

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	3
2. Określenie wymagań szczegółowych	4
2.1. Schemat	4
3. Projektowanie	6
4. Implementacja	11
4.1. Firebase Auth	11
4.2. Baza danych	14
4.3. Tryb ciemny	14
4.4. Nawigacja	17
4.5. Dane biometryczne	19
4.6. Text Spinner - wysuwane pole wyboru	21
5. Testowanie	23
6. Podręcznik użytkownika	24
Literatura	25
Spis rysunków	25
Spis tabel	26
Spis listingów	27

1. Ogólne określenie wymagań

Aplikacja ma za zadanie podumowywać wydatki użytkownika. Klient będzie mógł wprowadzać dane dotyczące danego wydatku oraz podporządkować je kategorii (np. spożywcze/transport/podatki). Ma istnieć 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

Aplikacja będzie pisana w środowisku Android Studio językiem 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 włączeniu 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 formularza dodawania wydatku. Użytkownik będzie podawał tytuł wydatku, kwotę 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 statystyki. 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 użytkowników przy użyciu internetu.

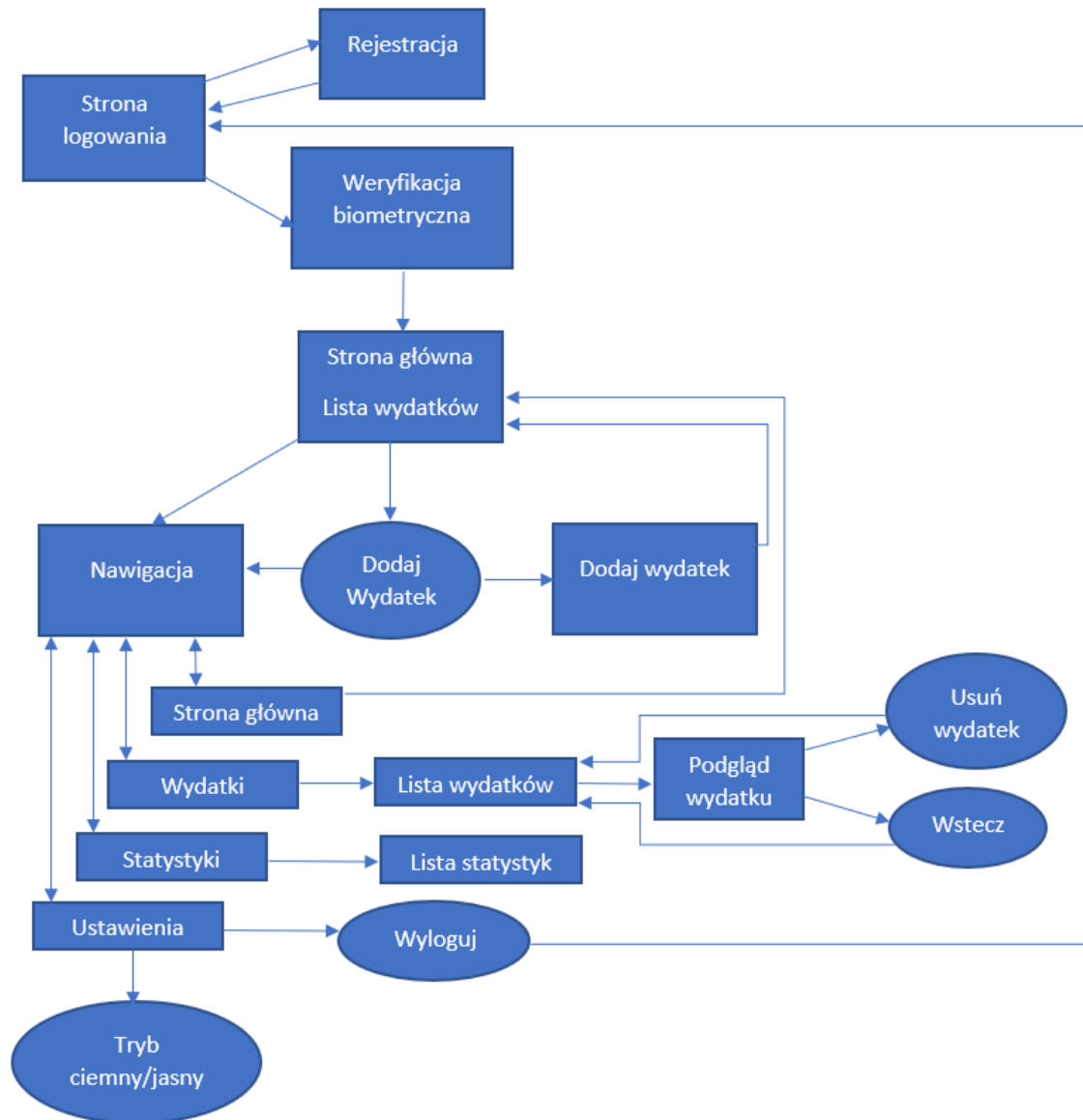
Podkategoria statystyki: suma wydatków, dochód, suma wydatków podporzątkowana kategoriom.

Podkategoria wydatki: wszystkie wydatki, możliwość usunięcia dowolnych wydatków.

Podkategoria ustawienia: przycisk switch dzięki któremy będziemy mieć możliwość przełączenia pomiędzy trybem jasnym i ciemnym aplikacji, dane użytkownika oraz przycisk wyloguj.

2.1. 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 przycisk "Dodaj wydatek", dzięki niemu można przejść do aktywności która umożliwia nam tą funkcję, po dodaniu wydatku przenosimy się na



Rys. 2.1. Schemat aplikacji

stronę główną. Nawigacja umożliwia nam poruszanie się po aktywnościach aplikacji, przejście do "wydatki" przenosi nas do listy wydatków, a po kliknięciu w wydatek przenosimy się 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 przycisk "Wyloguj", który wylogowuje nas z aplikacji i przenosi do strony logowania.

3. Projektowanie

Aplikacja wymagała przygotowania narzędzi do rozpoczęcia programowania, rozpoczęliśmy od instalacji IDE Android Studio, który umożliwia wygodne tworzenie layoutów każdej aktywności oraz kodowanie w różnych językach programowania. Na potrzeby projektowe wybraliśmy język JAVA, również jednym z ważniejszych narzędzi które wykorzystujemy 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 umożliwiająca połączenie z systemem Firebase Auth, odpowiada za cały system logowania oraz rejestrowania nowych użytkowników. Rozwijając myśl tej biblioteki, pobieramy z formularzy id zmiennych (e-mail, hasło) oraz inicjalizujemy bazę danych. Strona logowania (pierwsza po uruchomieniu aplikacji) w pierwszej kolejności sprawdza czy użytkownik jest zalogowany, jeżeli jest to przekierowuje go na stronę główną aplikacji, jeżeli nie to zostaje na stronie do logowania, gdzie po wpisaniu danych sprawdza je z danymi serwera i loguje się gdy są poprawne dane poprzez przycisk "zaloguj", jeżeli dane są 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.

Kontynuując, jeżeli logowanie przejdzie pomyślnie zanim uzyskamy dostęp do strony głównej aplikacja poprosi nas o zweryfikowanie właściciela poprzez dane biometryczne, dzięki bibliotece "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.

FirestoreUI 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 wydatku.

`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 ustytytuwać 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

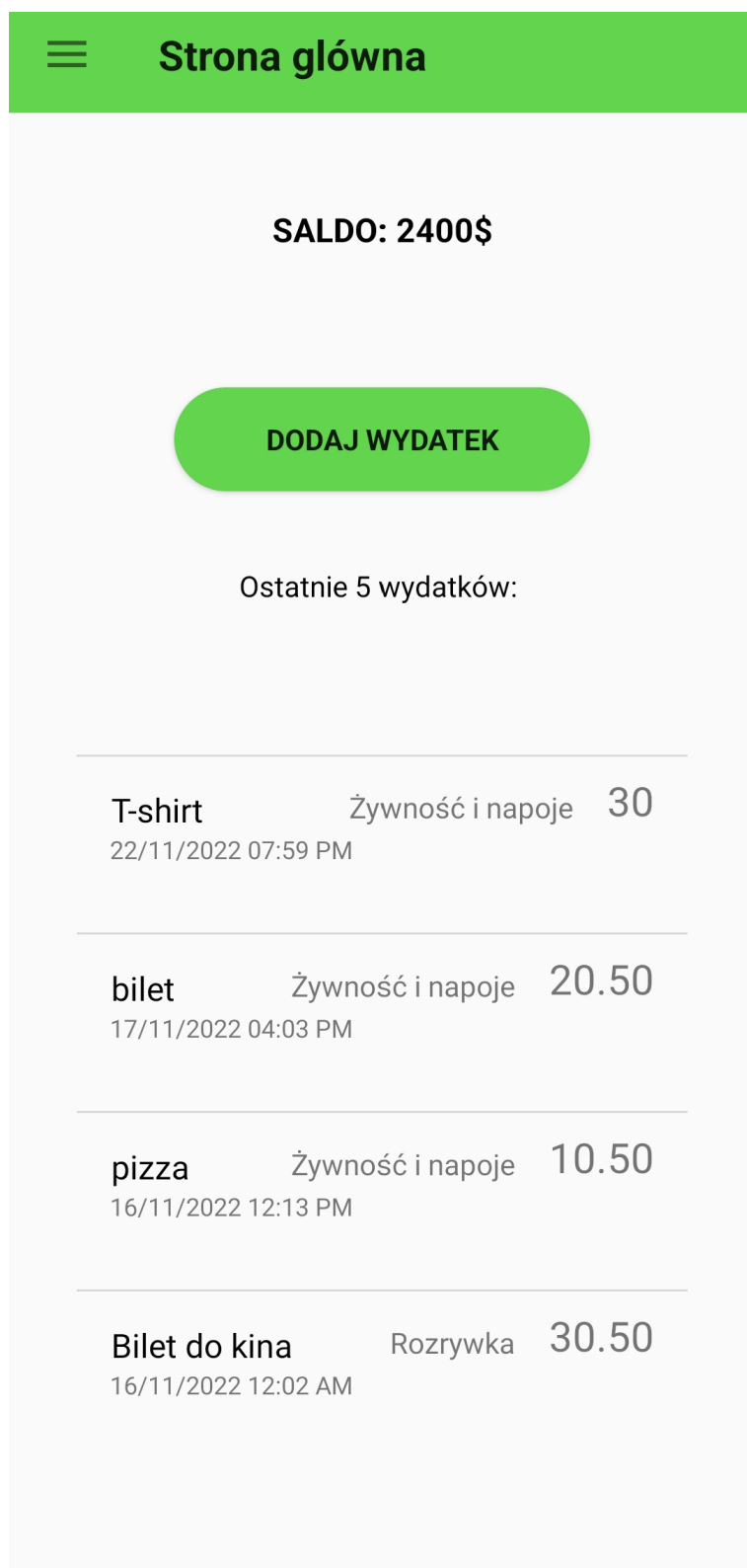
`androidx.recyclerview.widget.RecyclerView` - specjalna lista do wyświetlania wydatków z bazy danych.

`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. 8) przedstawia stronę główną aplikacji po zalogowaniu i potwierdzeniu danych biometrycznych. Górna część jest odpowiedzialna za nawigację po aplikacji, poniżej znajdują się saldo naszego konta, przycisk dodaj wydatek oraz



Rys. 3.1. Strona główna aplikacji

ostatnie 5 wydatków. Rysunek 3.2 (s. 10) przedstawia aktywność, gdzie pobrane są wszystkie wydatki użytkownika.

Wydatki			
T-shirt	Żywność i napoje	30	22/11/2022 07:59 PM
bilet	Żywność i napoje	20.50	17/11/2022 04:03 PM
pizza	Żywność i napoje	10.50	16/11/2022 12:13 PM
Bilet do kina	Rozrywka	30.50	16/11/2022 12:02 AM
Garnitur	Ubrania	500	15/11/2022 10:03 PM
Kebab	Zywnosc	28	15/11/2022 09:31 PM
Bilet autobusowy	Transport	5.50	15/11/2022 08:44 PM

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().
6         trim();
7         String login = regLogin.getText().toString();
8         if(TextUtils.isEmpty(email)){
9             regMail.setError("Email is Required");
10            return;
11        }
12        if (TextUtils.isEmpty(password)) {
13            regPassword.setError("Password is Required");
14            return;
15        }
16        if (password.length() < 6) {
17            regPassword.setError("Password must be >=6
18            characters");
19            return;
20        }
```

Listing 1. Firebase Auth naciśnięcie przycisku rejestracji

Kod 1 (s. 11) 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.

Metoda Firebase Auth (2 (s. 11)) tworząca użytkownika za pomocą emaila i hasła. Funkcja onComplete sprawdza czy rejestracja się powiodła. Jeśli tak, z Firebase Auth pobrane zostaje ID użytkownika i zapisane jako zmienna String. Następnie zostaje utworzona Hashmapa do której zapisane zostają login oraz email pobrany z formularza rejestracji. Hashmapa 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                }
16            });
17            startActivity(new Intent(getApplicationContext(), startPage.
              class));
18        } else {
19            Toast.makeText(registerPage.this, "Blad! " + task.getException
              ().getMessage(), Toast.LENGTH_SHORT).show();
20        }
21    }
22 }

```

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. 12) sprawdza czy jakiś użytkownik nie jest już zalogowany. Jeśli użytkownik jest zalogowany to 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 }

```

```

5         }
6     });

```

Listing 4. Listner przycisku register

Metoda 4 (s. 12) nasłuchuje, czy zostanie naciśnięty przycisk rejestracji, a jeśli zostanie przenosi użytkownika na stronę z formularzem rejestracji.

```

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
10             !", Toast.LENGTH_SHORT).show();
11              startActivity(new Intent(loginPage.this, loginPage.
12             class));
13          }
14
15          fAuth.signInWithEmailAndPassword(email,password).
16          addOnCompleteListener(new OnCompleteListener<AuthResult>() {
17              @Override
18              public void onComplete(@NonNull Task<AuthResult> task)
19              {
20                  if(task.isSuccessful()){
21                      Toast.makeText(loginPage.this, "Zalogowano",
22                      Toast.LENGTH_SHORT).show();
23                      startActivity(new Intent(getApplicationContext()
24                      (),fpPage.class));
25                  }else {
26                      Toast.makeText(loginPage.this, "Bład logowania!
27                      " + task.getException().getMessage(), Toast.LENGTH_SHORT).show
28                      ();
29                  }
30              }
31          });
32      }
33  });

```

Listing 5. System logowania

5 (s. 13) Po naciśnięciu przycisku logowania metoda 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 hasle - 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. 15) 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", poniżej również widać dane użytkownika - jego email oraz login.

Rysunek 4.2 (s. 15) to ciąg dalszy bazy danych, zatem po kliknięciu w kolekcję "wydatki" ukazują nam się dokumenty, które również jak użytkownicy noszą nazwę swojego ID - po kliknięciu w poszczególny dokument otrzymujemy dane wydatku, takie jak: data, kategoria, kwota oraz nazwa.

4.3. Tryb ciemny

6 Listner (s. 14) to tworzenie trybu ciemnego w aplikacji - wpierw trzeba utworzyć plik w folderze values, który będzie odpowiadał za kolory, plik do trybu jasne już został wygenerowany automatycznie, 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

7 W listnerze (s. 14) w pliku xml danej aktywności do elementu odnosimy się do wyżej określonych kolorów .

```
1 android:background="@color/backgroundColor"
2
```

Listing 7. Xml DarkMode

8 Listner (s. 16) pobiera przycisk oraz obsługuje go, do zmiennej isDarkModeOn pobiera status przez funkcję getDarkModeStatus(), jeżeli DarkMode jest aktywny to zmieniamy nazwę przycisku oraz odwrotnie w zależności od trybu. Funkcja onClick

<div> <div> <div>🏠</div> <div>> users > wxGls6zp1YQvS.</div> </div> <div>More in Google Cloud</div> </div>		
<div>menadzerwydatkow-5b241</div> <div> <div>+ Start collection</div> <div>users ></div> </div>	<div>users</div> <div> <div>+ Add document</div> <div> <div>0MQinBSI7AVWijjzdZsapf6LQNg2</div> <div>CZ02Eq25U2VdRM12MZK5OyK46Fw1</div> <div>H0nv8ELQvfSD1tDYbIXAezuzNx1</div> <div>Tt28Ri4Nq0WPvMagLiTmnjy4bZ23</div> <div>nbzEj1oraAhuHhKELVtb82AQEQo2</div> <div>wxGls6zp1YQvS2Q1kD13JnI1zVA3 ></div> <div>zMnALayoQzSFBVQLnVKGovzaj0q2</div> </div> </div>	<div>wxGls6zp1YQvS2Q1kD13JnI1zVA3</div> <div> <div>+ Start collection</div> <div>wydatki</div> <div> <div>+ Add field</div> <div> <div>email: "tester1@gmail.com"</div> <div>login: "tester1"</div> </div> </div> </div>

Rys. 4.1. Wygląd bazy danych.

<div> <div> <div>🏠</div> <div>> users > wxGls6zp1YQvS... > wydatki > 10PI5MM1AvKC.</div> </div> <div>More in Google Cloud</div> </div>		
<div>wxGls6zp1YQvS2Q1kD13JnI1zVA3</div> <div> <div>+ Start collection</div> <div>wydatki ></div> <div> <div>+ Add field</div> <div> <div>email: "tester1@gmail.com"</div> <div>login: "tester1"</div> </div> </div> </div>	<div>wydatki</div> <div> <div>+ Add document</div> <div> <div>10PI5MM1AvKC31dSNGf5 ></div> <div> <div>22WJZePcnBasVw3tTTg0</div> <div>2Mk9kJof1cbWZS01zsCx</div> <div>WYAeJrPdt4eEQ0IpBn05</div> <div>fB0aRj9Izse6cK8L8NUA</div> <div>hr5V2gDX17gjo3wwG398</div> <div>15UEBhQR02U85ashyX4D</div> <div>pcXTt8D1Gmbn7dxLS5eD</div> <div>qkgv7bY801BfstU18MbG</div> <div>s9WgzAgYo1GfkmEGSS6f</div> </div> </div> </div>	<div>10PI5MM1AvKC31dSNGf5</div> <div> <div>+ Start collection</div> <div> <div>+ Add field</div> <div> <div>Data: November 15, 2022 at 10:03:39 PM UTC+1</div> <div>Kategoria: "Ubrania"</div> <div>Kwota: "500"</div> <div>Nazwa: "Garnitur"</div> </div> </div> </div>

Rys. 4.2. Wygląd bazy danych cd.

zawiera ustawienie tego trybu dzięki "AppCompatActivity.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             AppCompatActivity.setDefaultNightMode(AppCompatActivity
15                 .MODE_NIGHT_NO);
16             btnlightdark.setText("On Dark Mode");
17             isDarkModeOn=false;
18         }else {
19             AppCompatActivity.setDefaultNightMode(AppCompatActivity
20                 .MODE_NIGHT_YES);
21             btnlightdark.setText("On Light Mode");
22             isDarkModeOn=true;
23         }
24     }
25 });

```

Listing 8. DarkMode Java Button obsługa

Listner 9 (s. 16) odpowiada za sprawdzanie trybu aplikacji w której jesteśmy aktualnie, oraz zwracamy true lub false.

```

1 private boolean getDarkModeStatus(){
2     int nightModeFlags =
3         ustawieniaPage.this.getResources().getConfiguration().
4         uiMode & Configuration.UI_MODE_NIGHT_MASK;
5     switch (nightModeFlags){
6         case Configuration.UI_MODE_NIGHT_YES:
7             return true;
8         case Configuration.UI_MODE_NIGHT_NO:
9             return false;
10        case Configuration.UI_MODE_NIGHT_UNDEFINED:
11            return false;
12    }
13    return false;
14 }

```


13 }

Listing 9. DarkMode sprawdzanie motywu

4.4. Nawigacja

Tworzenie menu nawigacji 10 (s. 17) – Najpierw w layoucie strony trzeba zaimplementować widget, który umożliwi wysuwanie się szuflad z krawędzi okna:

1 <androidx.drawerlayout.widget.DrawerLayout>

Listing 10. Tworzenie menu nawigacji

Następnie w listnerze 11 (s. 17) dodajemy bibliotekę NavigationView aby można było zaimplementować zawartość wszystkich typów szuflad nawigacyjnych:

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

Listing 11. Tworzenie NavigationView

Kod 12 (s. 17) odpowiada za wygląd naszego menu wysuwanego, do itemów przypisane są ikony, tytuł oraz id.

```

1 <menu xmlns:android="http://schemas.android.com/apk/res/android">
2
3 <group android:checkableBehavior="single">
4 <item
5     android:id="@+id/nav_start"
6     android:icon="@drawable/ic_start"
7     android:title="Strona glowna" />
8 <item
9     android:id="@+id/nav_wydatki"
10    android:icon="@drawable/ic_wydatki"
11    android:title="Wydatki" />
12    <item
13        android:id="@+id/nav_statystyki"
14        android:icon="@drawable/ic_statystyki"
15        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 listnerze 13 (s. 18) 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

Kod 14 (s. 18) odpowiada za wywoływanie wybranej pozycji w menu nawigacji:

```

1  navigationView.setNavigationItemSelectedListener(this);

```

Listing 14. Wywoływanie pozycji nawigacja

Listnere 15 (s. 18) tworzy metodę odpowiedzialną za poruszanie się między stronami. `onNavigationItemSelectedListener` odpowiada za to, że metoda zostaje wywołana po wybraniu elementu z menu nawigacji. Intent służy do przechodzenia między aktywnościami, `startActivity` uruchamia daną aktywność.

(Przykład na podstawie poruszania się ze strony startowej)

```

1  Override
2  public boolean onNavigationItemSelectedListener(@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              );
9              startActivity(a);
10             break;
11
12             case R.id.nav_statystyki:
13                 Intent b = new Intent(startPage.this, statystykiPage.
14                 class);
15                 startActivity(b);
16                 break;
17
18                 case R.id.nav_ustawienia:
19                     Intent c = new Intent(startPage.this, ustawieniaPage.
20                     class);
21                     startActivity(c);
22                     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 obsługuje system weryfikacji biometrycznej. Implementacja następuje w pliku app/build.gradle do której dopisujemy następującą linię kodu 16 (s. 19):

```

1 implementation "androidx.biometric:biometric:1.1.0"

```

Listing 16. Implementacja biblioteki biometric

Następnie w naszym projekcie utworzyliśmy nową aktywność, która będzie odpowiadała za weryfikację biometryczną - nazwaliśmy ją fpPage. Na ekranie layoutu dodaliśmy przycisk odpowiadający za wywołanie weryfikacji - przycisk ikoniczny 4.3 (s. 19).

**Rys. 4.3.** Ikoniczny przycisk biometryki

Kod 17 (s. 20) jest to zawartość pliku fppage.java, który odpowiada za ob-

sługe danych biometrycznych. Linijka 1 pobiera obrazek zaimplementowany wyżej do zmiennej imageViewLogin, następnie wywołujemy BiometricManager'a, oraz od linii 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 wywołujemy okienko autoryzacji, które opisane są 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 |
5      DEVICE_CREDENTIAL)) {
6      case BiometricManager.BIOMETRIC_SUCCESS:
7          Log.d("MY_APP_TAG", "Aplikacja używa danych
8              biometrycznych.");
9              break;
10         case BiometricManager.BIOMETRIC_ERROR_NO_HARDWARE:
11             Toast.makeText(this, "Sensor niedostępny", Toast.LENGTH
12                 LONG);
13             break;
14         case BiometricManager.BIOMETRIC_ERROR_HW_UNAVAILABLE:
15             Log.e("MY_APP_TAG", "Dane biometryczne niedostępne.");
16             break;
17         case BiometricManager.BIOMETRIC_ERROR_NONE_ENROLLED:
18             // Prompts the user to create credentials that your app
19             accepts.
20             final Intent enrollIntent = new Intent(Settings.
21                 ACTION_BIOMETRIC_ENROLL);
22             enrollIntent.putExtra(Settings.
23                 EXTRA_BIOMETRIC_AUTHENTICATORS_ALLOWED,
24                 BIOMETRIC_STRONG | DEVICE_CREDENTIAL);
25             startActivityForResult(enrollIntent, REQUEST_CODE);
26             break;
27     }
28     executor = ContextCompat.getMainExecutor(this);
29     biometricPrompt = new BiometricPrompt(fpPage.this,
30         executor, new BiometricPrompt.AuthenticationCallback() {
31             @Override
32             public void onAuthenticationError(int errorCode,
33                 @NonNull CharSequence
34                 errString) {
35                 super.onAuthenticationError(errorCode, errString);
36                 Toast.makeText(getApplicationContext(),

```

```

30         "Bład autoryzacji: " + errString, Toast.
LENGTH_SHORT)
31         .show();
32     }
33
34     @Override
35     public void onAuthenticationSucceeded(
36         @NonNull BiometricPrompt.AuthenticationResult result)
37     {
38         super.onAuthenticationSucceeded(result);
39         startActivity(new Intent(fpPage.this, startPage.class));
40         Toast.makeText(getApplicationContext(),
        "Autoryzacja powiodła się!", Toast.LENGTH_SHORT).
show();
41     }
42
43     @Override
44     public void onAuthenticationFailed() {
45         super.onAuthenticationFailed();
46         Toast.makeText(getApplicationContext(), "Autoryzacja
niepowiodła się!",
47             Toast.LENGTH_SHORT)
48             .show();
49     }
50 });
51
52 promptInfo = new BiometricPrompt.PromptInfo.Builder()
53     .setTitle("Biometryczne odblokowywanie")
54     .setSubtitle("Odblokuj używając danych biometrycznych")
55     .setNegativeButtonText("Użyj hasła")
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 oczywistą rzeczą w aplikacji mobilnej, tym bardziej w menadżerze 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 pro-

blemów z sortowaniem wydatków w bazie danych. Kod umieszczony w listnerze 18 (s. 22) jest to kod utworzenia właśnie tego "sztywnego" pola wyborów wydatków. Jak widać większość kodu przedstawia zwykłą personalizację elementu, aczkolwiek najważniejsze znajduje się 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 18 (s. 22) w pliku app/src/main/res/values/array.xml, oraz wywołujemy je w XAML'u dzięki nazwie string array'a.

```

1  <Spinner
2  <string-array name="kategorie">
3  <item>Zywnosc i napoje</item>
4  <item>Kredyty</item>
5  <item>Mieszkanie i opłaty</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>Odzież</item>
12 <item>Inne</item>
13 </string-array>

```

Listing 19. Array kategorie

5. Testowanie

6. Podręcznik użytkownika

Spis rysunków

1.1. Logo	3
2.1. Schemat aplikacji	5
3.1. Strona główna aplikacji	8
3.2. Strona podglądu wszystkich wydatków	10
4.1. Wygląd bazy danych.	15
4.2. Wygląd bazy danych cd.	15
4.3. Ikoniczny przycisk biometryki	19

Spis tabel

Spis listingów

1.	Firebase Auth naciśnięcie przycisku rejestracji	11
2.	Firebase Auth - Rejestracja	11
3.	Sprawdza zalogowanie	12
4.	Listner przycisku register	12
5.	System logowania	13
6.	night/colors.xml	14
7.	Xml DarkMode	14
8.	DarkMode Java Button obsługa	16
9.	DarkMode sprawdzanie motywu	16
10.	Tworzenie menu nawigacji	17
11.	Tworzenie NavigationView	17
12.	menu/main menu.xml	17
13.	Wysuwanie się nawigacji	18
14.	Wywoływanie pozycji nawigacja	18
15.	Tworzenie poruszanie się między stronami	18
16.	Implementacja biblioteki biometric	19
17.	Biometryka w pliku fppage.java	20
18.	TextSpinner XAML	22
19.	Array kategorie	22