wartość zmiennej `sex`, której domyślna wartość jest ustawiona na „female”, ponieważ taką wartość reprezentuje `Switch` w swoim położeniu startowym.

```
Intent registerIntent = getIntent();
Bundle registerBundle = registerIntent.getExtras();
String email = registerBundle.getString("email");
String password = registerBundle.getString("password");
```

*Listing 10 Pobranie danych z `Bundle` i przypisanie ich do zmiennych `email` i `password`.*

```
sexSwitch.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean
isChecked) {
        if (isChecked) {
            sex = "male";
        } else {
            sex = "female";
        }
    }
});
```

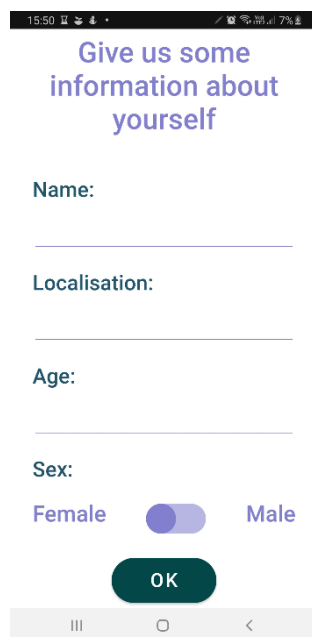
*Listing 11 Listener ustawiony na przełączniku.*

W chwili naciśnięcia przycisku informacje podane w polach tekstowych są zapisywane do odpowiednich zmiennych, a następnie przeprowadzana jest walidacja każdej z nich, co sprawia, że w przypadku nie uzupełnienia któregośkolwiek z pól, podania imienia lub nazwy miejscowości zawierającej liczbę, podania niemożliwego (ujemnego lub powyżej 130) lub niezgodnego z polityką aplikacji wieku (poniżej 18) zostanie wyświetlony odpowiedni komunikat. Walidacja przeprowadzana jest przy pomocy instrukcji warunkowych if...else, obsługi błędów try...catch i specjalnie napisanej w tym celu metody containsLettersOnly(), której zadaniem jest sprawdzenie czy łańcuch podany w argumencie zawiera tylko litery.

```
public boolean containsLettersOnly(String text) {
    char[] textCharacters = text.toCharArray();
    boolean isIt = true;

    for (char character : textCharacters) {
        if (!Character.isLetter(character)) {
            Log.d("VALID", character + " is NOT letter");
            isIt = false;
            return isIt;
        }
        Log.d("VALID", character + " is letter");
    }
    return isIt;
}
```

Listing 12 Definicja metody containsLettersOnly() w pliku UsersInfoActivity.java



15:50

Give us some information about yourself

Name:

Localisation:

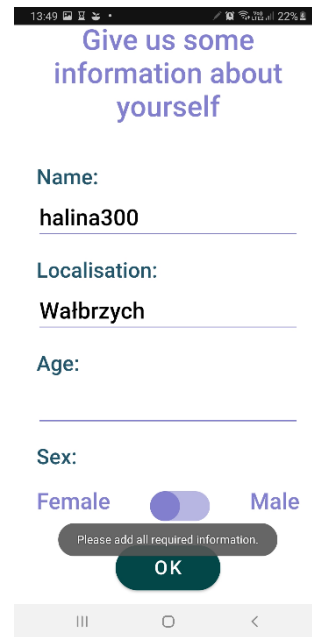
Age:

Sex:

Female Male

OK

Rysunek 25 Aktywność UsersInfoActivity.



13:49

Give us some information about yourself

Name: halina300

Localisation: Wałbrzych

Age:

Sex: Female Male

Please add all required information.

OK

Rysunek 26 Wyświetlenie komunikatu o niewypełnieniu jednego z pól.

```

String name = nameEditText.getText().toString().trim();
String localisation =
localisationEditText.getText().toString().trim();
ageFormat = true;

try {
    age = Integer.parseInt(ageEditText.getText().toString().trim());
} catch (NumberFormatException exception) {
    if (age != 0) {
        ageFormat = false;
    }
}

if (ageFormat) {
    if (name.isEmpty() || localisation.isEmpty() ||
ageEditText.getText().toString().isEmpty()) {
        Toast.makeText(getApplicationContext(), "Please add all
required information.", Toast.LENGTH_SHORT).show();
    } else {
        if (containsLettersOnly(name) &&
containsLettersOnly(localisation)) {
            if (age >= 130 || age < 18) {
                Toast.makeText(getApplicationContext(), "Insert
correct age, it must be greater or equal to 18.",
Toast.LENGTH_LONG).show();
            } else {
                name = name.substring(0, 1).toUpperCase() +
name.substring(1).toLowerCase();
                localisation = localisation.substring(0,
1).toUpperCase() + localisation.substring(1).toLowerCase();

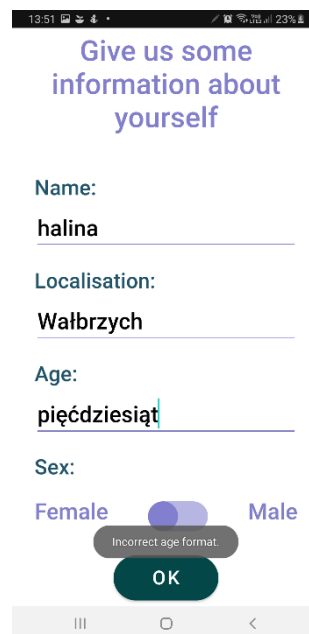
                Intent infoIntent = new
Intent(getApplicationContext(), PreferencesActivity.class);
                Bundle infoBundle = new Bundle();
                infoBundle.putString("email", email);
                infoBundle.putString("password", password);
                infoBundle.putString("name", name);
                infoBundle.putString("localisation", localisation);
                infoBundle.putInt("age", age);
                infoBundle.putString("sex", sex);
                infoIntent.putExtras(infoBundle);
                startActivity(infoIntent);
                finish();
            }
        } else {
            Toast.makeText(getApplicationContext(), "Name and
localisation must contain only letters.", Toast.LENGTH_LONG).show();
        }
    }
} else {
    Toast.makeText(getApplicationContext(), "Incorrect age format.",
Toast.LENGTH_LONG).show();
}
}

```

Listing 13 Walidacja danych wprowadzonych przez użytkownika i przesłanie jej za pomocą Bundle do aktywności PreferencesActivity.

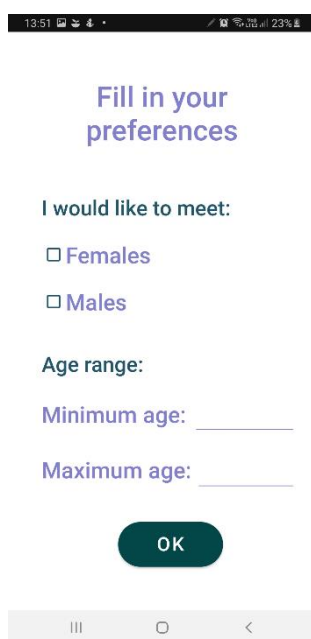


Rysunek 27 Wyświetlenie komunikatu o wpisaniu cyfr w pole, w którym powinny znajdować się tylko litery.

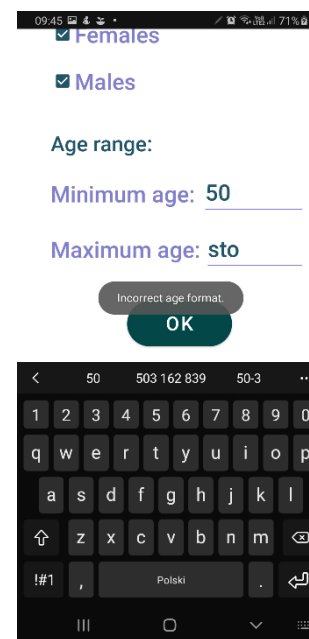


Rysunek 28 Wyświetlenie komunikatu o wpisaniu wieku o nieprawidłowym formacie.

Jeżeli wszystkie podane przez użytkownika informacje spełniają stawiane warunki, łańcuchy reprezentujące imię i lokalizację są modyfikowane tak, aby pierwsza litera łańcucha była wielką literą, a pozostałe małymi. Następnie wszystkie zgromadzone dotychczas informacje o użytkowniku są przesyłane do aktywności PreferencesActivity przy pomocy Bundle. Aplikacja uruchamia nową aktywność PreferencesActivity.



Rysunek 29 Aktywność PreferencesActivity.



Rysunek 30 Wyświetlenie komunikatu w związku z podaniem wieku w niewłaściwym formacie.

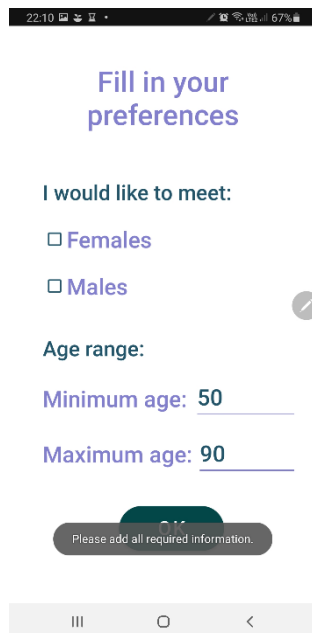


Aktywność `PreferencesActivity` zawiera dwa komponenty `CheckBox` reprezentujące płeć interesujących użytkownika osób, dwa komponenty `EditText`, w które należy wprowadzić minimalny i maksymalny wiek wyświetlanych użytkowników i przycisk służący do zatwierdzenia wprowadzonych danych. W momencie uruchomienia aktywności wszystkie dotychczas zebrane informacje o użytkowniku są zapisywane w zmiennych. Dzieje się to tak samo jak w poprzedniej aktywności.

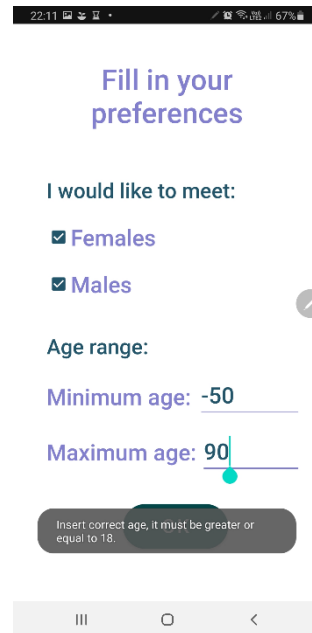
```
Intent infoIntent = getIntent();
Bundle infoBundle = infoIntent.getExtras();
String email = infoBundle.getString("email");
String password = infoBundle.getString("password");
String name = infoBundle.getString("name");
String localisation = infoBundle.getString("localisation");
int age = infoBundle.getInt("age");
String sex = infoBundle.getString("sex");
```

Listing 14 Przypisanie dotychczas zebranych informacji o użytkownikach do odpowiednich zmiennych.

W chwili, w której zostanie wciśnięty przycisk „OK” rozpoczyna się walidacja wpisanych informacji i sprawdzenie zaznaczenia `CheckBox`ów. Jeżeli któraś z informacji nie została wprowadzona wyświetlony zostanie informujący o tym komunikat. Jeżeli wpisany wiek jest w niewłaściwym formacie, ma niemożliwą lub niezgodną z polityką aplikacji wartość, zostanie wyświetlony odpowiedni komunikat.



Rysunek 31 Wyświetlenie komunikatu w wyniku niedostarczenia wszystkich wymaganych informacji przez użytkownika.



Rysunek 32 Wyświetlenie komunikatu informującego o nieprawidłowej wartości wprowadzonego wieku

```

try {
    minAge =
Integer.parseInt(minAgeEditText.getText().toString().trim());
    maxAge =
Integer.parseInt(maxAgeEditText.getText().toString().trim());
} catch (NumberFormatException exception) {
    if (age != 0) {
        Toast.makeText(getApplicationContext(), "Incorrect age
format.", Toast.LENGTH_LONG).show();
    }
}

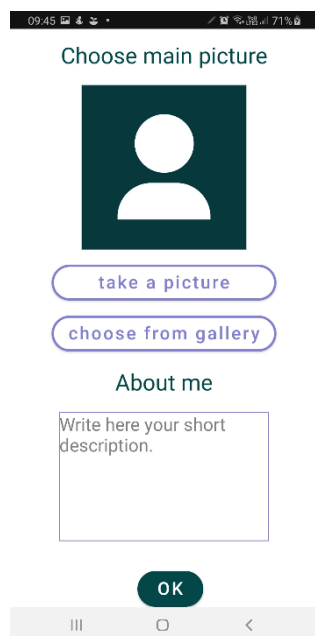
if ((!femalesCB.isChecked() && !malesCB.isChecked()) ||
minAgeEditText.getText().toString().isEmpty() ||
maxAgeEditText.getText().toString().isEmpty()){
    Toast.makeText(getApplicationContext(), "Please add all required
information.", Toast.LENGTH_SHORT).show();
} else {
    if (minAge < 18 || maxAge < 18 || minAge > 130 || maxAge > 130 ||
minAge >= maxAge) {
        Toast.makeText(getApplicationContext(), "Insert correct age,
it must be greater or equal to 18.", Toast.LENGTH_SHORT).show();
    } else {
        if (femalesCB.isChecked() && malesCB.isChecked()) {
            preferredSex = "both";
        } else if (femalesCB.isChecked() && !malesCB.isChecked()) {
            preferredSex = "female";
        } else {
            preferredSex = "male";
        }

        Intent prefIntent = new Intent(getApplicationContext(),
ProfileInfoActivity.class);
        Bundle prefBundle = new Bundle();
        prefBundle.putString("email", email);
        prefBundle.putString("password", password);
        prefBundle.putString("name", name);
        prefBundle.putString("localisation", localisation);
        prefBundle.putInt("age", age);
        prefBundle.putString("sex", sex);
        prefBundle.putString("preferredSex", preferredSex);
        prefBundle.putInt("minAge", minAge);
        prefBundle.putInt("maxAge", maxAge);
        prefIntent.putExtras(prefBundle);
        startActivity(prefIntent);
        finish();
    }
}
}

```

*Listing 15 Walidacja danych wprowadzonych przez użytkownika i wysłanie ich do aktywności ProfileInfoActivity*

Jeżeli wszystkie wprowadzone przez użytkownika informacje spełniają postawione im wymagania uruchamiana jest nowa aktywność ProfileInfoActivity i przesyłane są do niej wszystkie dotychczas zebrane dane.



Rysunek 33 Aktywność ProfileInfoActivity.



Rysunek 34 Aktywność ProfileInfoActivity z poprawnie wprowadzonymi danymi.

Kiedy aktywność ProfileInfoActivity zostanie uruchomiona informacje przesłane z aktywności PreferencesActivity zostają przypisane do odpowiednich zmiennych. Aktywność ta zawiera komponent ImageView umożliwiający podgląd wybranego zdjęcia, pole tekstowe, w które należy wpisać opis użytkownika, przycisk „OK” zatwierdzający wprowadzone informacje oraz dwa przyciski służące do wyboru zdjęcia, których dokładniejszym omówieniem zajmę się w następnym podrozdziale. Kiedy po wybraniu zdjęcia i wprowadzeniu opisu zostanie wciśnięty przycisk „OK”, następuje sprawdzenie czy został wpisany opis i czy jego długość nie przekracza 250 znaków. Jeżeli opis nie został wpisany lub jest niepoprawnej długości zostanie wyświetlony odpowiedni komunikat. W przeciwnym wypadku przy pomocy wszystkich dotychczas wprowadzonych danych i UID zalogowanego użytkownika zostanie utworzony nowy obiekt klasy User. Przy życiu tego obiektu i metody addUserToDB() klasy DataBaseHelper, dane użytkownika zostaną wprowadzone do bazy Firebase Realtime Database. Jeżeli operacja ta zakończy się powodzeniem, zostanie utworzona nowa intencja i aplikacja przejdzie do aktywności NavigationActivity.

```

String description = descriptionEditText.getText().toString().trim();

if (description.isEmpty()) {
    Toast.makeText(getApplicationContext(), "Description is
required", Toast.LENGTH_SHORT).show();
} else {
    if (description.length() > 250) {
        Toast.makeText(getApplicationContext(), "Description is too
long.", Toast.LENGTH_SHORT).show();
    } else {
        userAge = Integer.toString(age);
        userMinAge = Integer.toString(minAge);
        userMaxAge = Integer.toString(maxAge);
        user = new User(dbHelper.getCurrentUserId(), email, password,
name, userAge, sex, localisation, preferredSex, description,
fileName, imageUri, userMinAge, userMaxAge);
        dbHelper.addUserToDB(user).addOnSuccessListener(success->
        {
            Toast.makeText(getApplicationContext(), "User added to
database!", Toast.LENGTH_SHORT).show();
            Intent navIntent = new Intent(getApplicationContext(),
NavigationActivity.class);
            startActivity(navIntent);
            finish();
        }).addOnFailureListener(error->
        {
            Toast.makeText(getApplicationContext(), "" +
error.getMessage(), Toast.LENGTH_SHORT).show();
        });
    }
}
}

```

Listing 17 Walidacja opisu wprowadzonego przez użytkownika, utworzenie obiektu, wywołanie metody addUserToDB() na obiekcie klasy DataBaseHelper.

```

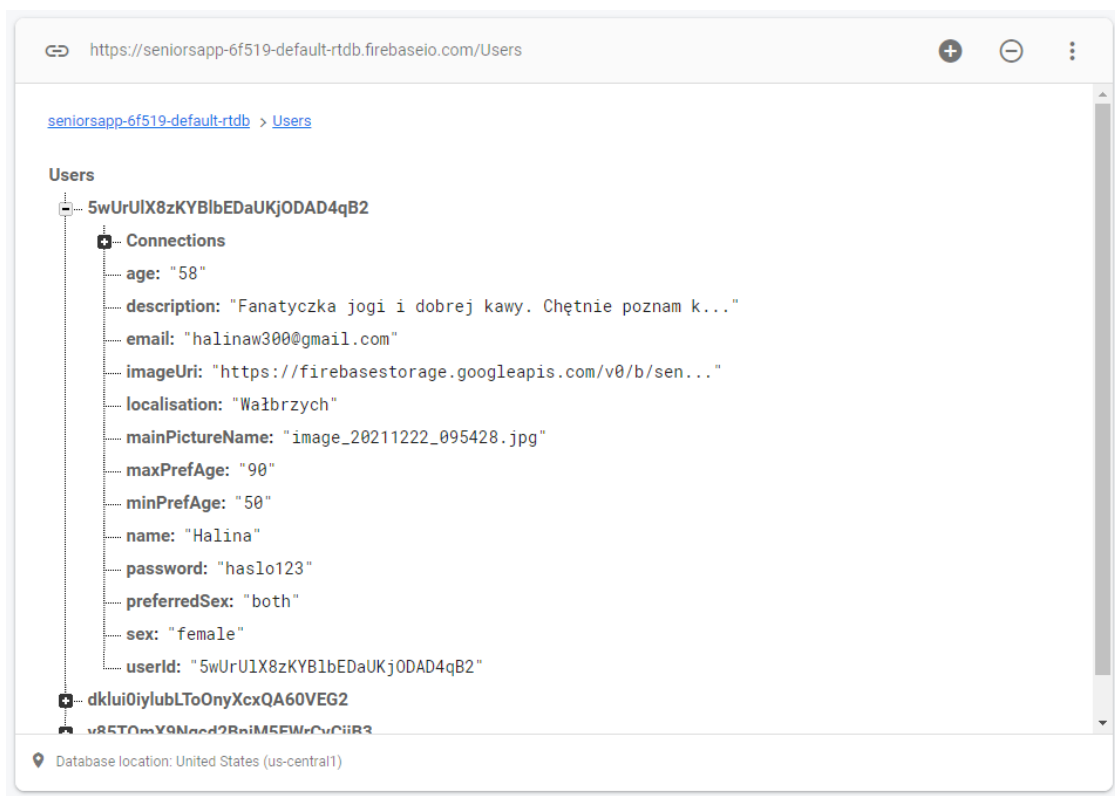
private FirebaseDatabase db = FirebaseDatabase.getInstance();
private DatabaseReference databaseReference = db.getReference();
private String currentUserId =
FirebaseAuth.getInstance().getCurrentUser().getUid();
private DatabaseReference currentUserReference =
databaseReference.child("Users").child(currentUserId);
private StorageReference storageReference =
FirebaseStorage.getInstance().getReference();

public DataBaseHelper() {
}

public Task<Void> addUserToDB(User user) {
    return
databaseReference.child("Users").child(currentUserId).setValue(user);
}

```

Listing 16 Składowe, konstruktor i definicja metody addUserToDB() klasy DataBaseHelper.



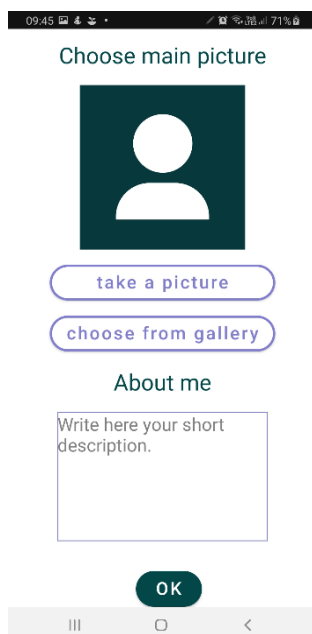
Rysunek 35 Zdjęcie interfejsu użytkownika Firebase Realtime Database po dodaniu użytkownika poprzez naciśnięcie przycisku „OK” w aktywności ProfileInfoActivity.



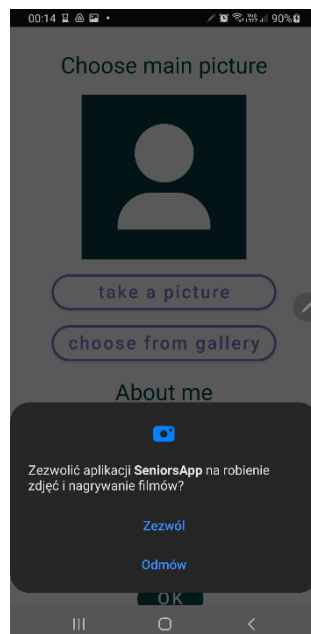
Rysunek 36 Aktywność *NavigationActivity* po udanym dodaniu użytkownika to *Firebase RealtimeDatabase*.

### 3.5 Dodawanie zdjęcia

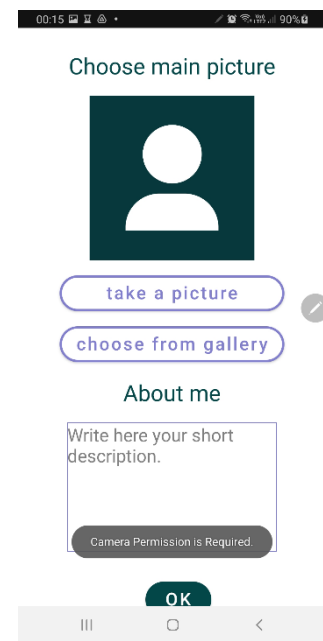
Dodawanie zdjęcia przez użytkownika może odbyć się na dwa sposoby. Pierwszy polega na zrobieniu nowego zdjęcia i odbywa się poprzez naciśnięcie przycisku „take a picture”, które powoduje wywołanie metody `askCameraPermission()`. Działanie tej metody polega na zapytaniu użytkownika o pozwolenie na dostęp do aparatu. Następnie w metodzie `onRequestPermissionsResult()` sprawdzane jest czy dostęp został udzielony. Jeżeli nie, wyświetlany jest komunikat informujący o tym, że udzielenie dostępu jest niezbędne do zrobienia zdjęcia. Jeżeli dostęp został udzielony aplikacja tworzy plik, w którym zostanie zapisane zrobione zdjęcie przy pomocy metod `dispatchTakePictureIntent()` i `createImageFile()`. Drugi sposób wymaga naciśnięcia przycisku „choose from gallery”, po czym otwierana jest aplikacja Galeria, w której użytkownik może wybrać jedno ze zdjęć mieszczących się w którymkolwiek z folderów galerii. Dodatkowo dzięki przesłonięciu metody `onActivityResult()` aplikacja rozpoznaje jaki `requestCode` został zwrócony i w ten sposób dostosowuje w jaki sposób należy nazwać plik, wyświetlić go i zapisać do Firebase Storage.



Rysunek 37 Aktywność *ProfileInfoActivity*.



Rysunek 38 Prośba o dostęp do aplikacji aparatu.



Rysunek 39 Komunikat wyświetlany w przypadku odmowy dostępu.

```
cameraBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        askCameraPermission();
    }
});
```

Listing 18 Wywołanie metody `askCameraPermission()` w momencie naciśnięcia przycisku „take a picture”.

```

private void askCameraPermission() {
    if (ContextCompat.checkSelfPermission(getApplicationContext(),
Manifest.permission.CAMERA) != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(this, new String[]
{Manifest.permission.CAMERA}, CAMERA_PERMISSION_CODE);
    } else {
        dispatchTakePictureIntent();
    }
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull
String[] permissions, @NonNull int[] grantResults) {
    if (requestCode == CAMERA_PERMISSION_CODE) {
        if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
            dispatchTakePictureIntent();
        } else {
            Toast.makeText(getApplicationContext(), "Camera
Permission is Required.", Toast.LENGTH_SHORT).show();
        }
    }
}
}

```

*Listing 19 Definicja metody askCameraPermission() i przesłonięcie metody onRequestPermissionsResult().*

```

private File createImageFile() throws IOException {
    String timeStamp = new
SimpleDateFormat("yyyyMMdd_HH:mm:ss").format(new Date());
    String imageFileName = "image_" + timeStamp + "_";
    File storageDir =
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_P
ICTURES);
    File image = File.createTempFile(
imageFileName,
".jpg",
storageDir);

    currentPhotoPath = image.getAbsolutePath();
    return image;
}

```

*Listing 20 Definicja metody createImageFile().*

```

private void dispatchTakePictureIntent() {
    Intent takePictureIntent = new
Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    if (takePictureIntent.resolveActivity(getPackageManager()) !=
null) {
        File photoFile = null;
        try {
            photoFile = createImageFile();
        } catch (IOException ex) {
            Toast.makeText(getApplicationContext(), "Something went
wrong!", Toast.LENGTH_SHORT).show();
        }
        if (photoFile != null) {
            Uri photoURI = FileProvider.getUriForFile(this,
"com.roksanagulewska.android.fileprovider",
photoFile);
            takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT,
photoURI);
            startActivityForResult(takePictureIntent,
CAMERA_REQUEST_CODE);
        }
    }
}

```

Listing 22 Definicja metody `dispatchTakePictureIntent()`.

```

@Override
protected void onActivityResult(int requestCode, int resultCode,
@Nullable Intent data) {
    if (requestCode == CAMERA_REQUEST_CODE) {
        if (resultCode == Activity.RESULT_OK) {
            File file = new File(currentPhotoPath);
            mainPicture.setImageURI(Uri.fromFile(file));
            Intent mediaScanIntent = new
Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE);
            contentUri = Uri.fromFile(file);
            mediaScanIntent.setData(contentUri);
            this.sendBroadcast(mediaScanIntent);
            fileName = file.getName();
            uploadImageToFirebase(file.getName(), contentUri);
        }
    } else if (requestCode == GALLERY_REQUEST_CODE) {
        if (resultCode == Activity.RESULT_OK) {
            contentUri = data.getData();
            String timeStamp = new
SimpleDateFormat("yyyyMMdd_HH:mm:ss").format(new Date());
            String imageFileName = "image_" + timeStamp + "." +
getFileExt(contentUri);
            mainPicture.setImageURI(contentUri);
            fileName = imageFileName;
            uploadImageToFirebase(imageFileName, contentUri);
        }
    }
}

```

Listing 21 Przesłonięcie metody `onActivityResult()`.



```

public void uploadImageToFirebase(String fileName, Uri contentUri) {
    StorageReference imageStorageReference =
    dbHelper.getStorageReference().child("Pictures").child(fileName);

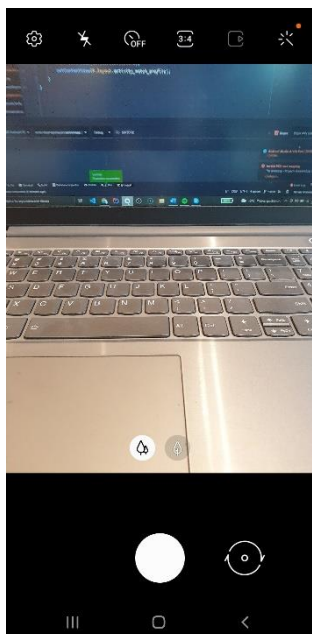
    imageStorageReference.putFile(contentUri).addOnSuccessListener(new
    OnSuccessListener<UploadTask.TaskSnapshot>() {
        @Override
        public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {

            imageStorageReference.getDownloadUrl().addOnSuccessListener(new
            OnSuccessListener<Uri>() {
                @Override
                public void onSuccess(Uri uri) {
                    imageUri = uri.toString();
                    imageUrl =
                    imageStorageReference.getDownloadUrl().toString();
                }
            });
            Toast.makeText(getApplicationContext(), "Image uploaded
            successfully!", Toast.LENGTH_SHORT);
        }
    }).addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            Toast.makeText(getApplicationContext(), "Upload failed!",
            Toast.LENGTH_SHORT);
        }
    });
}

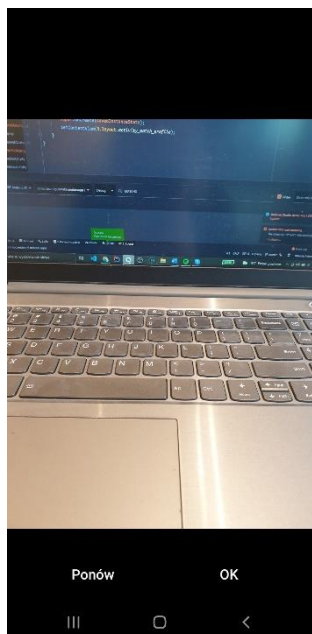
```

*Listing 23 Definicja metody uploadImageToFirebase() odpowiedzialnej za dodanie zdjęcia do bazy danych Firebase Storage.*

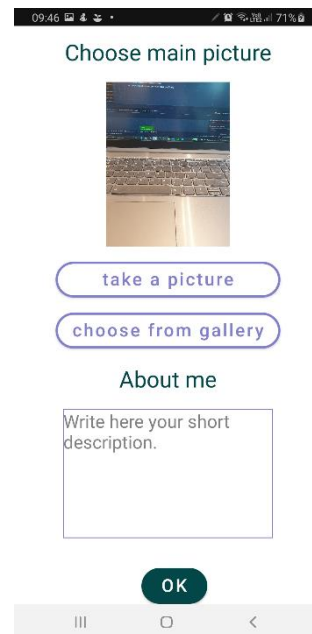
W metodzie uploadImageToFirebase() odbywa się dodawanie pliku o nazwie i Uri podanych w argumentach do Firebase Storage. Tworzony jest obiekt klasy StorageReference, który zawiera ścieżkę w Firebase Storage, pod której adresem ma znajdować się przesłany plik. Następnie przy użyciu listenera pod tą ścieżką ma zostać zapisany plik o Uri podanym w argumencie metody i w zależności od powodzenia operacji ma zostać wyświetlony odpowiedni komunikat.



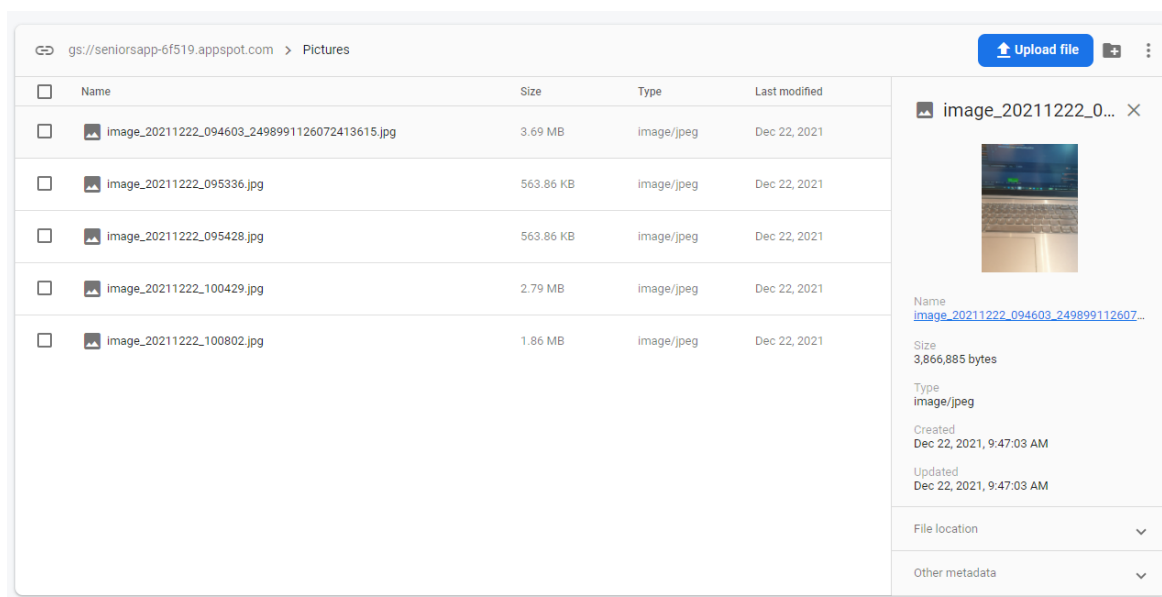
Rysunek 40 Włączenie aplikacji aparatu, po zaakceptowaniu prośby o dostęp do aparatu.



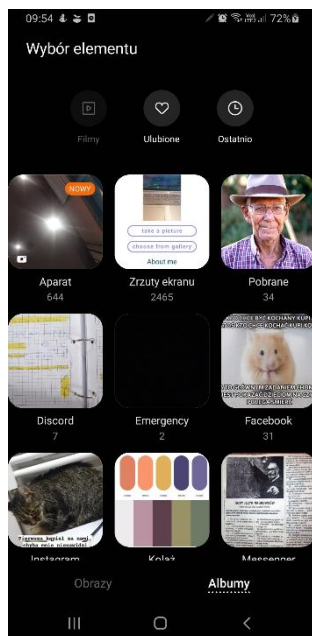
Rysunek 41 Wyświetlenie zrobionego zdjęcia i oczekiwanie na akceptację albo ponowienie próby zrobienia zdjęcia.



Rysunek 42 Wyświetlenie zrobionego zdjęcia w ImageView w ProfileInfoActivity.



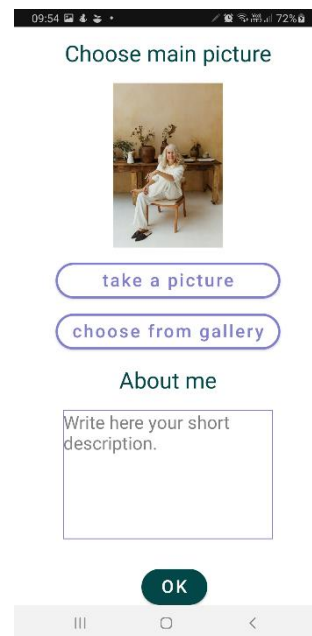
Rysunek 43 Zdjęcie interfejsu użytkownika Firebase, na którym widać, że zdjęcie zrobione przez aplikację za pomocą aparatu zostało dodane do Firebase Storage.



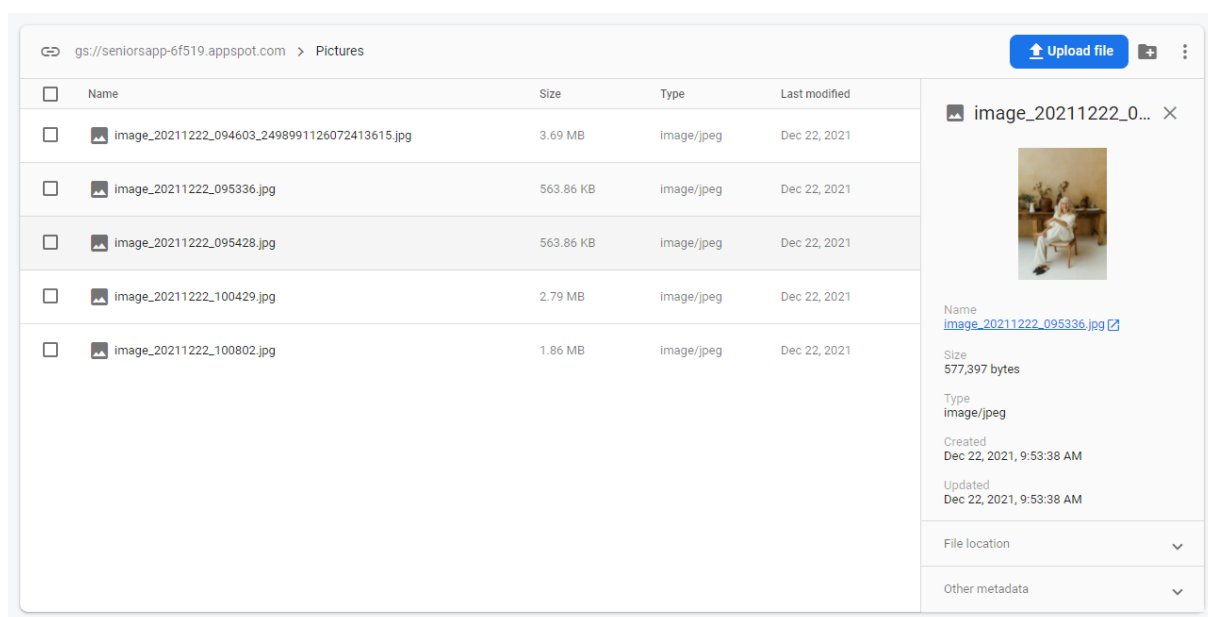
Rysunek 44 Uruchomienie aplikacji galerii w wyniku naciśnięcia przycisku „choose from gallery”.



Rysunek 45 Wejście do folderu pobrane, z którego wybierane jest zdjęcie.



Rysunek 46 Załadowanie wybranego zdjęcia do ImageView.



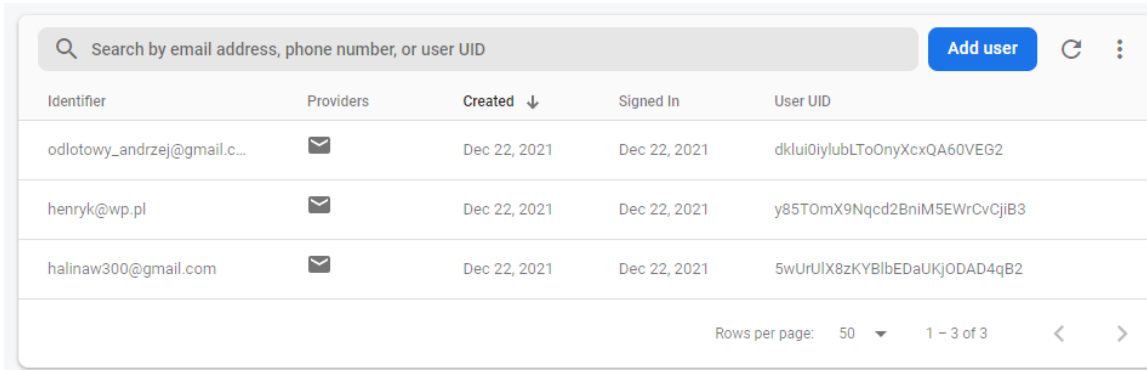
Rysunek 47 Zdjęcie interfejsu użytkownika Firebase, na którym widać, że zdjęcie wybrane z galerii zostało dodane do Firebase Storage.

### 3.6 Baza danych

Kluczowymi elementami Firebase, które zostały wykorzystane w aplikacji są Firebase Authentication, Firebase Realtime Database i Firebase Storage.

- Firebase Authentication – kompleksowe rozwiązanie w zakresie uwierzytelniania użytkownika, w aplikacji umożliwia rejestrację i bezpieczne logowanie za pomocą loginu i hasła.

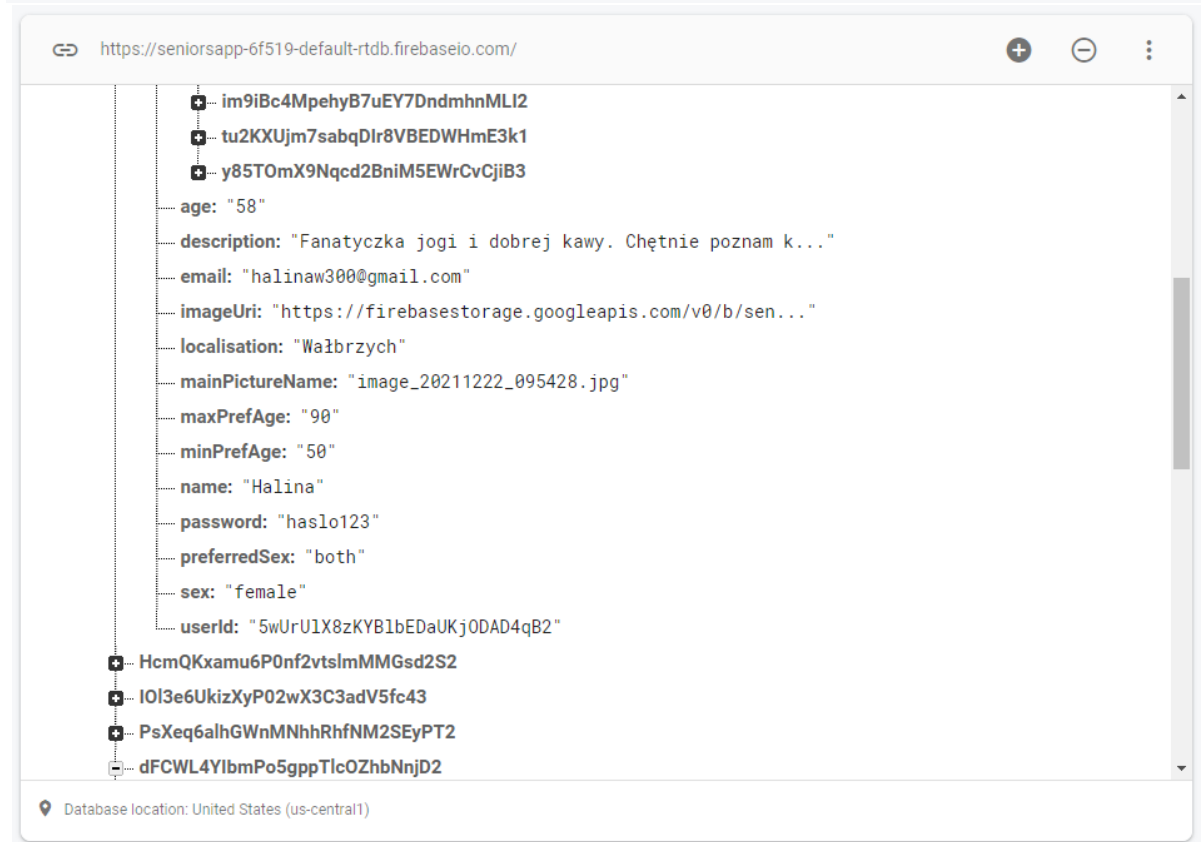
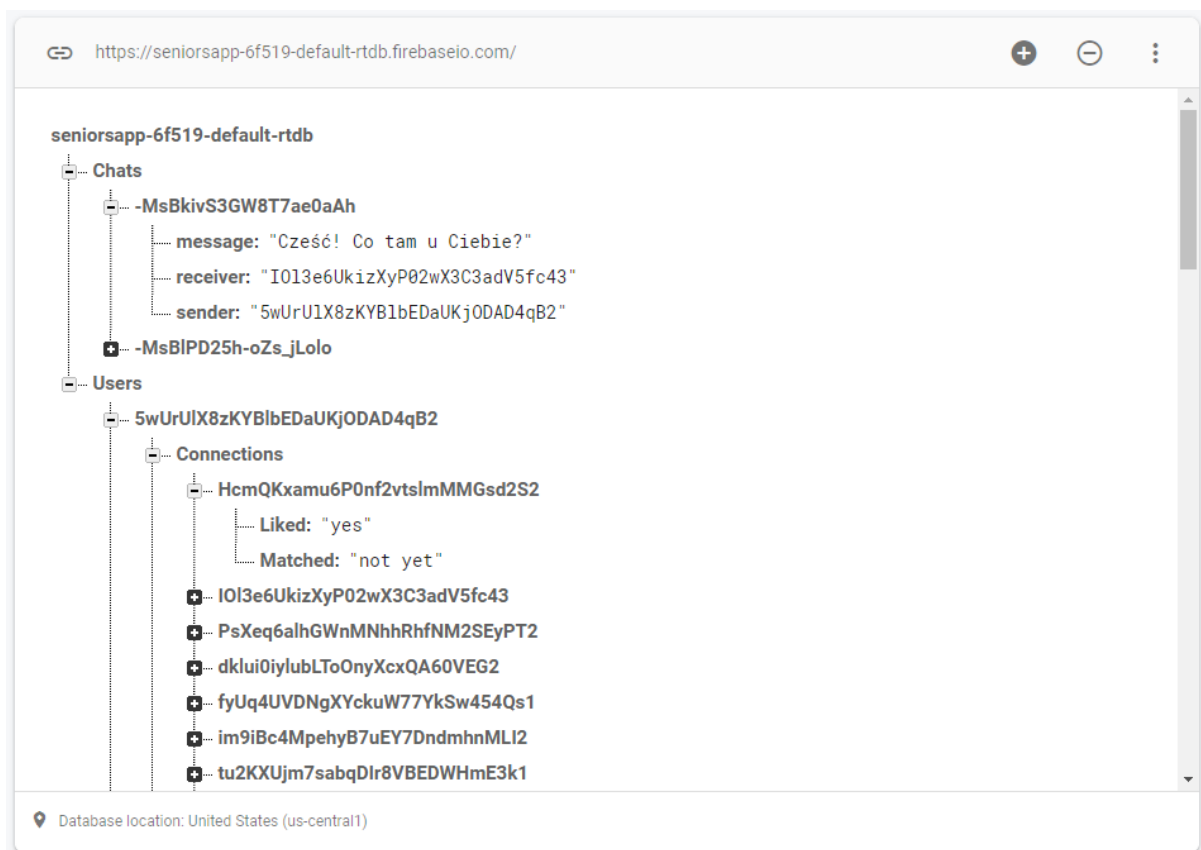
Rysunek przedstawia fragment interfejsu użytkownika Firebase Authentication, jest to lista zarejestrowanych przez Firebase użytkowników aplikacji. W poszczególnych kolumnach znajdują się informacje takie jak identyfikator użytkownika - którym w tym przypadku jest adres e-mail, dostawca, data utworzenia konta użytkownika, data ostatniego logowania i UID – czyli id użytkownika. Zarówno id jak i email są unikalne.



Identifier	Providers	Created ↓	Signed In	User UID
odlotowy_andrzej@gmail.c...	✉	Dec 22, 2021	Dec 22, 2021	dklui0iyIubLToOnyXcxQA60VEG2
henryk@wp.pl	✉	Dec 22, 2021	Dec 22, 2021	y85T0mX9Nqcd2BniM5EWrCvCjiB3
halinaw300@gmail.com	✉	Dec 22, 2021	Dec 22, 2021	5wUrUIX8zKYBlbEDaUKjODAD4qB2

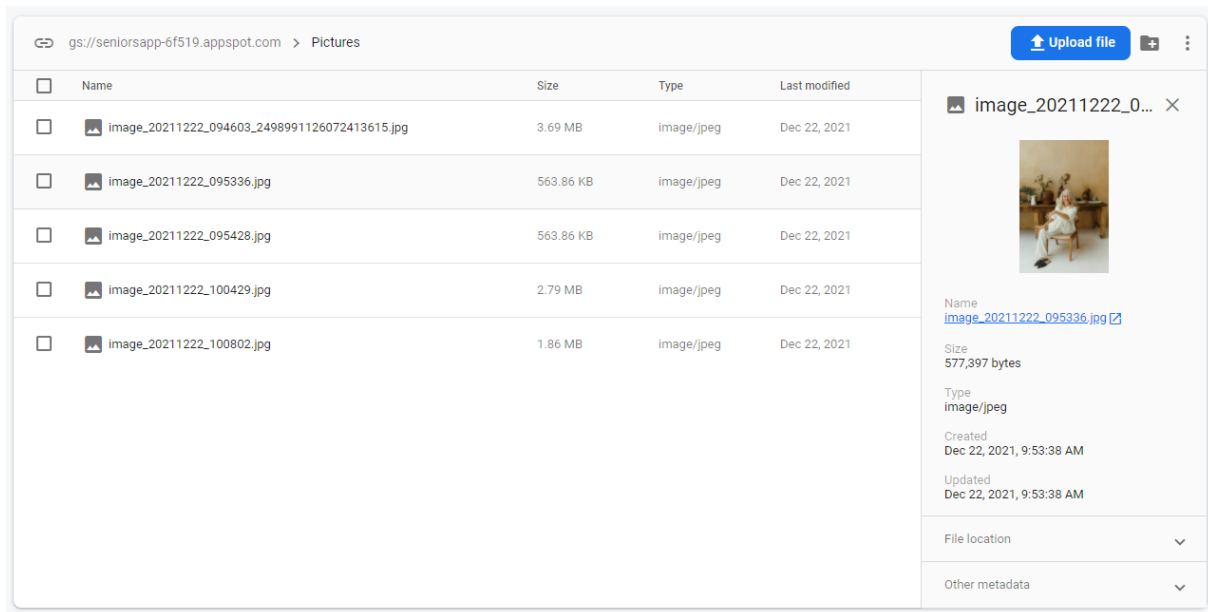
Rysunek 48 Firebase Authentication

- Firebase Realtime Database – baza danych czasu rzeczywistego – zapewnia aktualizację danych w momencie, w którym w bazie pojawi się zmiana, dzięki czemu użytkownik naszej aplikacji zawsze ma dostęp do aktualnych danych. Jest to baza NoSQL, przechowuje ona dane w formacie JSON. Posiada jeden element nadrzędny, tzw. węzeł główny, którego węzłami podrzędnymi są dane przechowywane w formacie klucz - wartość. W bazie połączonej z aplikacją przechowywane są informacje o użytkowniku, informacje o jego preferencjach i nazwa oraz Uri jego zdjęcia. Ponadto w momencie wygenerowania listy potencjalnych znajomych dla zalogowanego użytkownika, w elemencie podrzędnym węzła Users, o nazwie tożsamej z UID zalogowanego użytkownika pojawia się element podrzędny Connections, który z kolei posiada elementy o nazwach będących UID użytkowników, którzy zostali rozpoznani przez algorytm jako pasujący do preferencji użytkownika. Inny węzeł tego samego rzędu co Users to Chats. Wewnątrz znajdują się wiadomości identyfikowane po unikalnym wygenerowanym przez Firebase id. Ich elementami podrzędnymi są nadawca, adresat oraz treść wiadomości.



Rysunek 49 Firebase Realtime Database

- Firebase Storage – zewnętrzna powierzchnia dyskowa - odpowiada za przechowywanie plików, w przypadku tej aplikacji są to zdjęcia dodane przez użytkowników.



Rysunek 50 Fragment interfejsu użytkownika Firebase Storage

### 3.7 Menu

Aktywność uruchamiana po zalogowaniu lub pełnej rejestracji (wraz z dodaniem informacji o użytkowniku, zdjęcia, opisu) to `NavigationActivity`. Aktywność ta zawiera fragmenty, między którymi użytkownik może się przemieszczać za pomocą wybierając pozycje z menu umieszczonego wewnątrz wysuwanej bocznej szuflady (`DrawerLayout`). Fragmenty to modularne części aktywności. Każdy fragment musi być zagnieżdżony w jakiejś aktywności i posiada swój własny cykl życia powiązany z tą aktywnością. Fragmentem wyświetlanym w momencie przejścia do aktywności `NavigationActivity` jest `FindNewFriendsFragment`. Dzieje się tak dzięki ustawieniu go jako wybranego elementu z menu przy pomocy metody `setCheckedItem()`.

```

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:tools="http://schemas.android.com/tools"
      tools:showIn="navigation_view">

    <group android:checkableBehavior="single">

        <item
            android:id="@+id/nav_my_profile"
            android:icon="@drawable/ic_myprofile"
            android:title="My profile" />

        <item
            android:id="@+id/nav_messages"
            android:icon="@drawable/ic_messages"
            android:title="Messages" />

        <item
            android:id="@+id/nav_find_friends"
            android:icon="@drawable/ic_friends"
            android:title="Find new friends" />

        <item
            android:id="@+id/nav_settings"
            android:icon="@drawable/ic_settings"
            android:title="Settings" />
    </group>

    <item
        android:id="@+id/nav_logout"
        android:icon="@drawable/ic_logout"
        android:title="Log out" />

</menu>

```

Listing 24 Zawartość pliku `nav_menu.xml` definiującego formę menu, ikony i nazwy elementów.

```

if (savedInstanceState == null) {

    getSupportFragmentManager().beginTransaction().replace(R.id.fragment_
container, new FindNewFriendsFragment()).commit();
    navigationView.setCheckedItem(R.id.nav_find_friends);
}

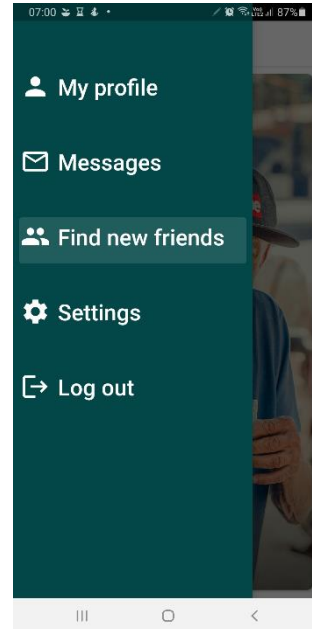
```

Listing 25 Otwarcie fragmentu `FindNewFriendsFragment` podczas startu aktywności `NavigationActivity`.

Otwarcie szuflady menu odbywa się poprzez naciśnięcie ikony hamburgera znajdującej się w lewym górnym rogu aktywności `NavigationActivity`. Dzięki przesłonięciu metody `onBackPressed()` naciśnięcie przycisku „Wróć” zamiast zamknąć aplikację, zamyka szufladę.



Rysunek 51 Aktywność `NavigationActivity` z otwartym fragmentem `FindNewFriendsFragment`.



Rysunek 52 Otwarta szuflada z menu i wybranym elementem `FindNewFriendsFragment`.

```
@Override
public void onBackPressed() {
    if (drawer.isDrawerOpen(GravityCompat.START)) {
        drawer.closeDrawer(GravityCompat.START);
    } else {
        super.onBackPressed();
    }
}
```

Listing 26 Przesłonięcie metody `onBackPressed()`.

Zdecydowałam się na użycie fragmentów, aby zapewnić użytkownikowi prostą nawigację między podstawowymi funkcjonalnościami aplikacji przy pomocy wygodnego, rozwijanego menu, które jest dostępne po zalogowaniu. Nazwy elementów menu dobrze określają funkcje pełnione przez poszczególne fragmenty, czyniąc poruszanie się po aplikacji bardzo intuicyjnym.



### 3.8 Wyświetlanie profilu zalogowanego użytkownika

Wybierając pozycję „My profile” w NavigationActivity aplikacja wyświetla fragment MyProfileFragment. W jego układzie znajduje się komponent ImageView, w którym wyświetlane jest zdjęcie zalogowanego użytkownika i komponenty TextView, w których wyświetlane jest jego imię, lokalizacja, wiek i opis. Wszystkie te informacje są pobierane z bazy danych za pomocą ValueEventListener i wyświetlane przy użyciu metody displayCurrentUsersInfo() w momencie wybrania fragmentu MyProfileFragment.



*Rysunek 53 Fragment MyProfileActivity wypełniony danymi zalogowanego użytkownika.*

```

ValueEventListener currentUserDataValueEventListener = new
ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        if(snapshot.hasChildren()) {
            currentUserName =
snapshot.child("name").getValue().toString();
            currentUserLocalisation =
snapshot.child("localisation").getValue().toString();
            currentUserAge =
snapshot.child("age").getValue().toString();
            currentUserDescription =
snapshot.child("description").getValue().toString();
            currentUserImageName =
snapshot.child("mainPictureName").getValue().toString();
            currentUserImageUri =
snapshot.child("imageUri").getValue().toString();
            displayCurrentUsersInfo();
        }
    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {
    }
};

dbHelper.getCurrentUserReference().addValueEventListener(currentUser
DataValueEventListener);

```

Listing 27 Pobranie z bazy danych informacji o użytkowniku, aby wyświetlić je w profilu.

```

private void displayCurrentUsersInfo() {
    if (currentUserImageName.length() > 27) {
        Picasso.get()
            .load(currentUserImageUri)
            .fit()
            .centerCrop()
            .rotate(90)
            .into(imageView);
    } else {
        Picasso.get()
            .load(currentUserImageUri)
            .fit()
            .centerCrop()
            .into(imageView);
    }
    nameTxt.setText(currentUserName);
    localisationTxt.setText(currentUserLocalisation);
    ageTxt.setText(currentUserAge);
    descriptionTxt.setText(currentUserDescription);
}

```

Listing 28 Definicja metody displayCurrentUsersInfo() w pliku MyProfileFragment.java

### 3.9 Wyświetlanie potencjalnych znajomych

W momencie uruchomienia aktywności `NavigationActivity`, a tym samym wyświetlenia fragmentu `FindNewFriendsFragment` sprawdzane jest czy w bazie danych znajdują się użytkownicy wpasowujący się w preferencje zalogowanego użytkownika. Dzieje się to poprzez sprawdzenie preferencji użytkownika przy pomocy metody `checkUsersPreferences()`, a następnie metody `listPotentialMatches()`. Wydobywanie danych z bazy odbywa się poprzez ustawienie `ValueEventListener` na wybranej referencji do bazy danych i przesłonięcie metod `onDataChange()` i `onCancelled()`. Metoda `onDataChange()` odczytuje statyczną migawkę zawartości bazy danych na podanej ścieżce (referencji). Jej kod wykonuje się raz podczas dodania listenera i za każdym razem gdy dane w podanej ścieżce ulegną zmianie (w tym elementy podrzędne). Metoda `onCancelled()` jest wywoływana w przypadku anulowania odczytu danych z bazy.

```
public void checkUsersPreferences() {
    ValueEventListener checkPreferencesValueEventListener = new
    ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot)
        {
            if(dataSnapshot.hasChildren()) {
                preferredSex =
                dataSnapshot.child("preferredSex").getValue().toString();
                minPrefAge =
                dataSnapshot.child("minPrefAge").getValue().toString();
                maxPrefAge =
                dataSnapshot.child("maxPrefAge").getValue().toString();
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                Log.w("READ_PREFERENCES_ERROR", "Failed to read
                preferences values.", error.toException());
            }
        };

        dbHelper.getCurrentUserReference().addValueEventListener(checkPrefere
        ncesValueEventListener);
    }
}
```

*Listing 29 Definicja metody `checkUsersPreferences()`.*

```

public void listPotentialMatches() {
    ValueEventListener listPotentialMatchesValueEventListener = new
    ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot)
        {
            for (DataSnapshot snapshot : dataSnapshot.getChildren())
            {
                User user = snapshot.getValue(User.class);
                if (preferredSex.equals(user.getSex()) ||
                preferredSex.equals("both")) {
                    if (Integer.parseInt(user.getAge()) <=
                    Integer.parseInt(maxPrefAge) && Integer.parseInt(user.getAge()) >=
                    Integer.parseInt(minPrefAge)) {
                        if (!dbHelper.getCurrentUserId().equals(user.getUserId())) {
                            potentialMatchesList.add(user);
                        }
                    }
                }
            }
            if (isFirstTime) {
                isFirstTime = false;
                for (User element : potentialMatchesList) {
                    if
                    (dataSnapshot.child(dbHelper.getCurrentUserId()).child("Connections")
                    .hasChild(element.getUserId())) {
                        Log.d("PREFF", "Takie connections już
                        istnieje! " + potentialMatchesList.size());
                    }
                    if
                    (dataSnapshot.child(dbHelper.getCurrentUserId()).child("Connections")
                    .child(element.getUserId()).child("Liked").getValue().equals("not
                    yet")) {
                        potentialMatchesList2.add(element);
                    }
                } else {
                    dbHelper.getCurrentUserReference().child("Connections").child(element
                    .getUserId()).child("Liked").setValue("not yet");
                    dbHelper.getCurrentUserReference().child("Connections").child(element
                    .getUserId()).child("Matched").setValue("not yet");
                    potentialMatchesList2.add(element);
                }
            }
            addItemModelList();
        }
        adapter.notifyDataSetChanged();
    }
    @Override
    public void onCancelled(@NonNull DatabaseError error) {
        Log.w("READ_POTENTIAL_MATCHES_ERROR", "Failed to read
        values.", error.toException());
    }
};

dbHelper.getDatabaseReference().child("Users").addValueEventListener(
listPotentialMatchesValueEventListener);
}

```

Listing 30 Definicja metody listPotentialMatches()

W metodzie `listPotentialMatches` każdy element podrzędny węzła `Users` jest wykorzystywany do stworzenia obiektu typu `User`. Składowe obiekty mają nadawane wartości poszczególnych elementów podrzędnych każdego użytkownika. Następnie sprawdzane jest czy składowe tych obiektów spełniają kryteria określone przez preferencje zalogowanego użytkownika i na tej podstawie, dodawane lub nie do listy `potentialMatchesList`. Następnie w węźle zalogowanego użytkownika tworzony jest węzeł podrzędny „Connections”. Tam dodawane są elementy podrzędne będące UID użytkowników z listy, każdy z nich ma 2 elementy „Liked” i „Matched” o wartościach „not yet”, które sygnalizują, że zalogowany użytkownik jeszcze nie zdecydował czy chce nawiązać kontakt z tym użytkownikiem, a zatem ten element listy musi być wyświetlany. Zmienna o nazwie `isFirstTime` jest flagą pomocniczą, której użycie zapobiega nadpisywaniu już polubionych lub odrzuconych użytkowników. Metoda `addItemModelList()` ma za zadanie utworzenie obiektu `ItemModel` dla każdego użytkownika z listy `potentialMatchesList2`.

```
private void addItemModelList () {
    items.clear();
    Log.d("PREFL", "ListaMatches: " + potentialMatchesList.size());
    Log.d("PREFL", "ListaMatches2: " + potentialMatchesList2.size());
    for (User potentialMatch : potentialMatchesList2) {
        items.add(new ItemModel(potentialMatch.getEmail(),
            potentialMatch.getUserId(), potentialMatch.getName(),
            potentialMatch.getAge(), potentialMatch.getLocalisation(),
            potentialMatch.getMainPictureName(), potentialMatch.getImageUri(),
            potentialMatch.getDescription()));
    }
    Log.d("PREFL", "ListaItem1: " + items.size());
}
```

*Listing 31 Definicja metody `addItemModelList()`*

`ItemModel` to klasa używana przez `CardStackAdapter` do wyświetlania tylko niektórych informacji o użytkownikach przy użyciu układu `item_card.xml`. Umieszczanie danych na kartach odbywa się w metodzie `setData()` w pliku `CardStackAdapter`.

```

void setData(ItemModel data) {
    if (data.getImageName().length() > 27) {
        Picasso.get()
            .load(data.getImageUri())
            .fit()
            .centerCrop()
            .rotate(90)
            .into(image);
    } else {
        Picasso.get()
            .load(data.getImageUri())
            .fit()
            .centerCrop()
            .into(image);
    }

    name.setText(data.getName());
    age.setText(String.valueOf(data.getAge()));
    localisation.setText(data.getLocalisation());
    description.setText(data.getDescription());
}

```

Listing 32 Definicja metody `setData()` w pliku `CardStackAdapter.java`



Rysunek 54 Wyświetlenie karty z informacjami o użytkowniku i jego zdjęciem w `FindNewFriendsFragment` w `NavigationActivity`.

### 3.10 Mechanizm polubień i odrzuceń

Karty z informacjami o użytkownikach można przesuwac w prawo – co oznacza polubienie/akceptację użytkownika, lub w lewo – co oznacza odrzucenie. W momencie polubienia lub odrzucenia aktualizuje się wartość elementu podrzędnego „Liked” węzła „Connections” użytkownika, którego kartę przesunęliśmy. Przed przesunięciem element ten miał wartość „not yet”, po przesunięciu w prawo na przesuwanej karcie pojawi się ikona serca i wartość zostanie zmieniona na „yes”, a w przypadku przesunięcia w lewo pojawi się „X” i wartość zostanie zmieniona na „no”. W momencie przesunięcia w jakąkolwiek stronę dane użytkownika w postaci obiektu `ItemModel` są usuwane z listy `items`, a na jego miejsce wskakuje kolejny element z listy, którego składowe również są wyświetlane w taki sam sposób.

```
if (direction == Direction.Right){
    Toast.makeText(getContext(), "Like", Toast.LENGTH_SHORT).show();
    String cardUserId = items.get(0).getUserId();
    Log.d("TEEST", "user ID: " + cardUserId);
    Log.d("TEEST", "user name: " + items.get(0).getName());
    Map<String, Object> map = new HashMap<String, Object>();
    map.put("Liked", "yes");

    dbHelper.getCurrentUserReference().child("Connections").child(cardUser
    rId).updateChildren(map);
} else if (direction == Direction.Left){
    String cardUser = items.get(0).getUserId();
    Toast.makeText(getContext(), "Dislike",
    Toast.LENGTH_SHORT).show();
    Map<String, Object> map = new HashMap<String, Object>();
    map.put("Liked", "no");

    dbHelper.getCurrentUserReference().child("Connections").child(cardUse
    r).updateChildren(map);
}

items.remove(0);
```

*Listing 33 Mechanizm przesuwania kart i aktualizacji bazy danych.*

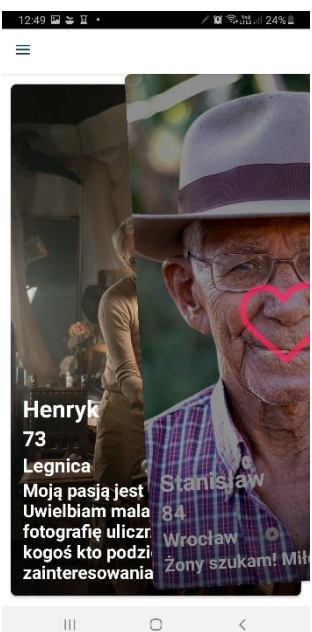
Kiedy użytkownicy z listy się wyczerpią i zostanie przesunięta ostatnia karta, odsłonięte zostanie tło, na którym widnieje komunikat informujący o wyczerpaniu potencjalnych par. Kiedy do bazy zostanie dodany ktoś nowy, pasujący do wymagań użytkownika, wtedy do listy zostanie dodany nowy obiekt `ItemModel` i karta nowego użytkownika zostanie wyświetlona.



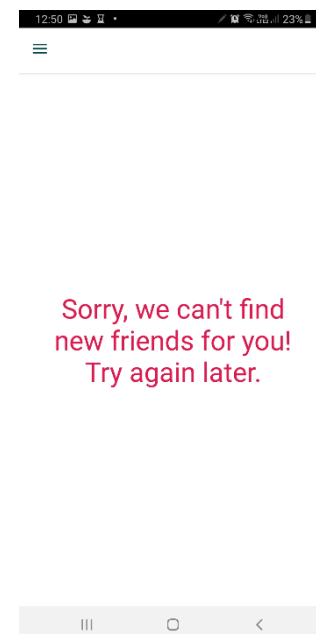
Rysunek 55 Przesunięcie w lewo, czyli odrzucenie użytkownika z karty.



Rysunek 56 Wyświetlenie karty z danymi następnego użytkownika z listy.



Rysunek 57 Przesunięcie w prawo, czyli polubienie/akceptacja użytkownika z karty.



Rysunek 58 Komunikat odsłonięty w przypadku wyczerpania użytkowników z listy.



### 3.11 Wyświetlanie listy dopasowanych użytkowników

Wyświetlanie listy osób dopasowanych do zalogowanego użytkownika odbywa się wewnątrz fragmentu MessagesFragment w aktywności NavigationActivity. W metodzie onCreate() wywoływana jest metoda listLikedUsers() jej działanie polega na pobraniu UID z węzła podrzędnego Connections zalogowanego użytkownika i dodaniu do listy potentialMatchesList tych UID, które w elemencie Liked miały wartość „yes”. W ten sposób zostaje utworzona lista osób, które zostały polubione przez zalogowanego użytkownika. Wewnątrz tej metody wywoływana jest metoda listMatches(), która w argumencie przyjmuje wypełnioną listę potentialMatchesList. Takie rozwiązanie zostało wybrane ze względu na to, że metoda onDataChange() jest metodą asynchroniczną, w wyniku czego wywołanie metody przyjmującej listę wypełnianą przy użyciu metody onDataChange() skutkowałoby przyjmowaniem pustej listy, ponieważ onDataChange wykonałoby się później niż zostałaby wywołana metoda listMatches().

```
public void listLikedUsers() {
    potentialMatchesList.clear();
    ValueEventListener checkLikesValueEventListener = new
    ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot)
        {
            for (DataSnapshot snapshot : dataSnapshot.getChildren())
            {
                String potentialFriend = snapshot.getKey();

                if(snapshot.child("Liked").getValue().toString().equals("yes")) {
                    Log.d("MATCHX", "L: yes");
                    potentialMatchesList.add(potentialFriend);
                } else if
                (snapshot.child("Liked").getValue().toString().equals("no")) {
                    Log.d("MATCHX", "L: no");
                }
            }
            listMatches(potentialMatchesList);
        }
        @Override
        public void onCancelled(@NonNull DatabaseError error) {
            Log.w("READ_LIKES_ERROR", "Failed to read preferences
            values.", error.toException());
        }
    };

    dbHelper.getCurrentUserReference().child("Connections").addValueEvent
    Listener(checkLikesValueEventListener);
}
```

Listing 34 Definicja metody listLikedUsers() w pliku MessagesFragment.java.

```

private void listMatches(List<String> list) {
    ValueEventListener checkIfMatchValueEventListener = new
    ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot)
        {
            for (DataSnapshot snapshot : dataSnapshot.getChildren())
            {
                if(potentialMatchesList.contains(snapshot.getKey()))
                {
                    if
                    (snapshot.child("Connections").child(dbHelper.getCurrentUserId()).exists()) {
                        if
                        (snapshot.child("Connections").child(dbHelper.getCurrentUserId()).child("Liked").getValue().toString().equals("yes")) {
                            matchesList.add(snapshot.getKey());
                            Map<String, Object> map = new
                            HashMap<String, Object>();
                            map.put("Matched", "yes");

                            dbHelper.getCurrentUserReference().child("Connections").child(snapshot.getKey()).updateChildren(map);
                        } else if
                        (snapshot.child("Connections").child(dbHelper.getCurrentUserId()).child("Liked").getValue().toString().equals("no")) {
                            Map<String, Object> map = new
                            HashMap<String, Object>();
                            map.put("Matched", "no");

                            dbHelper.getCurrentUserReference().child("Connections").child(snapshot.getKey()).updateChildren(map);
                        }
                    }
                }
            }
            for (String match : matchesList) {
                if (dataSnapshot.hasChild(match)) {
                    User matchedUser =
                    dataSnapshot.child(match).getValue(User.class);
                    friendsList.add(matchedUser);
                }
            }
            addItemMatchModelList();
            adapter.notifyDataSetChanged();
        }
        @Override
        public void onCancelled(@NonNull DatabaseError error) {
        }
    };

    dbHelper.getDatabaseReference().child("Users").addValueEventListener(
    checkIfMatchValueEventListener);
}

```

Listing 35 Definicja metody listMatches() w pliku MessagesFragment.java.

Metoda `listMatches()` ma za zadanie sprawdzenie czy osoby, których UID znajduje się na liście `potentialMatchesList` zalogowanego użytkownika również go polubiły. Jeżeli osoba z listy polubiła zalogowanego użytkownika, jest ona dodawana do listy `matchesList` i wartość elementu `Matched` w jej elemencie podrzędnym, wewnątrz węzła `Connections` zalogowanego użytkownika zostaje zmieniona na „yes”, jeżeli jest przeciwnie to wartość jest zmieniana na „no”. Następnie dla każdego elementu listy `matchesList` tworzy obiekt klasy `User`, a jego składowym nadaje wartości odpowiadające informacjom o danym użytkowniku zgromadzonym w bazie. Obiekty te są dodawane do listy `friendsList`, a następnie wywoływana jest metoda `addItemMatchModelList()`, która ma za zadanie utworzyć element `ItemMatchModel` dla każdego elementu listy `friendsList` i dodać go do listy `friendsListToDisplay`, analogicznie jak w metodzie `addItemModelList()` w pliku `FindNewFriendsFragment.java`.

```
private void addItemMatchModelList () {
    friendsListToDisplay.clear();
    for (User friend : friendsList) {
        friendsListToDisplay.add(new
ItemMatchModel (friend.getUserId(), friend.getName(),
friend.getMainPictureName(), friend.getImageUri()));
    }
    adapter = new MatchesListAdapter (friendsListToDisplay,
getContext());
    matchesRecyclerView.setAdapter (adapter);
    adapter.notifyDataSetChanged();
}
```

*Listing 36 Definicja metody `addItemMatchModelList()` w pliku `MessagesFragment.java`.*

`ItemMatchModel` jest klasą używaną przez `MatchesListAdapter` do wyświetlania informacji o użytkownikach przy użyciu układu `item_match.xml`. Umieszczanie danych na kartach odbywa się w metodzie `setData()` w pliku `MatchesListAdapter`, analogicznie do metody o tej samej nazwie `CardStackAdapter`.

Układ fragmentu `MessagesFragment` został stworzony przy pomocy komponentu `RecyclerView`, dzięki czemu wyświetlana jest lista elementów `item_match.xml`.

```

void setData(ItemMatchModel data) {
    Log.d("MATCHX", "setData");
    if (data.getImageName().length() > 27) {
        Picasso.get()
            .load(data.getImageUri())
            .fit()
            .centerCrop()
            .rotate(90)
            .into(image);
    } else {
        Picasso.get()
            .load(data.getImageUri())
            .fit()
            .centerCrop()
            .into(image);
    }
    name.setText(data.getName());
    clickedUserId = data.getUserId();
    clickedUserName = data.getName();
}

```

Listing 37 Definicja metody setData() w pliku MatchesListAdapter.java

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="160dp"
    android:background="@color/white">

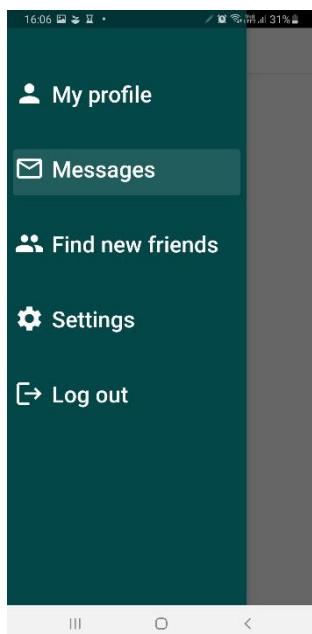
    <ImageView
        android:id="@+id/item_match_image"
        android:layout_width="100dp"
        android:layout_height="120dp"
        android:layout_marginStart="25dp"
        android:scaleType="centerCrop"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/item_match_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="20dp"
        android:textColor="@color/color2"
        android:textSize="36dp"
        android:fontFamily="sans-serif-medium"
        android:textStyle="bold"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toEndOf="@+id/item_match_image"
        app:layout_constraintTop_toTopOf="parent" />

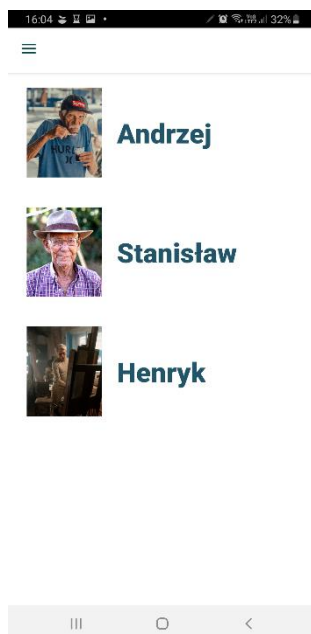
</androidx.constraintlayout.widget.ConstraintLayout>

```

Listing 38 Zawartość pliku item\_match.xml



Rysunek 59 Wysunięta szuflada menu z zaznaczonym elementem połączonym z MessagesFragment.



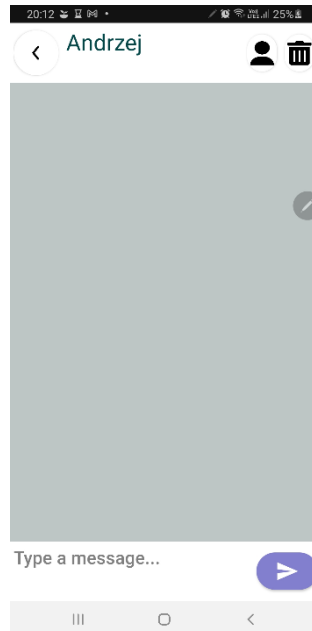
Rysunek 60 Lista dopasowanych użytkowników wewnątrz fragmentu MessagesFragment w NavigationActivity.

### 3.12 Wysyłanie i wyświetlanie wiadomości

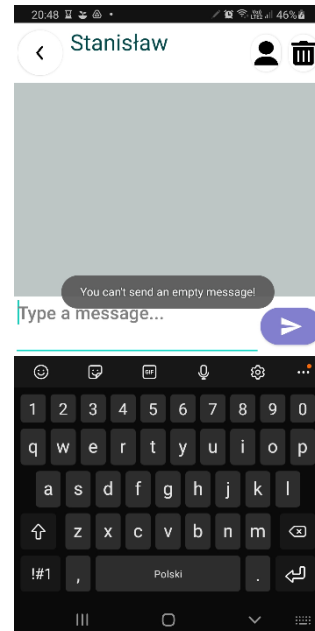
Korzystanie z aplikacji nie miałoby sensu bez możliwości nawiązania kontaktu między dopasowanymi osobami. Dzięki omówionemu wcześniej mechanizmowi polubień i odrzuceń użytkownik może wymieniać wiadomości tylko z osobami, z którymi wyrażona przez niego chęć kontaktu została odwzajemniona. Dzięki temu jeżeli chciałby urwać kontakt wystarczy, że usunie osobę ze swojej listy dopasowań, co zostanie dokładniej omówione w jednym z dalszych podrozdziałów.

Po dotknięciu jednego z elementów zawierających imiona i zdjęcia użytkowników na liście dopasowań, aplikacja natychmiast uruchamia aktywność ChatActivity. Zawiera ona cztery przyciski, edytowalne pole tekstowe, zwykłe pole tekstowe i komponent ScrollView, w którym będą wyświetlane wiadomości wymieniane między zalogowanym użytkownikiem a osobą wybraną z listy. Przycisk w lewym górnym rogu układu pozwala na powrót do listy dopasowań. Zaraz obok niego znajduje się pole tekstowe, w którym wyświetlane jest imię wybranego użytkownika z listy dopasowań, przycisk po prawej stronie tego pola pozwala wyświetlić profil wybranego użytkownika. Przycisk po prawej stronie od niego pozwala na usunięcie użytkownika z listy dopasowań. Na szarym polu wyświetlane będą wiadomości. W polu tekstowym u dołu układu należy wpisać treść wiadomości, a następnie nacisnąć fioletowy przycisk znajdujący się po prawej stronie, aby

wysłać wiadomość. Jeżeli użytkownik spróbuje nacisnąć przycisk wysyłania nie wprowadzając treści, zostanie wyświetlony komunikat informujący o braku możliwości wysłania pustej wiadomości. W przypadku próby wysłania wiadomości o długości przekraczającej 250 znaków zostanie wyświetlony komunikat informujący, że wiadomość jest zbyt długa.



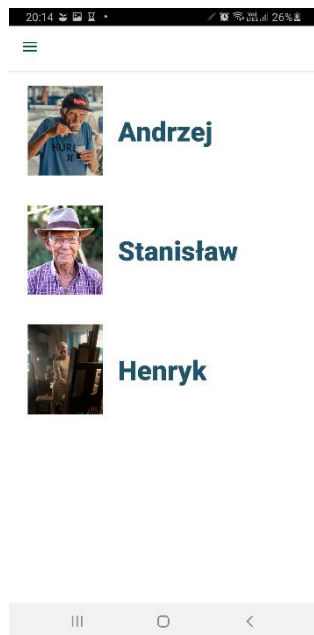
*Rysunek 61 Aktywność ChatActivity kiedy nie wymieniono jeszcze żadnych wiadomości.*



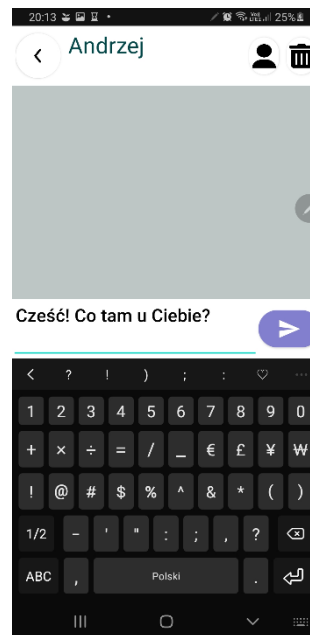
*Rysunek 62 Komunikat wyświetlany po próbie wysłania pustej wiadomości.*

```
backBtn.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        Intent backIntent = new Intent(getApplicationContext(),  
MessagesFragment.class);  
        startActivity(backIntent);  
        finish();  
    }  
});
```

*Listing 39 Uruchomienie aktywności MessageFragment następujące po wciśnięciu przycisku po lewej stronie układu*



Rysunek 63 Powrót do aktywności `MessageActivity` w wyniku naciśnięcia przycisku.



Rysunek 64 Wpisywanie treści wiadomości do edytowalnego pola tekstowego.

Po naciśnięciu fioletowego przycisku do zmiennej `messageToSend` zapisywany jest łańcuch tekstowy umieszczony przez użytkownika w komponencie `EditText`. Następnie jest sprawdzane czy łańcuch ten nie jest pusty lub dłuższy niż 250 znaków. Jeżeli wiadomość spełnia postawione jej wymagania, to wywoływana jest metoda `sendMessage()` i w komponencie `EditText` jest ustawiany pusty łańcuch tekstowy, dzięki czemu, aby napisać następną wiadomość nie ma potrzeby usuwania poprzednio wpisanego tekstu.

```

sendBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String messageToSend =
messageEditText.getText().toString().trim();
        if(messageToSend.equals("")) {
            Toast.makeText(getApplicationContext(), "You can't send
an empty message!", Toast.LENGTH_SHORT).show();
        } else {
            if(messageToSend.length() >= 251) {
                Toast.makeText(getApplicationContext(), "Your message
is too long! Type less than 250 characters.",
Toast.LENGTH_LONG).show();
            } else {
                Log.d("CHATX", "poprawnie");
                sendMessage(dbHelper.getCurrentUserId(), matchId,
messageToSend);
                messageEditText.setText("");
            }
        }
    }
});

```

Listing 40 Walidacja długości wiadomości i wywołanie metody sendMessage().

```

private void sendMessage(String sender, String receiver, String
message) {
    Map<String, Object> map = new HashMap<>();
    map.put("sender", sender);
    map.put("receiver", receiver);
    map.put("message", message);

    dbHelper.getDatabaseReference().child("Chats").push().setValue(map);
}

```

Listing 41 Definicja metody sendMessage().

Metoda readMessage() jest wywoływana w metodzie onCreate(), a zatem po starcie aktywności. Przy pomocy ValueEventListener sprawdzane jest które z elementów podrzędnych węzła Chats są wiadomościami wymienionymi między zalogowanym użytkownikiem a wybranym z listy dopasowań. Te wiadomości są dodawane do listy która wyświetlana jest wewnątrz ScrollView za pomocą adaptera ChatAdapter. Aktywności wysłane przez zalogowanego użytkownika wyświetlane są na tle w kolorze zielonym, a ich tekst jest w kolorze białym, natomiast wiadomości otrzymywane od dopasowanego użytkownika wyświetlane są na białym tle, a ich tekst ma kolor czarny.



```

private void readMessages(String currentUserId, String matchId) {
    messagesToDisplayList = new ArrayList<>();
    ValueEventListener readMessagesValueEventListener = new
    ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            messagesToDisplayList.clear();
            for(DataSnapshot dataSnapshot : snapshot.getChildren()) {
                ChatModel chat =
                dataSnapshot.getValue(ChatModel.class);
                if (chat.getReceiver().equals(currentUserId) &&
                chat.getSender().equals(matchId) ||
                chat.getReceiver().equals(matchId) &&
                chat.getSender().equals(currentUserId)) {
                    messagesToDisplayList.add(chat);
                }
                adapter = new ChatAdapter(messagesToDisplayList,
                ChatActivity.this);
                chatRecyclerView.setAdapter(adapter);
            }

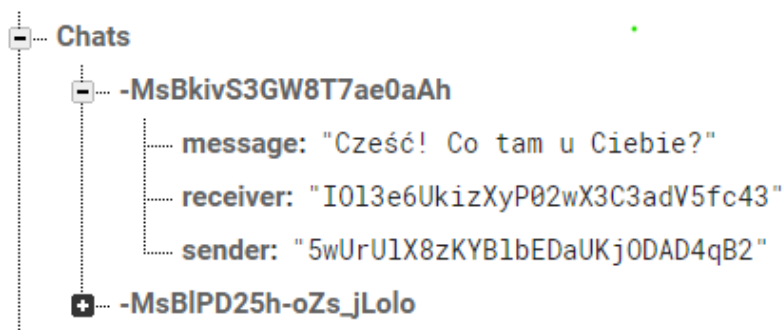
            @Override
            public void onCancelled(@NonNull DatabaseError error) {
            }
        };

        dbHelper.getDatabaseReference().child("Chats").addValueEventListener(
        readMessagesValueEventListener);
    }
}

```

Listing 42 Definicja metody readMessage().

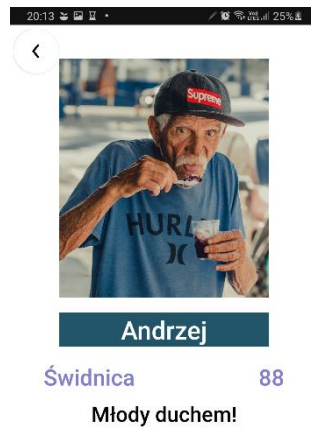
#### seniorsapp-6f519-default-rtdb



Rysunek 65 Fragment bazy danych Firebase Realtime Database zawierający węzeł Chat.

### 3.13 Wyświetlanie profilu dopasowanego użytkownika

Aby wyświetlić profil użytkownika będącego na liście dopasowań należy wybrać go z listy w MessagesFragment, a następnie nacisnąć przycisk oznaczony uproszczonym portretem znajdujący się w aktywności ChatActivity. W ten sposób uruchomiona zostanie aktywność MatchProfileActivity, która zawiera przycisk umożliwiający powrót do fragmentu MessagesFragment, komponent ImageView wyświetlający zdjęcie użytkownika i cztery komponenty TextView wyświetlające informacje o wybranym użytkowniku; imię, lokalizację, wiek i opis. Aktywność ta działa analogicznie do fargmentu MyProfileFragment, w którym wyświetlane są te same informacje, ale dotyczące zalogowanego użytkownika.



Rysunek 66 Aktywność MatchProfileFragment wyświetlająca informacje o wybranym użytkowniku.

Przy pomocy ValueEventListener z bazy danych pobierane są informacje o wybranym użytkowniku, które następnie są wyświetlane przy pomocy metody displayUserInfo() działającej identycznie do jej odpowiednika zdefiniowanego w aktywności MyProfileFragment. Dzięki przesłaniu UID wybranego użytkownika z ChatActivity, jasne jest które elementy podrzędne węzła Users należy pobrać.

```

Intent chatIntent = getIntent();
Bundle chatBundle = chatIntent.getExtras();
matchId = chatBundle.getString("matchId");

imageView = findViewById(R.id.usersImage);
nameTxt = findViewById(R.id.usersName);
localisationTxt = findViewById(R.id.usersLocalisation);
ageTxt = findViewById(R.id.usersAge);
descriptionTxt = findViewById(R.id.usersDescription);
backBtn = findViewById(R.id.backBtn);

ValueEventListener matchDataValueEventListener = new
 ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        if(snapshot.hasChildren()) {
            matchName = snapshot.child("name").getValue().toString();
            matchLocalisation =
snapshot.child("localisation").getValue().toString();
            matchAge = snapshot.child("age").getValue().toString();
            matchDescription =
snapshot.child("description").getValue().toString();
            matchImageName =
snapshot.child("mainPictureName").getValue().toString();
            matchImageUri =
snapshot.child("imageUri").getValue().toString();

            displayUsersInfo();
        }
    }

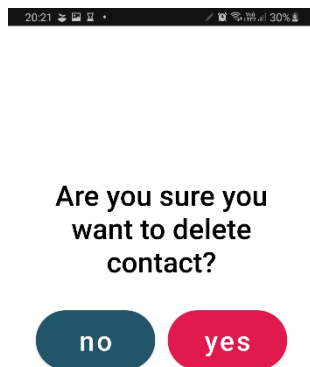
    @Override
    public void onCancelled(@NonNull DatabaseError error) {
    }
};

```

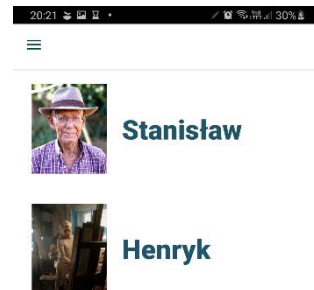
*Listing 43 Zawartość metody onCreate w pliku MatchProfileActivity.java.*

### 3.14 Usunięcie użytkownika z listy dopasowań

W wyniku różnych sytuacji może się zdarzyć, że użytkownik nie będzie chciał dłużej utrzymywać kontaktu z wybranymi osobami, z tego też powodu zdecydowałam się na stworzenie możliwości usunięcia użytkownika z listy dopasowań. Dzięki takiemu działaniu nie będzie dalszej możliwości wymiany wiadomości pomiędzy zalogowanym użytkownikiem, a użytkownikiem usuniętym przez niego z listy dopasowań. Aby to zrobić należy wybrać użytkownika z listy, a następnie po uruchomieniu aktywności ChatActivity nacisnąć przycisk oznaczony ikoną kosza na śmieci. Naciśnięcie tego przycisku spowoduje uruchomienie aktywności DeleteMatchActivity. W tej aktywności użytkownikowi ukazywany jest komunikat proszący o potwierdzenie chęci usunięcia użytkownika z listy dopasowań. Takie rozwiązanie pozwala uniknąć przypadkowego usunięcia użytkownika, podczas np. chęci naciśnięcia przycisku wyświetlającego profil użytkownika, który znajduje się w bardzo niskiej odległości.



Rysunek 67 Aktywność DeleteMatchActivity.



Rysunek 68 Fragment MessagesFragment po usunięciu użytkownika Maciej z listy dopasowań.

Usuwanie użytkownika odbywa się poprzez zmianę wartości „Liked”, z „yes” na „no”. Dzięki temu podczas uruchamiania fragmentu MessagesFragment do listy dopasowań nie zostanie dodany usunięty użytkownik, a tym samym nie zostanie on wyświetlony w tym fragmencie.

```
private void deleteMatch() {
    Map<String, Object> map = new HashMap<>();
    map.put("Liked", "no");

    dbHelper.getCurrentUserReference().child("Connections").child(matchId).updateChildren(map);
    Toast.makeText(getApplicationContext(), "Contact deleted.",
    Toast.LENGTH_SHORT);
}
```

Listing 44 Definicja metody deleteMatch().

Po naciśnięciu przycisku opatrzonego napisem „yes”, a tym samym potwierdzeniu chęci usunięcia użytkownika z listy dopasowań, w metodzie onClick przypisanej do tego przycisku wywoływana jest metoda deleteMatch().

#### seniorsapp-6f519-default-rtdb



Rysunek 69 Zmiana wartości elementu podrzędnego "Liked" na "no".

### 3.15 Aktualizacja informacji o użytkowniku

Gdy użytkownik wybierze pozycję Settings w menu w aktywności NavigationActivity, aplikacja wyświetli fragment SettingsFragment, w którym użytkownik ma możliwość zmiany wprowadzonych wcześniej informacji na swój temat. W układzie tego fragmentu w komponentach EditText wyświetlane są wprowadzone na jego temat informacje, które są pobierane z bazy danych w momencie uruchomienia fragmentu. Metodą za to odpowiedzialną jest displayCurrentUsersInfo(). Aby zmienić jakąś informację należy ją edytować w odpowiednim polu, a następnie po edycji wszystkich wybranych przez siebie informacji należy nacisnąć przycisk „confirm changes”. W ten sposób wartości wprowadzone w edytowalnych polach tekstowych nadpiszą te dotychczas znajdujące się w bazie danych. Każde z pól jest walidowane w ten sam sposób jak przy pierwszym wprowadzaniu tych informacji podczas rejestracji. Aktualizowaniem informacji w bazie danych zajmują się metody loadNewData() (to w niej znajduje się walidacja każdego z pól) oraz addNewDataToDataBase() (umieszcza nowe informacje w bazie).

```
private void displayCurrentUsersInfo() {
    if (currentUserImageName.length() > 27) {
        Picasso.get()
            .load(currentUserImageUri)
            .fit()
            .centerCrop()
            .rotate(90)
            .into(imageView);
    } else {
        Picasso.get()
            .load(currentUserImageUri)
            .fit()
            .centerCrop()
            .into(imageView);
    }
    nameTxt.setText(currentUserName);
    localisationTxt.setText(currentUserLocalisation);
    ageTxt.setText(currentUserAge);
    descriptionTxt.setText(currentUserDescription);
}
```

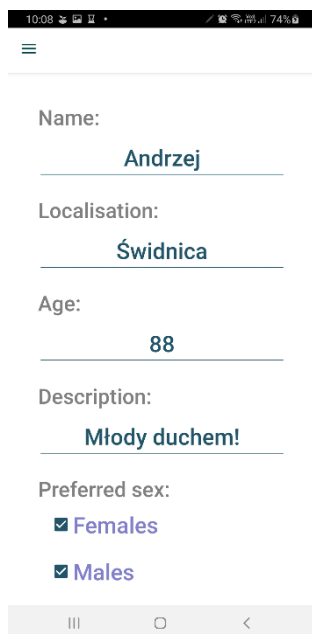
Listing 45 Definicja metody displayCurrentUsersInfo() w pliku SettingsFragment.

```

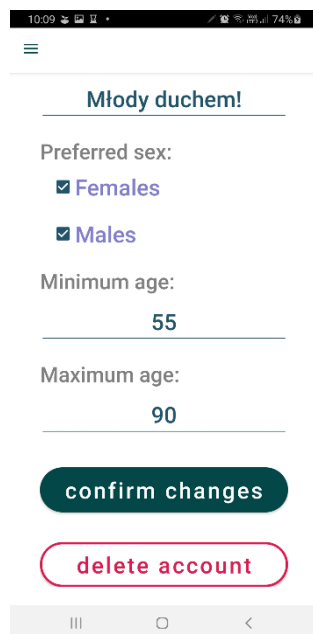
private boolean loadNewData() {
    currentUserName = nameTxt.getText().toString().trim();
    currentUserLocalisation =
localisationTxt.getText().toString().trim();
    currentUserAge = ageTxt.getText().toString().trim();
    currentUserDescription =
descriptionTxt.getText().toString().trim();
    currentUserMinPrefAge = minAgeTxt.getText().toString().trim();
    currentUserMaxPrefAge = maxAgeTxt.getText().toString().trim();
    if(femalesCB.isChecked() && malesCB.isChecked()) {
        currentUserPreferredSex = "both";
    } else if(femalesCB.isChecked()) {
        currentUserPreferredSex = "female";
    } else if(malesCB.isChecked()) {
        currentUserPreferredSex = "male";
    }
    try {
        age = Integer.parseInt(currentUserAge);
        minAge = Integer.parseInt(currentUserMinPrefAge);
        maxAge = Integer.parseInt(currentUserMaxPrefAge);
    } catch (NumberFormatException exception) {
        if (age != 0 || minAge != 0 || maxAge != 0) {
            Toast.makeText(getContext(), "Incorrect age format.",
Toast.LENGTH_LONG).show();
        }
    }
    if (currentUserName.isEmpty() ||
currentUserLocalisation.isEmpty() || currentUserAge.isEmpty() ||
currentUserDescription.isEmpty() || currentUserMinPrefAge.isEmpty()
|| currentUserMaxPrefAge.isEmpty() || (!malesCB.isChecked() &&
!femalesCB.isChecked())) {
        Toast.makeText(getContext(), "Please add all required
information.", Toast.LENGTH_SHORT).show();
    } else {
        if (containsLettersOnly(currentUserName) &&
containsLettersOnly(currentUserLocalisation)) {
            if (age >= 130 || age < 18 || minAge >= 130 || minAge <
18 || maxAge >= 130 || maxAge < 18 || (minAge >= maxAge)) {
                Toast.makeText(getContext(), "Insert correct age, it
must be greater or equal to 18.", Toast.LENGTH_LONG).show();
            } else {
                currentUserName = currentUserName.substring(0,
1).toUpperCase() + currentUserName.substring(1).toLowerCase();
                currentUserLocalisation =
currentUserLocalisation.substring(0, 1).toUpperCase() +
currentUserLocalisation.substring(1).toLowerCase();
                return true;
            }
        } else {
            Toast.makeText(getContext(), "Name and localisation must
contain only letters.", Toast.LENGTH_LONG).show();
        }
    }
    return false;
}

```

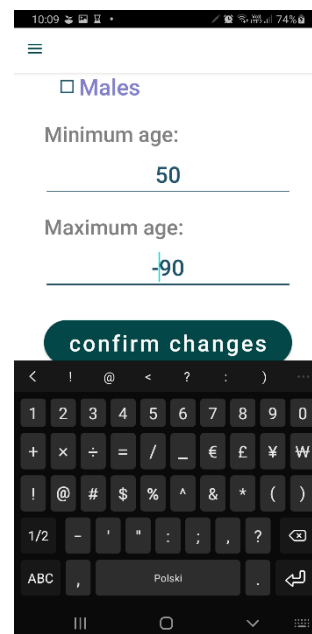
Listing 46 Definicja metody loadNewData() w pliku SettingsFragment.java.



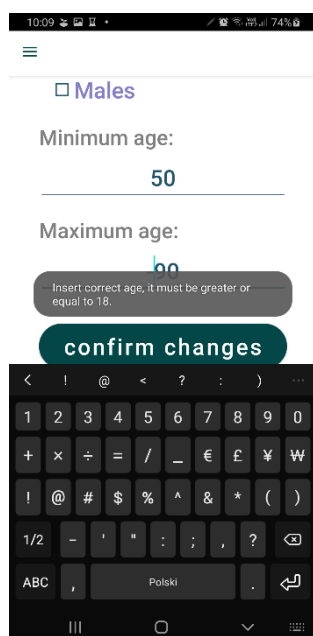
Rysunek 70 Pierwsza część układu fragmentu SettingsFragment.



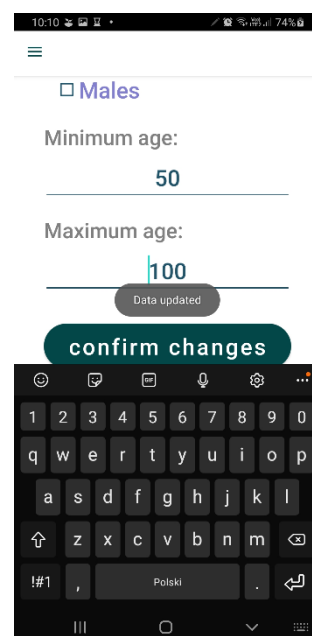
Rysunek 71 Druga część układu fragmentu SettingsFragment widoczna po scrollowaniu



Rysunek 72 Edycja maksymalnego wieku na niemożliwy.

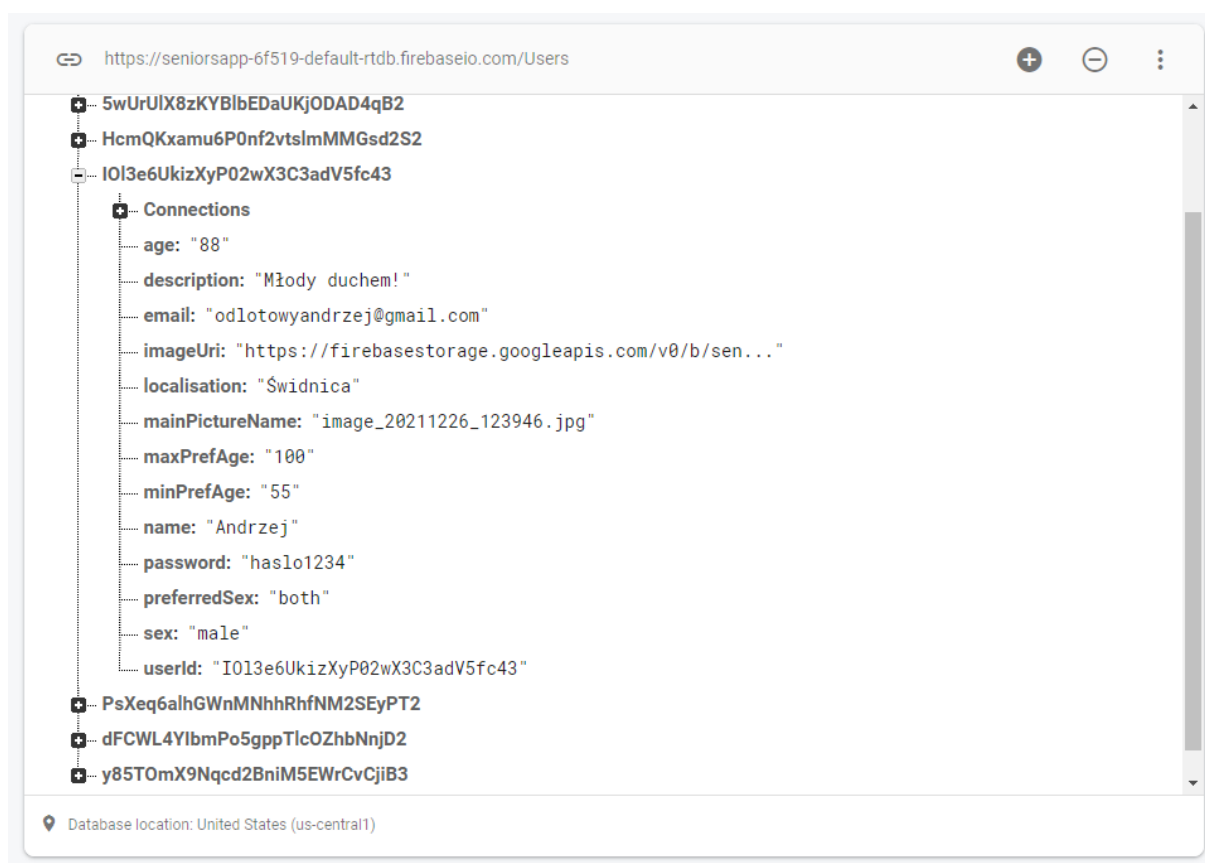


Rysunek 73 Wyświetlenie komunikatu o wpisaniu nieprawidłowego wieku.



Rysunek 74 Wyświetlenie komunikatu o poprawnej aktualizacji danych po wciśnięciu przycisku "confirm changes".

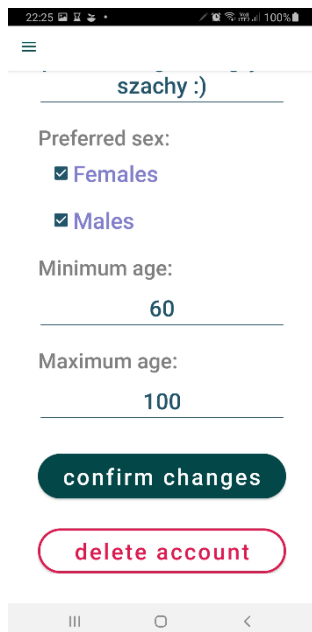




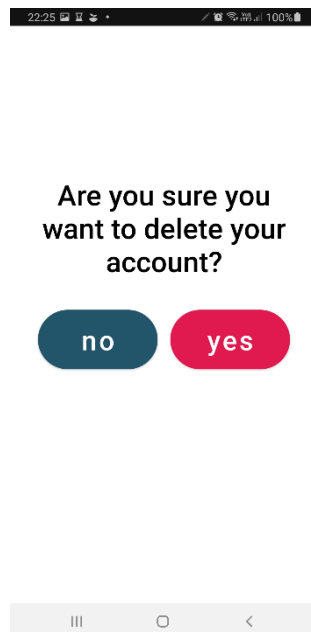
Rysunek 75 Zaktualizowane informacje w Firebase Realtime Database.

### 3.16 Usuwanie konta

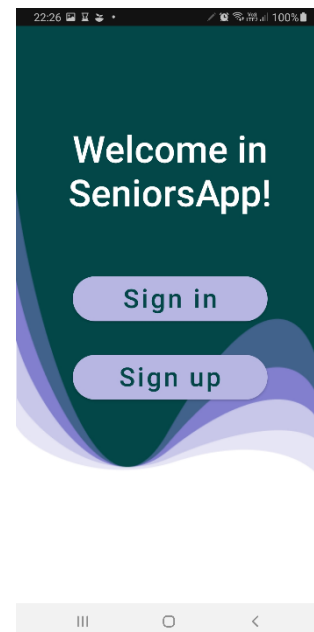
W każdej chwili użytkownik może zrezygnować z posiadania konta w aplikacji. Ze względu na to dodana została funkcjonalność usuwania konta, która zapewnia usunięcie wszystkich danych wprowadzonych przez użytkownika z bazy Firebase. Aby usunąć konto należy wybrać w Menu pozycję Settings, dzięki temu zostanie wyświetlony fragment SettingsFragment. Na samym dole jego układu znajduje się przycisk oznaczony napisem „delete account”, którego naciśnięcie uruchomi aktywność DeleteAccountActivity zawierającą komunikat z prośbą o potwierdzenie chęci usunięcia konta i dwa przyciski oznaczone napisami „no” i „yes”. Jeżeli użytkownik naciśnie przycisk „no”, zostanie uruchomiony fragment FindNewFriendsFragment, natomiast jeżeli użytkownik naciśnie przycisk „yes” jego konto oraz wszystkie dane zostaną usunięte poprzez wywołanie funkcji `deleteCurrentUserAccount()`. Następnie zostanie uruchomiona aktywność StartingActivity.



Rysunek 76 Dolna część fragmentu *SettingsFragment* zawierająca przycisk "delete account".



Rysunek 77 Aktywność *DeleteAccountActivity*.



Rysunek 78 Aktywność *StartingActivity* uruchomiona po naciśnięciu przycisku "yes".

W metodzie `deleteCurrentUserAccount()` dane są odczytywane przy pomocy `ValueEventListener`. Następnie zapisywana jest referencja do Uri dodanego zdjęcia, po czym za pomocą tej referencji jest ono usuwane z `Firebase Storage` przy pomocy `delete()`. W przypadku udanego usunięcia zdjęcia, za pomocą metody `removeValue()` usuwane są wszystkie informacje o zalogowanym użytkowniku z węzła `Users` w `Firebase Realtime Database`. Jeżeli ta operacja również zostanie zakończona pomyślnie, to usunięte zostanie konto użytkownika z `FirebaseAuthentication`, po czym zostanie uruchomiona aktywność `StartingActivity`.

```

private void deleteUserAccount() {
    currentUserReference.addValueEventListener(new
ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            if(snapshot.hasChildren()) {
                imageUrl =
snapshot.child("imageUrl").getValue().toString();
                imageStorageReference =
firebaseStorage.getReferenceFromUrl(imageUrl);

imageStorageReference.delete().addOnSuccessListener(new
OnSuccessListener<Void>() {
                    @Override
                    public void onSuccess(Void aVoid) {

currentUserReference.removeValue().addOnSuccessListener(new
OnSuccessListener<Void>() {
                            @Override
                            public void onSuccess(Void aVoid) {

currentUser.delete().addOnCompleteListener(new
OnCompleteListener<Void>() {
                                    @Override
                                    public void
onComplete(@NonNull Task<Void> task) {
                                            if (task.isSuccessful()) {
                                                Intent intent = new
Intent(getApplicationContext(), StartingActivity.class);

intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                                                startActivity(intent);
                                                finish();
                                            } else {

Toast.makeText(getApplicationContext(),
task.getException().getMessage(), Toast.LENGTH_LONG).show();
                                            }
                                        }
                                    });
                                }
                            });
                        }
                    });
                }
            }
        }
    });
}
}

```

Listing 47 Definicja metody deleteUserAccount().

## 4. Podsumowanie

Niniejszy rozdział zawiera podsumowanie pracy efektów pracy wykonanej podczas projektowania i implementacji aplikacji. Zamieszczono tu przemyślenia dotyczące możliwości dalszego rozwoju powstałego oprogramowania, opis trudności które zostały napotkane oraz wnioski końcowe podsumowujące w jakim stopniu aplikacja została zrealizowana zgodnie z zamierzeniami.

### 4.1 Problemy implementacyjne

Największym wyzwaniem podczas implementacji aplikacji okazało się odpowiednie wykorzystanie metod asynchronicznych takich jak `onDataChange()`. Ze względu na specyfikę metod tego typu zdarzały się sytuacje, w których listy były zapełniane danymi dopiero po wykorzystaniu ich przez inną funkcję, co skutkowało wykorzystywaniem pustych list. Udało się zażegnać ten problem poprzez wywoływanie metod pracujących na listach wewnątrz końcowej części definicji metody `onDataChange()`, dzięki czemu nie było możliwości wywołania tych metod przed zakończeniem uzupełniania listy przy pomocy danych z bazy Firebase RealtimeDatabase. Innym wyzwaniem okazała się również implementacja Rozsuwanego Menu z aktywności `NavigationActivity`. Ze względu na mnogość plików które były niezbędne do utworzenia w tym celu, proces ten potrwał znacznie dłużej niż początkowo szacowałam.

### 4.2 Możliwości rozwoju aplikacji

Wszystkie łańcuchy tekstowe wyświetlane domyślnie przez aplikację (komunikaty, napisy umieszczone na elementach takich jak przyciski, menu i tp.) zostały umieszczone w pliku `strings.xml`. Takie rozwiązanie pozwala na proste rozszerzenie wersji językowych, w których występuje aplikacja, w tym celu wystarczy podmienić plik `strings.xml` na odpowiedni dla danej wersji językowej, a zmiany w kodzie trzeba wykonać tylko w tym pliku tłumacząc poszczególne łańcuchy. Korzystną zmianą byłoby dodanie powiadomień, dzięki czemu użytkownik nie musiałby uruchamiać aplikacji, aby sprawdzić czy dostał nową wiadomość. Kolejnym udogodnieniem byłoby dodanie klasycznych gier do aplikacji, dzięki takiej funkcjonalności sparowani użytkownicy mogliby rywalizować w grach takich jak szachy i warcaby co stworzyłoby nowe możliwości wspólnego spędzania czasu mimo dzielącej odległości. Bardzo przydatne byłoby również dodanie sprawdzania lokalizacji użytkownika i wyświetlania w pierwszej kolejności potencjalnych znajomych którzy są najbliżej. Mogłoby to ułatwić nawiązywanie relacji, które następnie można przenieść do świata rzeczywistego.

## 4.3 Wnioski końcowe

W niniejszej pracy inżynierskiej opisano działanie i sposób implementacji aplikacji mobilnej, której przeznaczeniem jest ułatwienie nawiązywania kontaktów towarzyskich seniorom. Aplikacja została specjalnie dostosowana do potrzeb tej grupy użytkowników poprzez odpowiednie dostosowanie interfejsu użytkownika, tak aby każda funkcjonalność była prosta w obsłudze, a jej elementy dobrze widoczne. Aplikacja została napisana w języku Java na platformę Android. Bardzo dobrym rozwiązaniem okazało się wykorzystanie platformy Firebase, co znacznie ułatwiło pracę z danymi. Wybór języka Java okazał się bardzo korzystny, ponieważ jest on uniwersalny i ma bardzo wyczerpująco napisaną dokumentację. Wszystkie wymagania postawione podczas części projektowej tworzenia aplikacji zostały spełnione. Każda funkcjonalność aplikacji działa zgodnie z założeniami i jest odpowiednio zabezpieczona przed wprowadzeniem danych niewłaściwego typu lub o nieprawidłowych wartościach, które mogłyby niekontrolowanie zatrzymać działanie aplikacji i utrudniać użytkownikom wygodne z niej korzystanie.

## 4.4 Spis listingów

Listing 1 Walidacja informacji wprowadzonych przez użytkownika po naciśnięciu przycisku „OK”.	10
Listing 2 Definicja funkcji registerUser().	12
Listing 3 Przypisanie danych wprowadzonych do odpowiednich zmiennych i wywołanie metody loginUser().	14
Listing 4 Definicja metody loginUser().	15
Listing 5 Definicja metody logOut().	16
Listing 6 Wywołanie metody logOut() w przypadku wybrania elementu o id nav_logout w pliku NavigationActivity.java wewnątrz metody onNavigationItemSelected().	16
Listing 7 Definicja metody isConnected() w pliku RegistrationActivity.java.	17
Listing 8 Kod wykonywany w następstwie naciśnięcia przycisku OK w aktywności RegistrationActivity i wywołanie metody isConnected() w setOnClickListener() wywołanym na przycisku w pliku RegistrationActivity.java.	18
Listing 9 Kod wykonywany w następstwie naciśnięcia przycisku Try Again w pliku RegisterInternetAccessActivity.java	20
Listing 10 Pobranie danych z Bundle i przypisanie ich do zmiennych email i password.	21
Listing 11 Listener ustawiony na przełączniku.	21

Listing 12 Definicja metody containsLettersOnly() w pliku UsersInfoActivity.java .....	22
Listing 13 Walidacja danych wprowadzonych przez użytkownika i przesłanie jej za pomocą Bundle do aktywności PreferencesActivity. ....	23
Listing 14 Przypisanie dotychczas zebranych informacji o użytkownikach do odpowiednich zmiennych. ....	25
Listing 15 Walidacja danych wprowadzonych przez użytkownika i wysłanie ich do aktywności ProfileInfoActivity .....	26
Listing 16 Składowe, konstruktor i definicja metody addUserToDB() klasy DataBaseHelper. ....	28
Listing 17 Walidacja opisu wprowadzonego przez użytkownika, utworzenie obiektu, wywołanie metody addUserToDB() na obiekcie klasy DataBaseHelper.....	28
Listing 18 Wywołanie metody askCameraPermission() w momencie naciśnięcia przycisku „take a picture”.....	30
Listing 19 Definicja metody askCameraPermission() i przesłonięcie metody onRequestPermissionsResult(). ....	31
Listing 20 Definicja metody createImageFile().....	31
Listing 21 Przesłonięcie metody onActivityResult(). ....	32
Listing 22 Definicja metody dispatchTakePictureIntent(). ....	32
Listing 23 Definicja metody uploadImageToFirebase() odpowiedzialnej za dodanie zdjęcia do bazy danych Firebase Storage. ....	33
Listing 24 Zawartość pliku nav_menu.xml definiującego formę menu, ikony i nazwy elementów.....	39
Listing 25 Otwarcie fragmentu FindNewFriendsFragment podczas startu aktywności NavigationActivity. ....	39
Listing 26 Przesłonięcie metody onBackPressed(). ....	40
Listing 27 Pobranie z bazy danych informacji o użytkowniku, aby wyświetlić je w profilu. ....	42
Listing 28 Definicja metody displayCurrentUsersInfo() w pliku MyProfileFragment.java .....	42
Listing 29 Definicja metody checkUsersPreferences(). ....	43
Listing 30 Definicja metody listPotentialMatches().....	44
Listing 31 Definicja metody addItemModelList().....	45
Listing 32 Definicja metody setData() w pliku CardStackAdapter.java .....	46
Listing 33 Mechanizm przesuwania kart i aktualizacji bazy danych. ....	47

Listing 34 Definicja metody listLikedUsers() w pliku MessagesFragment.java. ....	49
Listing 35 Definicja metody listMatches() w pliku MessagesFragment.java. ....	50
Listing 36 Definicja metody addItemMatchModelList() w pliku MessagesFragment.java. .....	51
Listing 37 Definicja metody setData() w pliku MatchesListAdapter.java.....	52
Listing 38 Zawartość pliku item_match.xml.....	52
Listing 39 Uruchomienie aktywności MessageFragment następujące po wciśnięciu przycisku po lewej stronie układu .....	54
Listing 40 Walidacja długości wiadomości i wywołanie metody sendMessage(). ....	56
Listing 41 Definicja metody sendMessage(). ....	56
Listing 42 Definicja metody readMessage(). ....	57
Listing 43 Zawartość metody onCreate w pliku MatchProfileActivity.java.....	59
Listing 44 Definicja metody deleteMatch().....	61
Listing 45 Definicja metody displayCurrentUsersInfo() w pliku SettingsFragment. ....	62
Listing 46 Definicja metody loadNewData() w pliku SettingsFragment.java. ....	63
Listing 47 Definicja metody deleteCurrentUserAccount(). ....	67

## 4.5 Literatura

- [1] M. Płonkowski, Android Studio. Tworzenie aplikacji mobilnych, Helion, 2017
- [2] D. Jemerov, S. Isakova, Kotlin w Akcji, Helion, 2018
- [3] B. Phillips, C. Stewart, K. Marsicano, Programowanie aplikacji dla Androida. The Big Nerd Ranch Guide, Wyd.III, Helion, 2017
- [4] J. Annuzzi Jr., L. Darcey, S. Conder, Wyd. 5, Helion, 2016
- [5] Dokumentacja Androida <https://developer.android.com/guide> Data dostępu: 05.10.2021
- [6] Dane statystyczne dotyczące użytkowania Internetu przez seniorów <https://www.pewresearch.org/internet/2017/05/17/technology-use-among-seniors/> Data dostępu: 19.12.2021
- [7] Dane statystyczne dotyczące użytkowania mediów społecznościowych przez seniorów <https://www.pewresearch.org/internet/fact-sheet/social-media/> Data dostępu: 19.12.2021
- [8] Dane statystyczne dotyczące popularności poszczególnych systemów operacyjnych przeznaczonych na urządzenia mobilne <https://gs.statcounter.com/os-market-share/mobile/worldwide> Data dostępu: 19.12.2021

Licencja: Creative Commons

[9] Baza zdjęć <https://www.pexels.com/pl-pl/creative-commons-images/>

Data dostępu: 21.12.2021

[10] Dokumentacja elektroniczna temat robienia zdjęć w aplikacjach na platformę Android przy pomocy aparatu

<https://developer.android.com/training/camera/photobasics>

[11] Dokumentacja dotycząca udzielania zgody na dostęp do aplikacji:

<https://developer.android.com/training/permissions/requesting>

[12] Artykuł na temat robienia zdjęć za pomocą aplikacji aparatu i wybieranie zdjęcia z galerii: <https://smallacademy.co/blog/android/capture-image-and-display-in-imageview/>

[13] Opis sposobu implementacji kart potencjalnych znajomych:

<https://github.com/JabirDev/TinderSwipeFragment>

[14] Artykuł na temat fragmentów: <https://damianchodorek.com/kurs-android-fragment-selektor-6/>

[15] Artykuł na temat wydobywania danych z Realtime Database:

<https://firebase.google.com/docs/database/android/read-and-write>

[16] David Griffiths, Dawn Griffiths, Android. Programowanie aplikacji. Rusz głową! Wydanie II, Helion 2018

[17] Cay S. Horstmann, Java. Podstawy. Wydanie XI, Helion 2019

[18] Dokumentacja Firebase <https://firebase.google.com/docs>