



Projekt z przedmiotu Wprowadzenie do technologii mobilnych

Temat: *“ExpenseApp” - aplikacja Android do zarządzania wydatkami*

Sylwia Kwiatkowska, Mikołaj Ogarek, Zuzanna Śmiech



Kraków, CZERWIEC 2020

Opis aplikacji

Cel aplikacji: Umożliwienie śledzenia na bieżąco swoich wpływów i wydatków oraz obliczanie ich bilansu.

Podstawowe Funkcjonalności:

1. *Rejestracja*

Użytkownik może zarejestrować się i utworzyć własne konto zabezpieczone hasłem poprzez podanie loginu w postaci adresu e-mail oraz dwukrotne podanie hasła spełniającego warunki (1 duża litera, 1 cyfra, 1 znak specjalny, minimalnie 8 znaków). Po zarejestrowaniu się użytkownik jest przekierowany bezpośrednio na stronę logowania. Na danym urządzeniu nie będzie możliwe zarejestrowanie ponownie użytkownika o takim samym loginie.

2. *Logowanie*

Po dokonanej uprzednio rejestracji, użytkownik loguje się w aplikacji podając login oraz hasło. Istnieje możliwość przejścia do ekranu rejestracji. Po zalogowaniu użytkownik ma dostęp do menu, z którego może sprawdzić saldo konta, dodać wydatek, dodać zarobek oraz zrestartować licznik wydatków.

3. *Wydatek*

Ekran dodawania wydatku ukazuje się po kliknięciu w menu na przycisk "SPEND". Użytkownik wprowadza kwotę oraz wybiera rodzaj wydatku. Dostępne rodzaje wydatków: BILLS, SHOPPING, MORTGAGE RATE, FOOD, PARTY, GIFTS, OTHERS. Po wprowadzeniu kwoty i rodzaju wydatku oraz przyciśnięciu przycisku "SPEND", użytkownik jest informowany o dodaniu wydatku i przekierowany bezpośrednio do ekranu statystyk.

4. *Zarobek*

Ekran dodawania zarobku ukazuje się po kliknięciu w menu na przycisk "EARN". Użytkownik wprowadza kwotę wpływu oraz klika "EARN". Ukazana jest informacja o zarobku i przekierowanie do ekranu statystyk.

5. *Statystyki*

Ekran statystyk ukazuje się po kliknięciu w menu na przycisk "STATISTICS". Użytkownik widzi jaki jest bilans wydatków oraz poszczególne wydatki, ich kwotę oraz rodzaj. Ukazane są również wpływy (ich kwota).

6. *Usunięcie danych*

Ekran usuwania danych ukazuje się po kliknięciu w menu na przycisk "RESET". Po zaakceptowaniu, wszystkie wydatki oraz wpływy są usuwane, a początkowe saldo konta ustawiany na 0.

Dodatkowe funkcjonalności:

- Podczas procesu logowania oraz rejestracji odtwarzana jest relaksacyjna muzyka umilająca ten, być może, stresujący dla wielu osób proces.

- Gdy dodawany jest wydatek, po którym saldo zmienia się z dodatniego na ujemny, aplikacja uruchamia powiadomienie push o zaistniałej sytuacji.
- Po przejściu w tryb samolotowy, użytkownik jest informowany o konieczności przewalutowania swoich wydatków za granicą na walutę PLN.

Spełnienie wymagań:

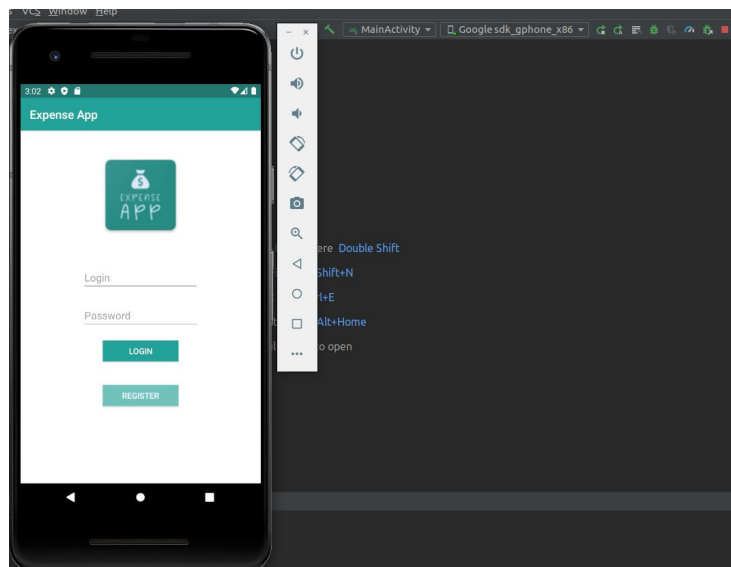
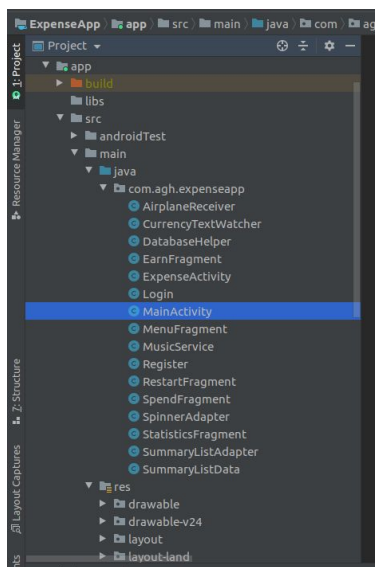
Aplikacja spełnia wymagania projektu, gdyż:

- Realizuje użyteczną funkcjonalność dla użytkownika
- Posiada GUI
- Używa fragmentów (Expense Activity: SpendFragment, EarnFragment, ResetFragment, StatisticsFragment, MenuFragment)
- Obsługuje orientację poziomą i pionową
- W orientacji poziomej następuje podmiana fragmentów (widok 2 fragmentów naraz).
- Posiada bazę danych SQLite, z której korzysta
- Używa serwisu do uruchomienia muzyki
- Jest zarejestrowana jako broadcast receiver (tryb samolotowy)
- Wykorzystuje push notyfikację

Inicjalizacja Projektu

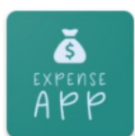
Aby uruchomić aplikację “ExpenseApp” z wykorzystaniem środowiska programistycznego Android studio należy:

1. Pobrać i poprawnie skonfigurować Android Studio zgodnie z [linkiem do instrukcji poprawnej konfiguracji tego środowiska programistycznego](#) .
2. Sklonować projekt znajdujący się w [repozytorium GitHub projektu ExpenseApp](#) .
Polecenie: `git clone <link tutaj>`
3. Zaimportowanie projektu ExpenseApp/app do środowiska
4. Poprawne skonfigurowanie emulatora bądź podpięcie do komputera telefonu z systemem Android zgodnie z [instrukcją na stronie](#) .
5. Uruchomienie MainActivity na urządzeniu:



Realizacja GUI

Aplikacja działa w orientacji pionowej i poziomej. Poniżej prezentacja widoków aplikacji z krótkim opisem.



Login

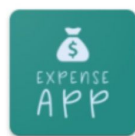
Password

LOGIN

REGISTER



Widok logowania (pierwszy widok aplikacji, dodatkowo gra muzyka):



Email

Password

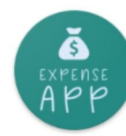
Confirm Password

REGISTER

BACK TO LOGIN



Widok rejestracji:
W przypadku złych danych rejestracji komunikat "Invalid data inserted" w przypadku poprawnej rejestracji "Registered Successfully"



MENU

STATISTICS

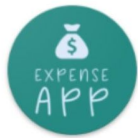
EARN

SPEND

RESET



Główne menu aplikacji (za jego pomocą przechodzimy do widoków pozwalających dodać wydatki/przychody, obejrzeć statystyki lub zresetować aplikację).



NEW SPENDING

Amount cash in PLN

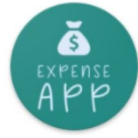
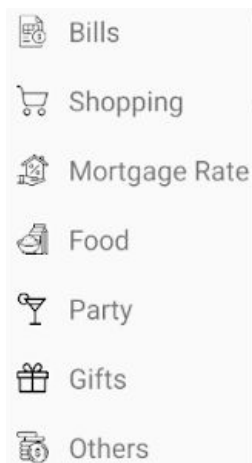
 Bills ▾

SPEND

BACK



Dodawanie wydatku
podajemy
kwotę oraz
wybieramy
kategorię
jedną z
dostępnych:



NEW EARNING

Amount cash in PLN

EARN

BACK










Dodawanie przychodu



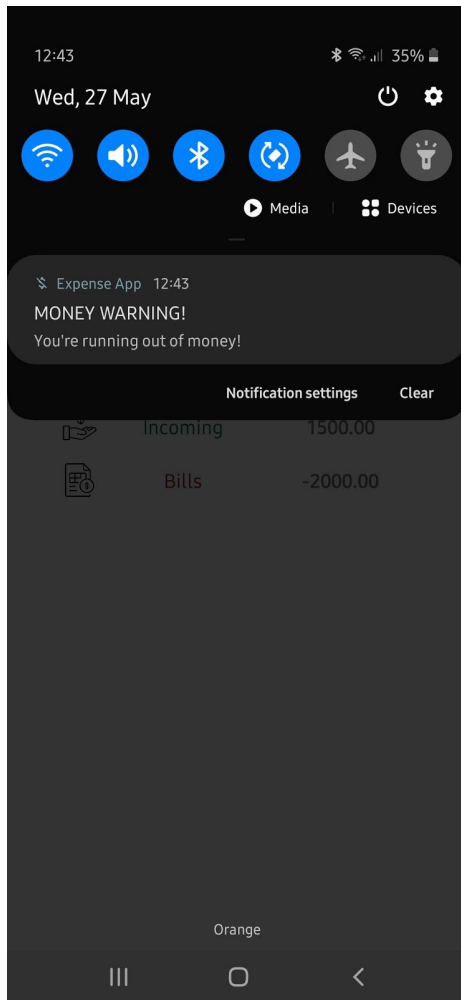
BACK

General balance
PLN 16934.1

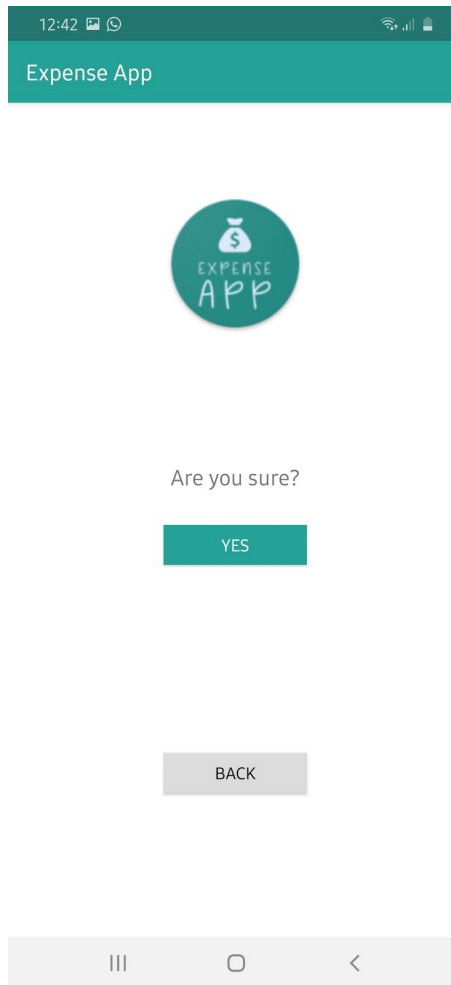
	Incoming	1500.00
	Bills	-2000.00
	Party	-365.88
	Incoming	20000.00
	Gifts	-250.00
	Mortgage Rate	-950.00
	Shopping	-1000.00



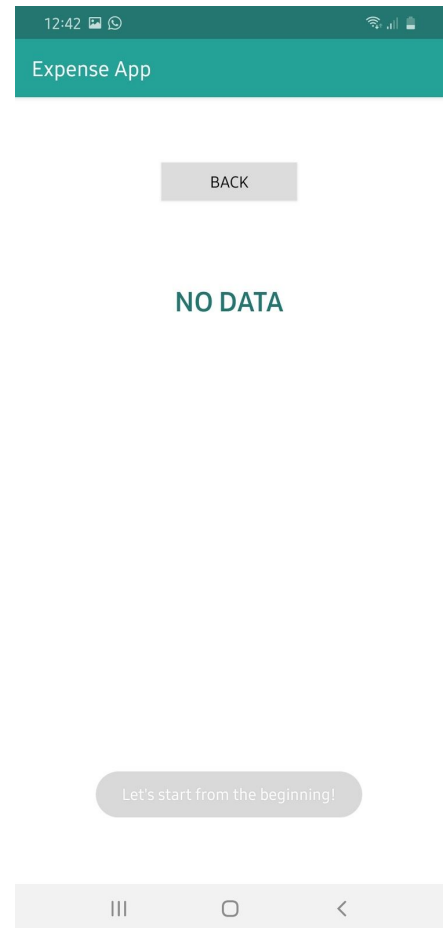
Widok statystyk ogólne
rozliczenie statystyk od początku
rejestracji użytkownika w
aplikacji.



Powiadomienia push w przypadku, gdy saldo użytkownika jest ujemne.



Komunikat potwierdzający reset konta (usunięcia całej historii aplikacji).



Widok statystyk dla konta użytkownika bez żadnej historii wpłat/wypłat.

Realizacja logiki biznesowej

Aplikacja przewiduje obsługę żądań takich jak:

- Dodanie nowego użytkownika - tworzenie nowych tabeli użytkownika i jego wydatków,
- Autoryzacja i autentykacja użytkownika - sprawdzenie czy użytkownik istnieje w bazie danych oraz czy jego dane logowania są poprawne,
- Dodanie nowego wydatku/przychodu - dodanie nowego rekordu do tabeli,
- Wyświetlenie obecnych statystyk - podsumowanie rekordów bazy danych,
- Reset aplikacji - usuwanie rekordów wszystkich rekordów użytkownika z tabeli wydatków,
- Wysłanie powiadomienia push kiedy saldo konta jest ujemne,
- Wysyłanie komunikatów potwierdzających wykonanie danej akcji.

Realizacja warstwy danych

Wszystkie dane przechowywane są w SQLite (również dane użytkownika, nie używaliśmy SQLCipher). Dostęp do danych został udostępniony warstwie logiki biznesowej za pomocą klasy `DatabaseHelper` posiadającej następujące metody:

- `public void onCreate(SQLiteDatabase db)` - utworzenie tabel "user" i "balance",
- `public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)` - usuwanie tabel "user" i "balance"
- `public boolean insertUser(String email, String password)` - dodanie nowego rekordu do tabeli "user"
- `public boolean insertPositiveBalance(String email, String amount, String purpose)` - dodanie rekordu (zarobku) do tabeli "balance"
- `public boolean insertNegativeBalance(String email, String amount, String purpose)` - dodanie rekordu (wydatku) do tabeli "balance"
- `public Cursor getBalance(String email)` - wydobyć wszystkich rekordów z tabeli "balance" należących do użytkownika email
- `public void deleteBalance(String email)` - usuwanie wszystkich rekordów z tabeli "balance" należących do użytkownika email
- `public Boolean checkIfEmailExists(String email)` - sprawdzenie czy w tabeli "user" znajduje się użytkownik email
- `public Boolean checkEmailAndPasswordIsCorrect(String email, String password)` - sprawdzenie czy podane dane znajdują się w tabeli "user"

Zapraszamy do sekcji [HOW-TO] pokazującej jak wykonaliśmy niektóre elementy aplikacji.

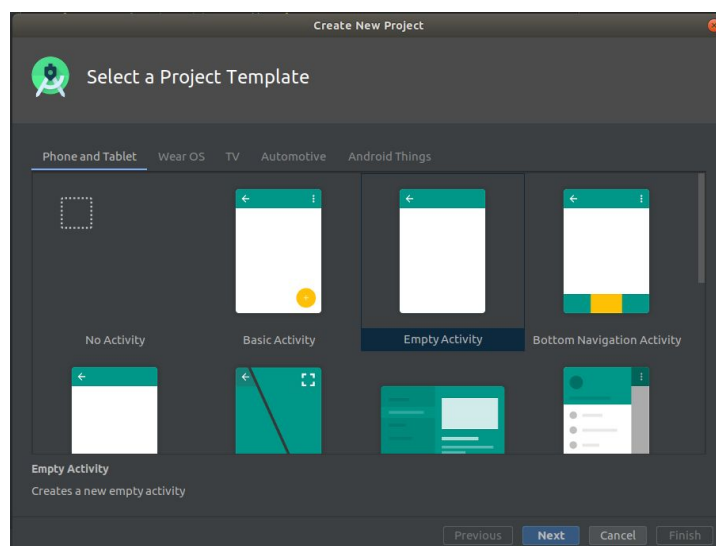
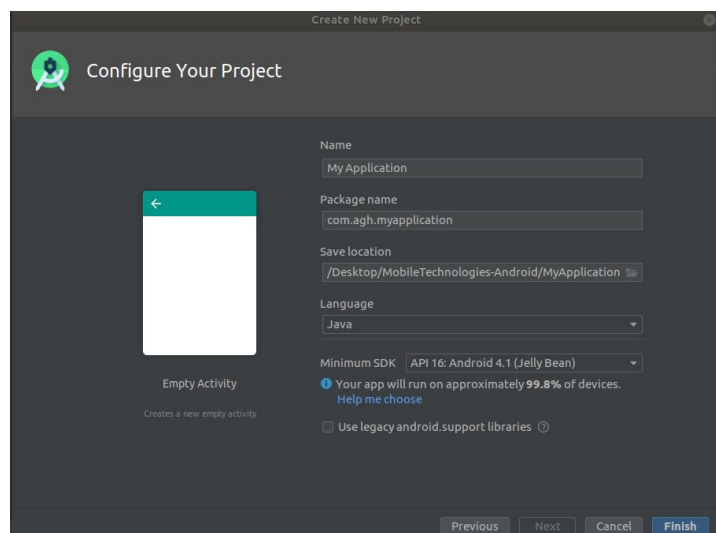
[HOW-TO] Inicjalizacja nowego projektu w Android Studio

File > New > New Project > Empty Activity

Uzupełniamy pola:

- Name
- Package Name
- Save location
- Language (Mamy do wyboru Java lub Kotlin)
- Minimum SDK

Klikamy Finish.



[HOW-TO] Utworzenie bazy danych oraz korzystanie z niej

Aplikacja mobilna z możliwością rejestracji, logowania bądź przechowywania informacji nie może działać bez bazy danych. W tym projekcie wykorzystaliśmy SQLite w Androidzie, które umożliwia w sposób łatwy zaimplementowanie bazy danych SQL poprzez klasę DatabaseHelper dziedziczącą po SQLiteOpenHelper.


```

+ DatabaseHelper extends SQLiteOpenHelper
- fields
+ constructors
+ DatabaseHelper(context: Context)
+ methods
+ onCreate(db: SQLiteDatabase): void
+ onUpgrade(db: SQLiteDatabase, oldVersion: int, newVersion: int): void
+ insertUser(email: String, password: String): boolean
+ insertPositiveBalance(email: String, amount: String, purpose: String): boolean
+ insertNegativeBalance(email: String, amount: String, purpose: String): boolean
+ getBalance(email: String): Cursor
+ deleteBalance(email: String): void
+ getBalanceSum(email: String): String
+ checkIfEmailExists(email: String): Boolean
+ checkEmailAndPasswordIsCorrect(email: String, password: String): Boolean

```

Klasa DatabaseHelper, która dziedziczy po SQLiteOpenHelper jest mechanizmem tworzącym i/lub aktualizującym strukturę naszej bazy. Posiada ona dwie nadpisane metody:

- onCreate(SQLiteDatabase db) – metoda wywoływana w momencie, gdy odwołujemy się do bazy danych, która jeszcze fizycznie nigdzie nie istnieje. W naszym przypadku wykonujemy kod SQL, który stworzy naszą tabelę.
- onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) – metoda ściśle związana z wersją naszej bazy. Wywoływana jest w momencie gdy w urządzeniu istnieje starsza wersja bazy, i służy do aktualizacji jej struktury, do wersji najnowszej.

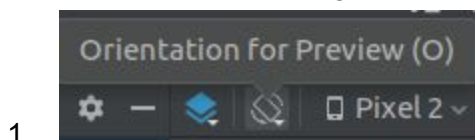
Dodatkowo klasa posiada szereg metod umożliwiających dostęp do bazy danych w tym między innymi dodawanie nowych użytkowników, dodawanie wydatków, wyszukiwanie pozycji dla konkretnego użytkownika itp.

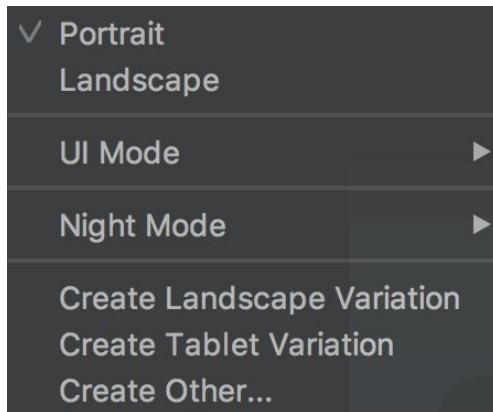
[HOW-TO] Obsługiwanie orientacji poziomej i pionowej

Każda aktywność bądź fragment posiada odpowiadający sobie layout, czyli wygląd-ułożenie elementów w GUI aplikacji. Jednym z zadań w tym projekcie było umożliwienie sprawnego i efektywnego korzystania z aplikacji w orientacji poziomej i pionowej.

Utworzenie widoku dla aktywności: res > layout > PPM: New > Layout resource file.

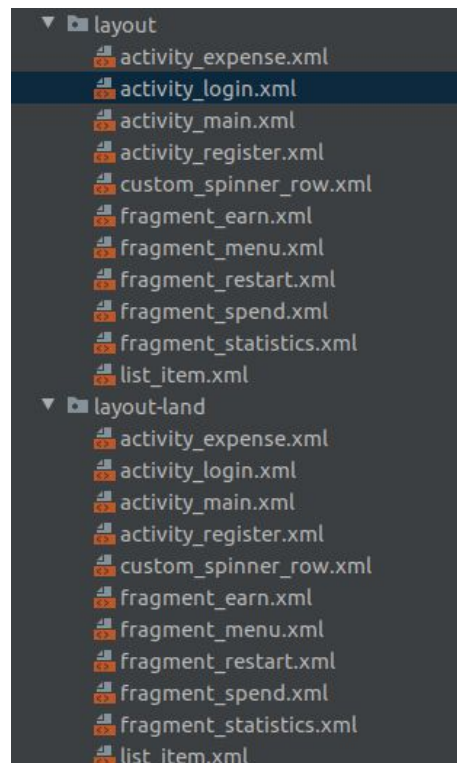
Utworzenie komplementarnego widoku pionowego:





- 2.
3. Create Landscape Variation.
Dokonujemy zmian dla orientacji poziomej.

Poniżej lista plików dla widoków w GUI dla orientacji poziomej i pionowej.



[HOW-TO] Podmiana fragmentów

Obiekty Fragment są świetnym narzędziem, które przydaje się w wielu problematycznych sytuacjach, w których tylko klasy Activity były dostępne. Fragmenty pozwalają na organizowanie komponentów interfejsu projektu dla różnych urządzeń, dając możliwość wyświetlania wielu

segmentów interfejsu na dużym ekranie, np. na tablecie lub wyświetlać jeden i połączyć wszystkie ze sobą na mniejszym ekranie.

Pomagają również podzielić kod na łatwe do zarządzania kawałki, bez potrzeby polegania na dużych i skomplikowanych klasach Activity.

W naszej aplikacji mamy następujące klasy Fragment:

- EarnFragment
- MenuFragment
- RestartFragment
- SpendFragment
- StatisticsFragment

Każdy z powyższych fragmentów odpowiada za część funkcjonalności całej aplikacji, posiada swój własny widok.

Dzięki zastosowaniu fragmentów możemy w orientacji poziomej wyświetlać wiele informacji naraz, które nie zmieściłyby się w orientacji pionowej (na przykład w orientacji poziomej po lewej stronie znajduje się menu natomiast po prawej widok statystyk, dodawania wydatków, bądź resetowania wydatków).

Aby pomóc w zarządzaniu fragmentami, Android zapewnia FragmentManager. Każde Activity posiada Android.App.FragmentManager, który znajdzie lub dynamicznie zmieni jego fragmenty. Każdy zestaw tych zmian jest znany jako transakcja i jest wykonywany przy użyciu Android.App.FragmentTransaction.

Zmiany fragmentów dokonujemy korzystając między innymi z metod: add(), remove(), replace(), commit(). Zmiany w transakcji nie są wykonywane natychmiast. Zamiast tego są one zaplanowane do uruchomienia w wątku interfejsu użytkownika działania tak szybko, jak to możliwe.

Przykład zamiany fragmentu z użyciem FragmentManager'a i metody replace():

```
b1.setOnClickListener((v) -> {  
    SpendFragment spendFragment = new SpendFragment();  
    spendFragment.setArguments(arguments);  
    FragmentTransaction transaction = getFragmentManager().beginTransaction();  
    int orientation = getResources().getConfiguration().orientation;  
    if (orientation == Configuration.ORIENTATION_LANDSCAPE) {  
        transaction.replace(R.id.fragment_container2, spendFragment);  
    }  
    else{  
        transaction.replace(R.id.fragment_container, spendFragment);  
    }  
    transaction.commit();  
});
```

[HOW-TO] Broadcast Receiver

3 kroki do twojego własnego receiver'a:

1. W pliku AndroidManifest.xml znajdującym się w katalogu manifest dodajemy nasz receiver:

<application

...

```
<receiver
    android:name=".AirplaneReceiver"
    android:exported="false">
```

```
</receiver>
```

</application>

2. Tworzymy klasę, która dziedziczy po klasie BroadcastReceiver.
3. Implementujemy metodę onReceive().

Przykład implementacji klasy AirplaneReceiver.

```
package com.agh.expenseapp;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class AirplaneReceiver extends BroadcastReceiver {
    /**
     * Class responsible for handling Airplane mode status changed.
     * Registers as Broadcast Receiver
     */

    @Override
    public void onReceive(Context context, Intent intent) {
        /**
         * Method makes a toast informing user to mind currency changes while abroad
         */
        String intentAction = intent.getAction();
        if (intentAction == Intent.ACTION_AIRPLANE_MODE_CHANGED) {
            Toast.makeText(context, "Please remember to convert a currency into PLN while abroad", Toast.LENGTH_LONG).show();
        }
    }
}
```

[HOW-TO] Utworzenie serwisu odtwarzającego muzykę

Android, jak większość współczesnych systemów operacyjnych, zawiera bardzo rozbudowane mechanizmy do pracy w tle. Jednym z takich mechanizmów są Usługi (Services), których zadaniem jest przetwarzanie zadań nie wymagających (lub wymagających w niewielkim stopniu) ingerencji użytkownika.

Oto fragment kodu naszej Usługi umożliwiającej odtwarzanie muzyki w trakcie logowania.

```
public class MusicService extends Service {  
    /*  
    Class enabling playing music  
    */  
    private MediaPlayer player;  
    @Override  
    public int onStartCommand(Intent intent, int flags, int startId) {  
        super.onStartCommand(intent, flags, startId);  
        player = MediaPlayer.create(context, this, R.raw.water);  
        player.setLooping(true);  
        player.start();  
        return START_STICKY;  
    }  
  
    @Override  
    public void onDestroy() {  
        super.onDestroy();  
        player.stop();  
    }  
}
```

Klasa, za pomocą której implementujemy ten komponent posiada własny cykl życia.

Cykl życia Usługi:

Podobnie do Aktywności, Usługi posiadają szereg metod związanych z ich cyklem życia. Mamy tutaj zatem:

- onCreate() wywoływane w momencie tworzenia Usługi,
- onStartCommand() wywoływane w momencie startu usługi
- onDestroy() wywoływane w momencie kończenia działania Usługi.