AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA

PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

Menadżer wydatków

Autor:
Marek Pichniarczyk
Mikołaj Kwiatek
Karol Wolski

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2022

Spis treści

1. Ogólne określenie wymagań

Aplikacja ma za zadanie podumowywać wydatki użytkownika. Klient będzie mógł wprowadzać dane dotyczące danego wydatku oraz podporządkować je katerogii (np. spożywcze/transport/podatki). Ma isnieć możliwość dodania także miesięcznych przychodów z których odejmowane będą wydatki. Do każdej pozycji będzie możliwość dodania zdjęcia paragonu lub faktury. Program użytkowy ma obliczać statystyki z okresu. Użytkownik będzie miał możliwość wprowadzenia dziennego limitu wydatków, po przekroczeniu zostanie o tym poinformowany. Wraz z końcem dnia aplikacja wyśle podsumowanie o wydanych przez użytkownika wydatkach. Użytkownik ma mieć możliwość wyboru trybu ciemnego lub jasnego. Odblokowanie aplikacji ma być za pomocą danych biometrycznych, głównym celem aplikacji ma być osobiste kontrolowanie wydatków oraz możliwość ich podporządkowania do różnych kategorii by mieć wgląd w różne pozycje. Ważną funkcją w całej aplikacji ma być wgląd do ostatnich wydatków, by móc zobaczyć na co ostatnio wydaliśmy pieniądze oraz szczegóły ich, tzn. data, kategoria oraz kwota. Aplikacja ma być bardzo prosta w obsłudze, bez niepotrzebnych funkcji, aczkolwiek potrzebny będzie tryb ciemny oraz jasny aplikacji, ponieważ niektórzy użytkownicy preferują tryb ciemny, który nie razi w oczy w wieczornych porach, ale również musi być możliwość ustawienia trybu jasnego. Baza danych musi być umieszczona w sieci, ponieważ użytkownik musi mieć możliwość korzystania z programu na różnych urządzeniach oraz mieć możliwość integracji w wydatki realtime - tzn. użytkownik na jednym telefonie może dodać wydatek, a na drugim w tym samym momencie, bez odświeżania aktywności ma się wyświetlić ten wydatek.

Rys. 1.1. Logo

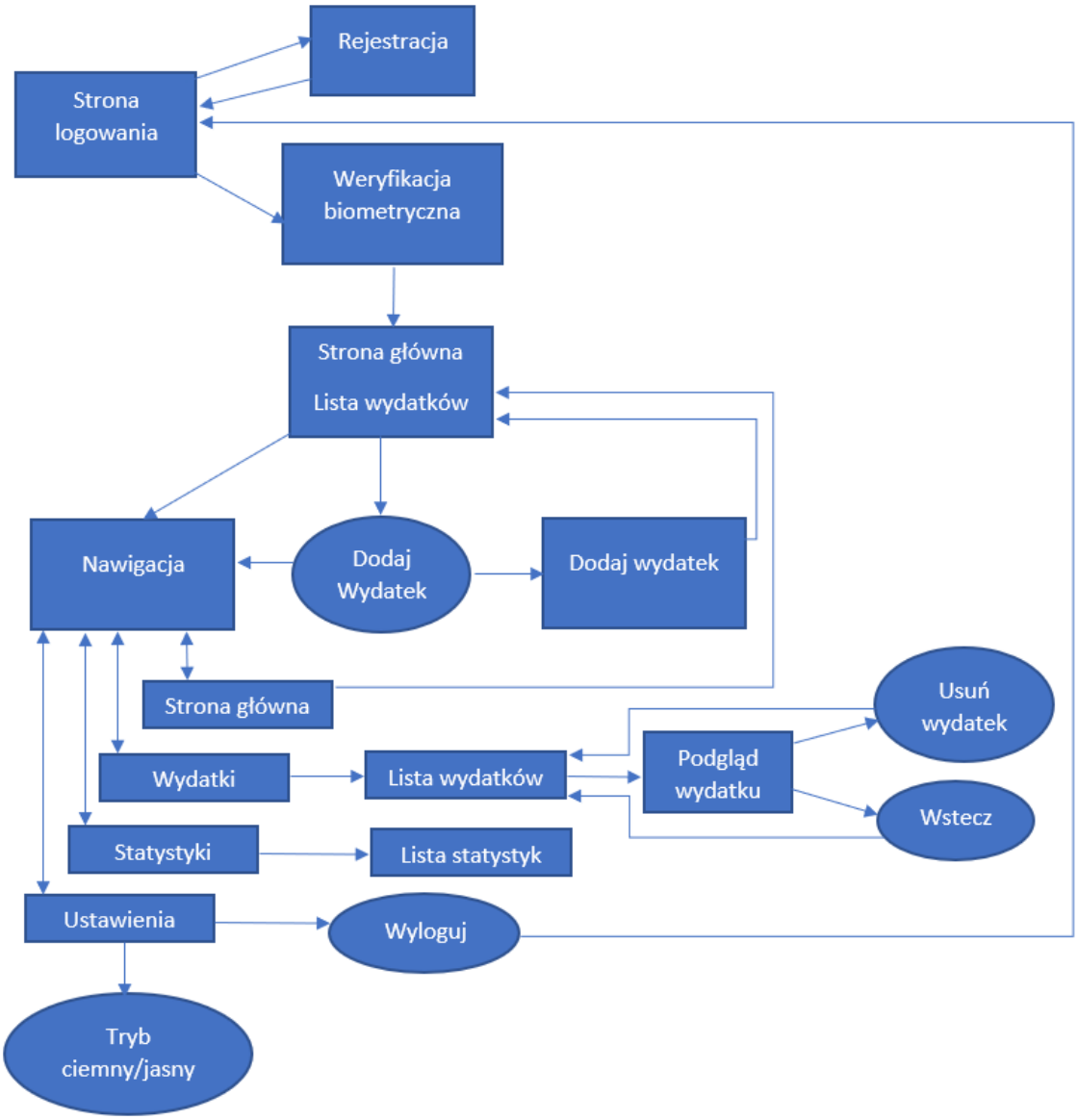
2. Określenie wymagań szczegółowych

2.1. Opis aplikacji

Aplikacja będzie pisana w środowisku Android Studio jęzkykiem Java. Celem aplikacji ma być kontrolowanie wydatków użytkownika, zakresem działań będą między innymi dodanie swojego dochodu z którego będziemy odejmować wydatki i wyliczać później kwotę pozostałą. Przy włączeniiu aplikacji ukazuje nam się formularz logowania, poniżej jest możliwość zarejestrowania użytkownika, po pomyślnym zalogowaniu aplikacja odsyła nas do strony głównej. Saldo konta będzie wyświetlane na głównej części aplikacji, będzie też przycisk do dodania wydatku - on przekieruje nas do widoku formalarza dodawania wydatku. Użytkownik będzie podawać tytuł wydatku, kwote wydatku, kategorię(np. transport, żywność etc.), oraz zdjęcie paragonu lub faktury, możliwe będzie również sporządzenie nagrania notatki głosowej do danego wydatku, do wydatku będzie pobierana data dodania z systemu. Będzie możliwość usunięcia każdego wydatku - to wszystko będzie w głównym widoku. Podkategoria statystyka będzie wyliczała dane statystki. Aplikacja będzie wysyłała powiadomienia o przekroczeniu limitu dziennego wydatków oraz pod koniec dnia wydanych pieniędzy. Baza danych będzie prowadzona na Firebase Realtime Database, dodana będzie możliwość również odblokowania aplikacji za pomocą danych biometrycznych. Logowanie do aplikacji będzie oparte na systemie Firebase Authentication, dzięki czemu możliwe będzie rejestracja/logowanie użytkoników przy użyciu internetu.
Podkategoria statystki: suma wydatków, dochód, suma wydatków podporządkowana kategoriom.
Podkategoria wydatki: wszystkie wydatki, możliwość usunięcia dowolnych wydatków.
Podkategoria ustawienia: przycisk switch dzięki któremy będziemy mieć możliwość przechodzenie pomiędzy trybem jasnym i ciemnym aplikacji, dane użytkownika oraz przycisk wyloguj.

2.2. Schemat

Rysunek 2.1 (s. 5) przedstawia schemat aplikacji. Poruszanie się po aplikacji rozpoczynamy od strony logowania, następnie możemy przejść do rejestracji lub zalogować się i zweryfikować dane biometryczne, następnie znajdujemy się na stronie głównej, tam mamy przcyisk "Dodaj wydatek", dzięki niemu można przejść do ak-



Rys. 2.1. Schemat aplikacji

tywności która umożliwia nam tą funkcję, po dodaniu wydatku przenosimy się na stronę główną. Nawigacja umożliwia nam poruszanie się po aktywnościach aplikacji, przejście do "wydatki"przenosi nas do listy wydatków, a po kliknięciu w wydatek przenosimy sie do podglądu tego wydatku, tam jest możliwość usunięcia wydatku oraz funkcja "wstecz", która przenosi nas z powrotem na listę wydatków. "Statystyki"ukazuje nam listę statystyk. "Ustawienia"posiada funkcję zmiany trybu jasny-/ciemny po kliknięciu, oraz przecisk "Wyloguj", który wylogowuje nas z aplikacji i przenosi do strony logowania.

2.3. Wykorzystywane technologie

2.3.1. Android

Android – system operacyjny z jądrem Linux dla urządzeń mobilnych takich jak telefony komórkowe, smartfony, tablety (tablety PC) i netbooki. W 2013 roku był najpopularniejszym systemem mobilnym na świecie. Wspomniane jądro oraz niektóre inne komponenty, które zaadaptowano do Androida opublikowane są na licencji GNU GPL. Android nie zawiera natomiast kodu pochodzącego z projektu GNU. Cecha ta odróżnia Androida od wielu innych istniejących obecnie dystrybucji Linuksa. Początkowo był rozwijany przez firmę Android Inc. (kupioną później przez Google), następnie przeszedł pod skrzydła Open Handset Alliance.
Android zrzesza przy sobie dużą społeczność deweloperów piszących aplikacje, które poszerzają funkcjonalność urządzeń. W sierpniu 2014 było dla tego systemu dostępnych ponad 1,3 miliona aplikacji w Google Play (wcześniej Android Market).
Według danych serwisu StatCounter z kwietnia 2017 roku Android miał największe udziały na rynku systemów operacyjnych.

2.3.2. Java

Java – współbieżny, oparty na klasach, obiektowy język programowania ogólnego zastosowania. Został stworzony przez grupę roboczą pod kierunkiem Jamesa Goslinga z firmy Sun Microsystems. Java jest językiem tworzenia programów źródłowych kompilowanych do kodu bajtowego, czyli postaci wykonywanej przez maszynę wirtualną. Język cechuje się silnym typowaniem. Jego podstawowe koncepcje zostały przejęte z języka Smalltalk (maszyna wirtualna, zarządzanie pamięcią) oraz z języka C++ (duża część składni i słów kluczowych).
Autorzy języka Java określili kilkanaście kluczowych koncepcji swojego języka. Najważniejsze z nich to:
Obiektowość, dziedziczenie, niezależność od architektury, sieciowość i obsługa programowania rozproszonego.

2.3.3. Android Studio

Android Studio – oficjalne środowisko programistyczne na platformę Android, zbudowane na podstawie oprogramowania IntelliJ IDEA od JetBrains. Może zostać zainstalowane na systemach Windows, macOS i opartych na Linuksie. Przed wydaniem Android Studio, głównym środowiskiem programistycznym dla systemu Android było Eclipse.

Android Studio zaprezentowano 15 maja 2013 roku na konferencji Google I/O. Pierwsze stabilne wydanie (1.0) nastąpiło w grudniu następnego roku. Środowisko obsługuje języki programowania: Java, C++ a także Kotlin. 7 maja 2019 ogłoszono, że ten ostatni zastąpił Javę jako preferowany do pisania aplikacji na system Android. Nie spowodowało to usunięcia wsparcia dla któregokolwiek z języków.
W Android Studio wbudowane jest narzędzie do projektowania widoków aplikacji w trybie WYSIWYG. Środowisko wspiera również emulator Androida, dzięki któremu aplikacje można testować bez dostępu do urządzenia mobilnego.
Instalacja Android Studio jest możliwa na komputerach wyposażonych w 64-bitowy procesor ze wsparciem dla wirtualizacji, co najmniej 8 GB pamięci RAM i tyle samo miejsca na dysku.

2.3.4. Firebase

Firebase to stworzona przez Firebase Inc. w 2011 roku platforma, która umożliwia tworzenie aplikacji mobilnych oraz internetowych. W 2014 roku została wykupiona przez Google. Gigant z Mountain View stale rozwija te platformę, dzięki czemu użytkownicy dostają coraz lepsze narzędzie do tworzenia swoich aplikacji. Google umożliwia swoim użytkownikom stworzenie maksymalnie 10 projektów. Każdy z nich na starcie posiada darmowy plan Spark. Jest to podstawowy plan, który jest idealny na początek. Można na nim sprawdzić, czy stworzona aplikacja, będzie przyciągać odbiorców czy też nie.
Gdy zapotrzebowanie na zasoby zacznie rosnąć, może wystąpić potrzeba zakupienia planu Flame, który miesięcznie kosztuje 25 dolarów. Limity możliwe do wykorzystania przez użytkowników w tym planie znacząco zostały podniesione. Dzięki temu aplikacja dalej może się rozwijać, a programiści mogą skupiać się na dodawaniu nowych funkcjonalności.
Ostatni z dostępnych planów to Blaze, który nie posiada stałej kwoty. Osoba będąca właścicielem projektu dokonuje opłaty na podstawie wykorzystanych zasobów w danym okresie rozliczeniowym.
https://firebase.google.com/pricing pod tym linkiem można zobaczyć limity dla pierwszych dwóch planów oraz stawki za dane zasoby dla ostatniego planu. Po utworzeniu projektu menu zostało podzielone na cztery części: Programowanie, Jakość, Analiza oraz Rozwój. Dzięki temu całą funkcjonalność mamy rozmieszczoną w dwóch kategoriach deweloperskich (Programowanie, Jakość) oraz dwóch kategoriach marketingowych (Analiza i Rozwój). Poniżej przedstawię pokrótce, to co możemy znaleźć w każdej z kategorii.

Funkcje wykorzystane w aplikacji:
Authentication – moduł ten odpowiada za określenie sposobów autoryzacji użytkowników. Do dyspozycji mamy logowanie poprzez adres e-mail, Facebooka, Twittera oraz wiele innych opcji.
Database – odpowiada za przechowywanie wprowadzanych przez użytkowników danych, np. dane potrzebne do wysyłki zamówionego przedmiotu. Firebase udostępnia dwa typy baz danych: Realtime Database oraz Cloud Firestore.
Storage – zadaniem tego modułu jest przechowywanie plików. Użytkownik może za pomocą aplikacji przesyłać różne pliki do projektu Firebase, np. pliki z rozszerzeniem pdf.

2.3.5. Github

GitHub – hostingowy serwis internetowy przeznaczony do projektów programistycznych wykorzystujących system kontroli wersji Git. Stworzony został przy wykorzystaniu frameworka Ruby on Rails i języka Erlang. Serwis działa od kwietnia 2008 roku. GitHub udostępnia darmowy hosting programów open source i prywatnych repozytoriów (część funkcji w ramach prywatnych repozytoriów jest płatna). W czerwcu 2018 ogłoszono, iż serwis zostanie przejęty przez przedsiębiorstwo Microsoft za kwotę 7,5 miliarda dolarów. W maju 2019 roku GitHub informuje, że ma około 37 milionów użytkowników i więcej niż 100 milionów repozytoriów (w tym co najmniej 28 milionów repozytoriów publicznych).

Kilka z funkcji, które występują w systemie GitHub:
bugtracker,
forki repozytoriów, czyli kopia w osobnym repozytorium należąca do innego użytkownika,
pull requesty – osoba mająca forka może zgłosić swój kod do złączenia z głównym repozytorium,
statystyki,
organizacje zrzeszające programistów pracujących nad repozytoriami,
web hooks – wywołanie operacji na repozytorium wysyła informację do innego serwisu lub skryptu,
wiki dla celów dokumentacji.

3. Projektowanie

Aplikacja wymagała przygotowania narzędzi do rozpoczęcia programowania, rozpoczęliśmy od instalacji IDE Android Studio, który umożliwa wygodne tworzenie layoutów każdej aktywności oraz kodowanie w różnych językach programowania. Na potrzeby projektowe wybraliśmy język JAVA, również jednym z ważniejszych narzędzi które wykorzystywujemy jest GitHub - hosting umożliwiający tworzenie zdalnego repozytorium oraz kontrolę wersji oprogramowania, dzięki temu możliwa jest współpraca z członkami projektu. Wykorzystujemy także Firebase - zestaw usług hostingowych dla każdego typu aplikacji. Oferuje NoSQL i hosting w czasie rzeczywistym baz danych, treści, uwierzytelnianie społecznościowe i powiadomienia lub usługi, takie jak serwer komunikacji w czasie rzeczywistym. Jako bazę danych wykorzystujemy Firebase Firestore Database. Do rejestracji i logowania używamy Firebase Authentication.

com.google.firebase.auth.FirebaseAuth - biblioteka umożliwajaca połączenie z systemem FireBase Auth, odpowiada za cały system logowania oraz rejestrowania nowych użytkoników. Rozwijając myśl tej biblioteki, pobieramy z formularzy id zmiennych(e-mail, hasło) oraz inicjalizujemy bazę danych. Strona logowania (pierwsza po uruchomieniu aplikacji) wpierw sprawdza czy użytkownik jest zalogowany, jeżeli jest to przekierowuje go na stronę główna aplikacji, jeżli nie to zostaje na stronie do logowania, gdzie po wpisaniu danych sprawdza je z danymi serwera i loguje się gdy są poprawne dane poprzez przcisk "zaloguj", jeżli dane sa nie poprawne odpowiada błędem. przycisk "Zarejestruj przekierowuje do aktywności w której zawarty jest formularz rejestracji, również podłączony pod FirebaseAuth, jeżeli rejestracja przebiegnie pomyślnie, przekierowuje do strony logowania i wyświetla komunikat o poprawnej rejestracji, jeżeli nie to wyświetla komunikat o niepoprawnej rejestracji.

Kontunując, jeżeli logowanie przejdzie pomyślnie zanim uzyskamy dostęp do strony głównej aplikacja poprosi nas o zweryfikowanie właściciela poprzez dane biometryczne, dzięki bibilotece "androidx.biometric:biometric:1.1.0".

com.google.firebase.firestore.FirebaseFirestore - biblioteka umożliwiająca obsługę bazy danych znajdującej się w Firebase, odpowiada za system przechowywania, odczytu i zapisu danych poszczególnych użytkowników. Cloud Firestore jest elastyczną, skalowalną, hierarchiczną bazą danych NoSQL w chmurze służącą do przechowywania i synchronizowania w czasie rzeczywistym danych między klientami i serwerem. Wy-

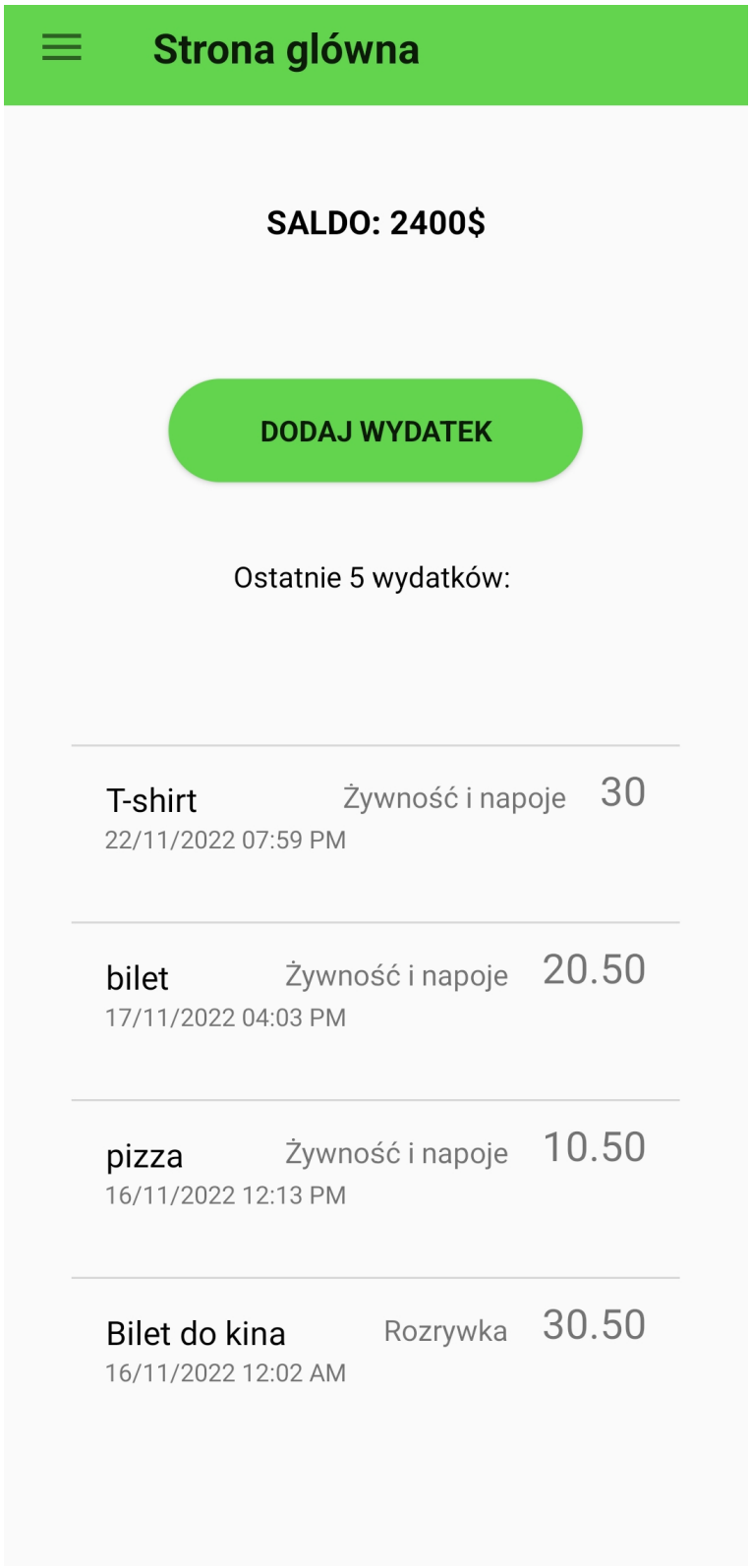
korzystujemy ją jako system do zapisywania wydatków w bazie danych dla poszczególnych użytkowników oraz do wyświetlania ich.
com.google.firebase.firestore.Query - biblioteka, którą wykorzystujemy do obsługi zapytań.
FirebaseUI ułatwia powiązanie danych z Cloud Firestore z interfejsem aplikacji.
com.google.firebase.firestore.DocumentReference - biblioteka daje nam możliwość odwoływania się do poszczególnych dokumentów w bazie danych.
java.util.Calendar- biblioteka, którą wykorzystujemy do pobierania systemowego czasu oraz daty dodania wydaatku.

android.provider.MediaStore - biblioteka odpowiadająca za tworzenie indeksowanej kolekcji zdjęć,wykorzystujemy ją do zapisywania zdjęć po ich zrobieniu.
android.graphics.Bitmap - biblioteka, którą używamy do kompresji zdjęć do formatu bitmapy.

Frontend - XAML
Wygląd naszej aplikacji jest obsługiwany przez język XAML, dzięki niemu jesteśmy w stanie ustytyuwać różne elementy aplikacji w wybranym przez nas miejscu. Oczywiście każdy element możemy edytować w wybrany przez nas sposób, odpowiadają za to takie opcje jak np. width, height, background, textColor, textStyle, textSize, paddingi oraz marginesy, również dla każdego elementu przypisujemy id.

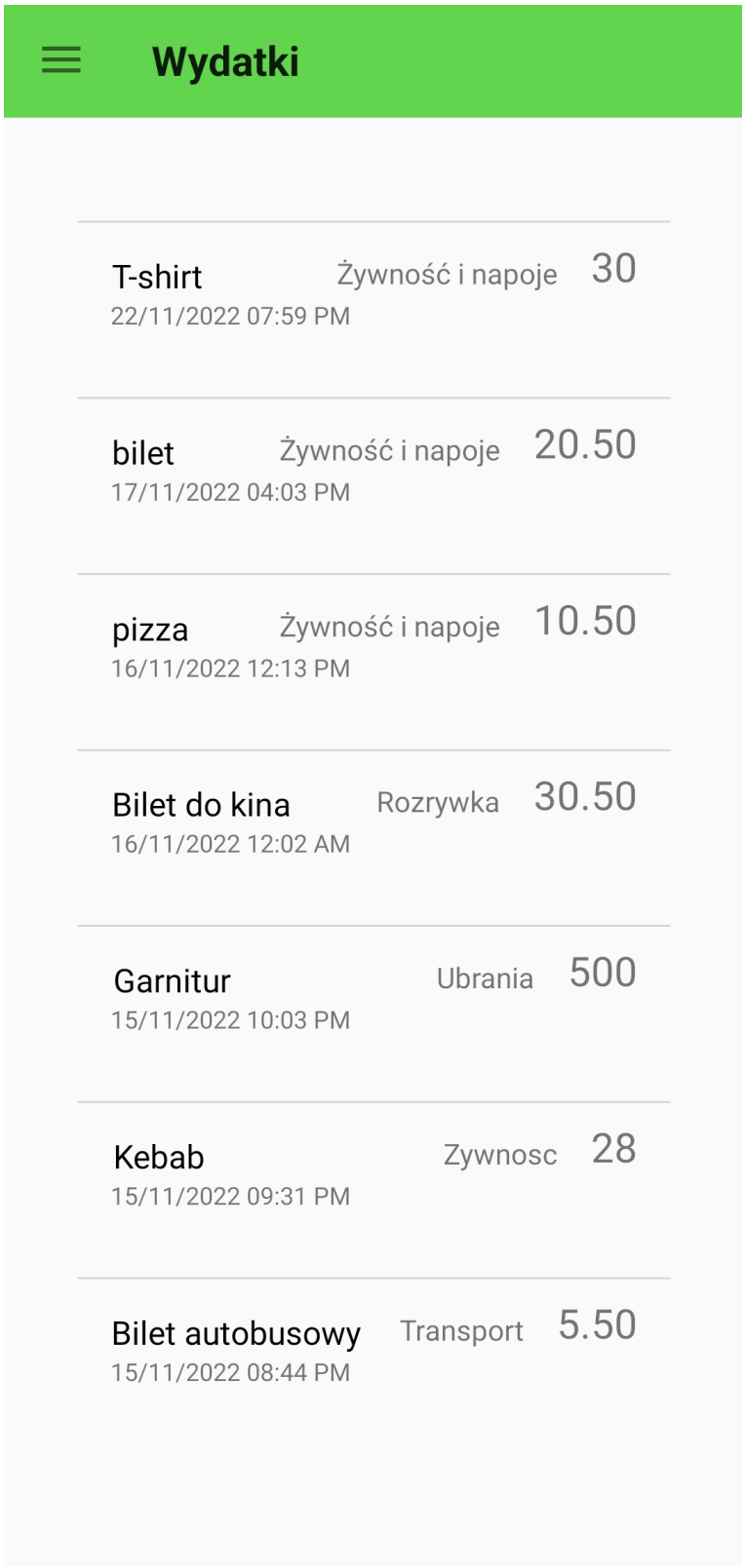
Elementy, które wykorzystujemy:
Button - wywołuje przycisk.
ImageView - tutaj wyświetlamy obraz.
EditText - tworzy pole do wpisania tekstu.
TextView - wyświetla tekst.
Spinner - lista rozwijana
androidx.recyclerview.widget.RecyclerView - specjalna lista do wyświetlania wydatków z bazy danych.
androidx.appcompat.widget.Toolbar - górna belka.
com.google.android.material.navigation.NavigationView - nawigacja aplikacji.
LinearLayout - kolekcja która uporządkowuje elementy w cały layout.

Rysunek 3.1 (s. 11) przedstawia stronę główną aplikacji po zalogowaniu i potwierdzeniu danych biometrycznych. Górna część jest odpowiedzialna za nawigację po aplikacji, poniżej znajduję sie saldo naszego konta, przycisk dodaj wydatek oraz



Rys. 3.1. Strona główna aplikacji

ostatnie 5 wydatków. Rysunek 3.2 (s. 13) przedstawia aktywność, gdzie pobrane są wszystkie wydatki użytkownika.



Rys. 3.2. Strona podglądu wszystkich wydatków

4. Implementacja

4.1. Firebase Auth

RegisterPage:

```
1  regBtn.setOnClickListener(new View.OnClickListener() {
2              @Override
3              public void onClick(View view) {
4                  String email = regMail.getText().toString().trim();
5                  String password = regPassword.getText().toString().
       trim();
6                  String login = regLogin.getText().toString();
7                  if(TextUtils.isEmpty(email)){
8                      regMail.setError("Email id Required");
9                      return;
10                 }
11
12                 if (TextUtils.isEmpty(password)) {
13                     regPassword.setError("Password is Required");
14                     return;
15                 }
16
17                 if (password.length() < 6) {
18                     regPassword.setError("Password must be >=6
       characters");
19                     return;
20                 }
```

Listing 1. Firebase Auth naciśnięcie przycisku rejestracji

Listing 1 (s. 14) to metoda, po naciśnięciu przycisku rejestracji, tworzy zmienne email, login i hasło, a następnie sprawdza czy pole email nie jest puste, czy pole hasło nie jest puste oraz czy hasło zawiera przynajmniej 6 znaków. Linijka 1 ustawia przycisk w tryb nasłuchiwania czy przycisk jest kliknięty, linijka 4 i 5 przypisuje text wpisnay w formularzu do odpowiadających ich stringom, linijka 7 sprawdza czy został wprowadzony jakiś tekst - tak samo w linii 12, a w linii 17 sprawdzamy długość hasła.

Metoda Firebase Auth w listingu (2 (s. 15)) to metoda tworząca użytkownika za pomocą emaila i hasła. Funckja onComplete w linii 3 sprawdza czy rejestarcja się powiodła. Jeśli tak, to w linii 6 z Firebase Auth pobrane zostaje ID użytkownika i zapisane jako zmienna String. Następnie w linii 8 zostaje utworzona Hashmapa do której zapisane zostają login oraz email pobrany z formularza rejestarcji. Hashmapa

w linii 11 zostaje dodana do bazy danych do kolekcji "users"w dokumencie o id poszczególnego użytkownika. Następnie przenosi do strony głównej aplikacji.

```
1  fAuth.createUserWithEmailAndPassword(email, password).
    addOnCompleteListener(new OnCompleteListener<AuthResult>() {
2    @Override
3     public void onComplete(@NonNull Task<AuthResult> task) {
4     if (task.isSuccessful()) {
5     Toast.makeText(registerPage.this, "Utworzono uzytkownika.",
   Toast.LENGTH_SHORT).show();
6     userID = fAuth.getCurrentUser().getUid();
7    DocumentReference documentReference = fStore.collection("users"
   ).document(userID);
8       Map<String,Object> user = new HashMap<>();
9    user.put("login",login);
10    user.put("email",email);
11    documentReference.set(user).addOnSuccessListener(new
   OnSuccessListener<Void>() {
12    @Override
13   public void onSuccess(Void unused) {
14    Log.d(TAG,"onSuccess: Konto zostalo utworzone dla "+ userID);
   } });
15   startActivity(new Intent(getApplicationContext(),startPage.
   class));
16   }else {\\
17   Toast.makeText(registerPage.this, "Blad! " + task.getException
   ().getMessage(), Toast.LENGTH_SHORT).show();
18   } } });\\\\
19
```

Listing 2. Firebase Auth - Rejestracja

```
1    if (fAuth.getCurrentUser() != null) {
2            startActivity(new Intent(getApplicationContext(),
   startPage.class));
3            finish();
4        }
5
```

Listing 3. Sprawdza zalogowanie

Instrukcja warunkowa, opisana 3 (s. 15) w linii 1 sprawdza czy jakiś użytkownik nie jest już zalogowany. Jeśli użytkownik jest zalogowany to w linii 2 przenosi go do strony głównej.

```
1 RegisterBtn.setOnClickListener(new View.OnClickListener() {
2                 @Override
3                 public void onClick(View view) {
4                     startActivity(new Intent(loginPage.this,
    registerPage.class));
5                 }
6             });
```

Listing 4. Listner przycisku register

Listing 4 (s. 16) nasłuchuje, czy zostanie naciśnięty przycisk rejestracji, a jeśli zostanie przenosi użytkownika na stronę z formularzem rejestracji, dzięki linii 4.

```
1  LoginBtn.setOnClickListener(new View.OnClickListener() {
2    @Override
3    public void onClick(View view) {
4
5        String email = Login.getText().toString().trim();
6        String password = Password.getText().toString().trim();
7
8        if(email.isEmpty() || password.isEmpty()){
9            Toast.makeText(loginPage.this, "Wprowadz dane logowania
    !", Toast.LENGTH_SHORT).show();
10           startActivity(new Intent(loginPage.this, loginPage.
    class));
11       }
12
13
14       fAuth.signInWithEmailAndPassword(email,password).
    addOnCompleteListener(new OnCompleteListener<AuthResult>() {
15           @Override
16           public void onComplete(@NonNull Task<AuthResult> task)
    {
17               if(task.isSuccessful()){
18                   Toast.makeText(loginPage.this, "Zalogowano",
    Toast.LENGTH_SHORT).show();
19                   startActivity(new Intent(getApplicationContext
    (),fpPage.class));
20               }else {
21                   Toast.makeText(loginPage.this, "Blad logowania!
     " + task.getException().getMessage(), Toast.LENGTH_SHORT).show
    ();
22               }
23           }
24       });
```

```
25      }
26  });
```

Listing 5. System logowania

Listing 5 (s. 16) po naciśnięciu przycisku logowania metoda w linii 5 i 6 zapisuje ciągi znaków email i hasło, wpisanych do odpowiednich pól i sprawdza czy jakieś pole nie zostało puste - jeśli tak,wyświetla odpowiedni komunikat. Jeśli email oraz hasło zostały wpisane, następuje sprawdzenie w Firebase czy w usłudze istnieje użytkownik o takim emailu i haśle - jeśli tak, wyświetla komunikat o zalogowaniu oraz przenosi użytkownika do strony z weryfikacją biometryczną. Jeśli nie znajdzie użytkownika o takich danych, wyświetli komunikat o błędzie logowania.

4.2. Baza danych

Rysunek 4.1 (s. 18) przedstawia wygląd bazy danych, pierwsza kolumna jest to kolekcja "users", następnie przechodzimy z niej do dokumentu - które noszą nazwę ID użytkownika. Następnie mamy wybór kolekcji, tutaj umieściliśmy "wydatki", ponieżej również widać dane użytkownika - jego email oraz login.

Rysunek 4.2 (s. 18) to ciąg dalszy bazy danych, zatem po kliknięciu w kolekcję "wydatki"ukazują nam się dokumenty, które również jak użytkownicy nosza nazwę swojego ID - po kliknięciu w poszczególny dokument otrzymujemy dane wydatku, takie jak: data, kategoria, kwota oraz nazwa.

4.3. Tryb ciemny

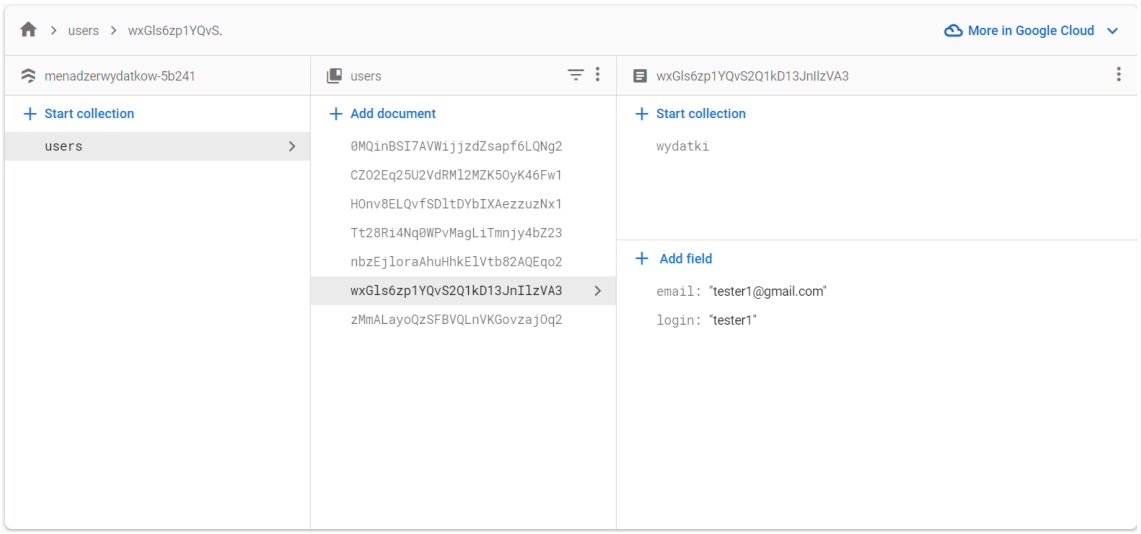
Listing 6 (s. 17) to ustawiania dotyczące trybu ciemnego, linia pierwsza będzie odpowiadać za kolor tekstu na naszych stronach a linia druga za kolor tła - wpierw trzeba utworzyć plik w folderze values, który będzie odpowiadał za kolory, plik do trybu jasne już został wygenerowany automatyczne, zatem tworzymy plik /values/night/colors.xml. Dopisujemy w pliku sekcję, która będzie odpowiadała za kolor:

```
1  <color name="textColor">#000000</color>
2  <color name="backgroundColor">#ffffff</color>
3
```

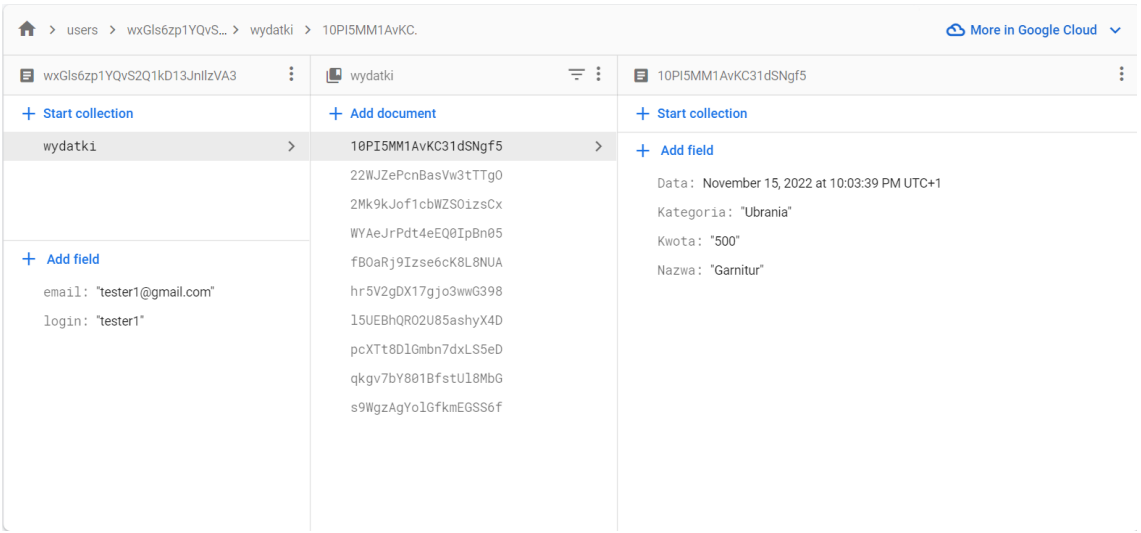
Listing 6. night/colors.xml

W listingu 7 (s. 17) w pliku xml danej aktywności do elementu odnosimy sie do wyżej określonych kolorów .

```
1  android:background="@color/backgroundColor"
```



Rys. 4.1. Wygląd bazy danych.



Rys. 4.2. Wygląd bazy danych cd.

```
2
```

Listing 7. Xml DarkMode

Listing 8 (s. 19) pobiera przycisk w linii 1 oraz obsługuje go, w linii 3 do zmiennej isDarkModeOn pobiera status przez funkcję getDarkModeStatus(), linia od 4 do 8 to insturkcja warunkowa, jeżeli DarkMode jest aktywny to zmieniamy nazwę przycisku oraz odwrotnie w zależności od trybu. Funkcja onClick w linii 12 zawiera ustawienie tego trybu dzięki "AppCompatDelegate.MODE NIGHT YES/NO"oraz przypisuje true lub false do pomocniczej zmiennej isDarkModeOn.

```
1  btnlightdark = findViewById(R.id.btndarklight);
2
3  isDarkModeOn = getDarkModeStatus();
4  if(isDarkModeOn){
5      btnlightdark.setText("On Light Mode");
6  }else {
7      btnlightdark.setText("On Dark Mode");
8  }
9
10 btnlightdark.setOnClickListener(new View.OnClickListener() {
11     @Override
12     public void onClick(View view) {
13         if(isDarkModeOn){
14             AppCompatDelegate.setDefaultNightMode(AppCompatDelegate
       .MODE_NIGHT_NO);
15             btnlightdark.setText("On Dark Mode");
16             isDarkModeOn=false;
17         }else {
18             AppCompatDelegate.setDefaultNightMode(AppCompatDelegate
       .MODE_NIGHT_YES);
19             btnlightdark.setText("On Light Mode");
20             isDarkModeOn=true;
21         }
22     }
23 });
```

Listing 8. DarkMode Java Button obsługa

Listing 9 (s. 19) odpowiada za sprawdzanie trybu aplikacji w której jesteśmy aktualnie dzięki linii 2 gdzie pobieramy konguguracje flagi, oraz zwracamy w liniach 6, 8 i 10 true lub false co odpowiada trybom.

```
1  private boolean getDarkModeStatus(){
2    int nightModeFlags =
3            ustawieniaPage.this.getResources().getConfiguration().
     uiMode & Configuration.UI_MODE_NIGHT_MASK;
```

```
  switch (nightModeFlags){
      case Configuration.UI_MODE_NIGHT_YES:
          return true;
      case Configuration.UI_MODE_NIGHT_NO:
          return false;
      case Configuration.UI_MODE_NIGHT_UNDEFINED:
          return false;
  }
  return false;
}
```

Listing 9. DarkMode sprawdzanie motywu

4.4. Nawigacja

Tworzenie menu nawigacji w listingu 10 (s. 20) – Najpierw w layoucie strony trzeba zaimplementować widget, który umożliwia wysuwanie się szuflad z krawędzi okna:

```
  <androidx.drawerlayout.widget.DrawerLayout>
```

Listing 10. Tworzenie menu nawigacji

Następnie w listingu 11 (s. 20) dodajemy bibliotekę NavigationView aby można było zaimplementować zawartość wszystkich typów szuflad nawigacyjnych:

```
  <com.google.android.material.navigation.NavigationView>
```

Listing 11. Tworzenie NavigationView

Listing 12 (s. 20) odpowiada za wygląd naszego menu wysuwanego, do itemów przypiasane są ikony, tytuł oraz id.

```
  <menu xmlns:android="http://schemas.android.com/apk/res/android">

    <group android:checkableBehavior="single">
  <item
                android:id="@+id/nav_start"
                  android:icon="@drawable/ic_start"
                  android:title="Strona glowna" />
  <item
              android:id="@+id/nav_wydatki"
                  android:icon="@drawable/ic_wydatki"
                  android:title="Wydatki" />
               <item
                  android:id="@+id/nav_statystyki"
                  android:icon="@drawable/ic_statystyki"
                  android:title="Statystyki" />
```

```
16      <item
17                android:id="@+id/nav_ustawienia"
18                  android:icon="@drawable/ic_ustawienia"
19                  android:title="Ustawienia" />
20    </group>
21
22 </menu>
```

Listing 12. menu/main menu.xml

W listingu 13 (s. 21) obsługujemy wysuwanie i chowanie się całej nawigacji.

```
1   navigationView.bringToFront();
2 ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(this,
    drawerLayout, toolbar, R.string.navigation_drawer_open, R.string
    .navigation_drawer_close);
3 drawerLayout.addDrawerListener(toggle);
4 toggle.syncState();
```

Listing 13. Wysuwanie się nawigacji

Listing 14 (s. 21) odpowiada za wywoływanie wybranej pozycji w menu nawigacji:

```
1   navigationView.setNavigationItemSelectedListener(this);
```

Listing 14. Wywoływanie pozycji nawigacja

Listing 15 (s. 21) tworzy metodę odpowiedzialną za poruszanie się między stronami. W linii 2 onNavigationItemSelected odpowiada za to, że metoda zostaję wywołana po wybraniu elementu z menu nawigacji. Intent w linii np. 7 służy do przechodzenia między aktywnościami, startActivity w linni np. 8 uruchamia daną aktywność.

(Przykład na podstawię poruszania się ze strony startowej)

```
1   Override
2 public boolean onNavigationItemSelected(@NonNull MenuItem menuItem)
    {
3
4     switch (menuItem.getItemId()) {
5
6         case R.id.nav_wydatki:
7             Intent a = new Intent(startPage.this, wydatkiPage.class
    );
8             startActivity(a);
9             break;
10
11        case R.id.nav_statystyki:
```

```
12              Intent b = new Intent(startPage.this, statystykiPage.
    class);
13              startActivity(b);
14              break;
15
16          case R.id.nav_ustawienia:
17              Intent c = new Intent(startPage.this, ustawieniaPage.
    class);
18              startActivity(c);
19              break;
20
21          case R.id.nav_start:
22              Intent d = new Intent(startPage.this, startPage.class);
23              startActivity(d);
24              break;
25      }
26
27      drawerLayout.closeDrawer(GravityCompat.START);
28
29      return true;
30  }
```

Listing 15. Tworzenie poruszanie się między stronami

4.5. Dane biometryczne

Pierwszą najważniejszą rzeczą, którą należy wykonać do obsługi danych biometrycznych jest zaimplementowanie biblioteki androida, która obsługuję system weryfikacji biometrycznej. Implemetnacja nastąpuję w pliku app/build.gradle do której dopisujemy następującą linijkę kodu zawartą w listingu 16 (s. 22):

```
1  implementation "androidx.biometric:biometric:1.1.0"
```

Listing 16. Implementacja bibiblioteki biometric

Następnie w naszym projkecie utworzyliśmy nową aktywność, która będzie odpowiadała za weryfikację biometryczną - nazwaliśmy ja fpPage. Na ekranie layoutu dodaliśmy przycisk odpowiadający za wywołanie weryfikacji - przycisk ikoniczny widniejący na obrazie 4.3 (s. 23).

Listing 17 (s. 23) jest to zawartość pliku fppage.java, który odpowiada za obsługe danych biometrycznych. Linijka 1 pobiera obrazek zaimplementowany wyżej do zmiennej imageViewLogin, następnie wywołujemy BiometricManager'a, oraz od llinijki 4 do 14 dzięki switchach i casach sprawdzamy czy operacje biometryczne powiodły się, efekt zwracamy w logach, lub w powiadomieniu w Toast'cie. W linii 23

Rys. 4.3. Ikoniczny przycisk biometryki

wywyołujemy okienko autoryzacji, które opisane sa w linii od 52 do 56 gdzie ustawiamy tytuł, podtytuł oraz przycisk, po udanej autoryzacji w linii 38 przenosimy się do naszej głównej strony aplikacji czyli startPage.class

```
1  imageViewLogin=findViewById(R.id.imageView);
2
3  BiometricManager biometricManager = BiometricManager.from(this);
4  switch (biometricManager.canAuthenticate(BIOMETRIC_STRONG |
    DEVICE_CREDENTIAL)) {
5      case BiometricManager.BIOMETRIC_SUCCESS:
6          Log.d("MY_APP_TAG", "Aplikacja uzywa danych
    biometrycznych.");
7          break;
8      case BiometricManager.BIOMETRIC_ERROR_NO_HARDWARE:
9          Toast.makeText(this,"Sensor niedostepny",Toast.LENGTH
    LONG);
10         break;
11     case BiometricManager.BIOMETRIC_ERROR_HW_UNAVAILABLE:
12         Log.e("MY_APP_TAG", "Dane biometryczne niedostepne.");
13         break;
14     case BiometricManager.BIOMETRIC_ERROR_NONE_ENROLLED:
15         // Prompts the user to create credentials that your app
    accepts.
16         final Intent enrollIntent = new Intent(Settings.
    ACTION_BIOMETRIC_ENROLL);
17         enrollIntent.putExtra(Settings.
    EXTRA_BIOMETRIC_AUTHENTICATORS_ALLOWED,
18                 BIOMETRIC_STRONG | DEVICE_CREDENTIAL);
19         startActivityForResult(enrollIntent, REQUEST_CODE);
20         break;
21 }
```

```
22  executor = ContextCompat.getMainExecutor(this);
23  biometricPrompt = new BiometricPrompt(fpPage.this,
24          executor, new BiometricPrompt.AuthenticationCallback() {
25      @Override
26      public void onAuthenticationError(int errorCode,
27                                        @NonNull CharSequence
     errString) {
28          super.onAuthenticationError(errorCode, errString);
29          Toast.makeText(getApplicationContext(),
30                  "Blad autoryzacji: " + errString, Toast.
     LENGTH_SHORT)
31                  .show();
32      }
33
34      @Override
35      public void onAuthenticationSucceeded(
36              @NonNull BiometricPrompt.AuthenticationResult result)
     {
37          super.onAuthenticationSucceeded(result);
38          startActivity(new Intent(fpPage.this,startPage.class));
39          Toast.makeText(getApplicationContext(),
40                  "Autoryzacja powiodla sie!", Toast.LENGTH_SHORT).
     show();
41      }
42
43      @Override
44      public void onAuthenticationFailed() {
45          super.onAuthenticationFailed();
46          Toast.makeText(getApplicationContext(), "Autoryzacja
     niepowiodla sie!",
47                  Toast.LENGTH_SHORT)
48                  .show();
49      }
50  });
51
52  promptInfo = new BiometricPrompt.PromptInfo.Builder()
53          .setTitle("Biometryczne odblokowywanie")
54          .setSubtitle("Odblokuj uzywajac danych biometrycznych")
55          .setNegativeButtonText("Uzyj hasla")
56          .build();
57
58  imageViewLogin.setOnClickListener(view -> {
59      biometricPrompt.authenticate(promptInfo);
```

```
60  });
```

Listing 17. Biometryka w pliku fppage.java

4.6. Text Spinner - wysuwane pole wyboru

Text Spinner czyli wysuwane pole wyboru, jest dosyć oczywsitą rzeczą w aplikacji mobilnej, tym bardziej w mendadżerze wydatków, akurat w nim zdecydowaliśmy się umieścić kategorię naszych wydatków na (Żywność i napoje/Odzież,etc.). Jest to dosyć istotna opcja w naszej aplikacji, ponieważ dzięki niemu nie będzie problemów z sortowaniem wydatków w bazie danych. Listing 18 (s. 25) jest to kod utworzenia właśnie tego "sztywnego"pola wyborów wydatków. Jak widać więszkość kodu przedstawia zwykłą personalizację elementu, aczkolwiek najwazniejsze znajduje sie w linii 8 w niej przypisujemy w nasze sztywne opcje wyboru.

```
1  <Spinner
2            android:id="@+id/kategoriaspinner"
3            android:layout_width="match_parent"
4            android:layout_height="wrap_content"
5            android:backgroundTint="#9C9C9C"
6            android:ems="10"
7            android:inputType="textPersonName" />
8            android:entries="@array/kategorie"
9            android:paddingTop="15dp"
10           android:paddingBottom="15dp" />
```

Listing 18. TextSpinner XAML

Elementy które znajdują się w spinnerze są określane właśnie w listingu 18 (s. 25) w pliku app/src/main/res/values/array.xml, oraz wywyłujemy je w XAML'u dzięki nazwie string array'a, którą przypisujemy w linijce 2,a pozostałe w znacznikach "item"to nasze pola wyboru.

```
1  <Spinner
2  <string-array name="kategorie">
3  <item>Zywnosc i napoje</item>
4  <item>Kredyty</item>
5  <item>Mieszkanie i oplaty</item>
6  <item>Transport i komunikacja</item>
7  <item>Chemia domowa</item>
8  <item>Zdrowie i uroda</item>
9  <item>Hobby i rozrywka</item>
10 <item>Urlop i wyjazdy</item>
11 <item>Odziez</item>
```

```
12   <item>Inne</item>
13 </string-array>
```

Listing 19. Array kategorie

5. Testowanie

6. Podręcznik użytkownika

Spis rysunków

Spis tabel

Spis listingów