



Politechnika Łódzka

Instytut Informatyki

## PRACA DYPLOMOWA INŻYNIERSKA

# APLIKACJA MOBILNA WSPOMAGAJĄCA PLANOWANIE PODRÓŻY

MOBILE APPLICATION SUPPORTING THE TRAVEL  
PLANNING

Wydział Fizyki Technicznej, Informatyki i Matematyki Stosowanej

Promotor: dr inż. Radosław Bednarski

Dypломант: Leonard Hovhannisyan

Nr albumu: 187203

Specjalność: Technologie Gier i Symulacji Komputerowych

Łódź, 6.02.2019



Instytut Informatyki

90-924 Łódź, ul. Wólczańska 215, budynek B9

tel. 042 631 27 97, 042 632 97 57, fax 042 630 34 14 email: office@ics.p.lodz.pl

# Spis treści

<b>1 Wstęp</b>	<b>2</b>
1.1 Urządzenia mobilne i system Android . . . . .	2
1.2 Branża turystyczna . . . . .	2
1.3 Cel i zakres pracy . . . . .	3
<b>2 Analiza istniejących rozwiązań</b>	<b>5</b>
2.1 PackKing . . . . .	5
2.2 PackPoint travel packing list . . . . .	6
2.3 My Luggage Checklist . . . . .	6
2.4 Porównanie istniejących rozwiązań . . . . .	7
<b>3 Opis narzędzi, praktyk programistycznych i programów potrzebnych do realizacji projektu</b>	<b>9</b>
3.1 Android Studio . . . . .	9
3.1.1 Room Persistence Library . . . . .	9
3.1.1.1 Komponent Database w aplikacji . . . . .	11
3.1.1.2 Komponenty Entity w aplikacji . . . . .	14
3.1.1.3 Komponenty Data Access Object w aplikacji . . . . .	15
3.2 StarUML . . . . .	17
3.3 GIMP . . . . .	17
<b>4 Opis i realizacja aplikacji</b>	<b>19</b>
4.1 Założenia . . . . .	19
4.2 Klasy aktywności . . . . .	19
4.3 Funkcjonalności i interakcje . . . . .	20
4.3.1 Główny ekran . . . . .	20
4.3.2 Ekran list . . . . .	23
4.3.3 Ekran przedmiotów . . . . .	25
4.3.4 Ekran szablonów . . . . .	28
4.3.5 Ekran generowania listy rzeczy do spakowania . . . . .	29
4.3.6 Dodawanie przedmiotu . . . . .	32
4.3.7 Edycja przedmiotu . . . . .	37
4.3.8 Dodawanie szablonu . . . . .	37
<b>5 Podsumowanie</b>	<b>44</b>
<b>6 Literatura</b>	<b>47</b>

7 Spis rysunków	49
8 Spis tabel	51
9 Spis listingów	52

# 1 Wstęp

## 1.1 Urządzenia mobilne i system Android

W ciągu ostatniej dekady urządzenia mobilne zaczęły przodować w statystykach ruchu w Internecie. Na początku roku 2009 niemal cały ruch w sieci inicjowany był przez komputery stacjonarne oraz laptopy, natomiast w roku 2018 smartfony oraz tablety mają już 56% udziału przy 43% udziału laptopów i desktopów (pozostały 1% to inne urządzenia np. konsole) [4]. Trendy pokazują, że udział urządzeń mobilnych będzie stale rósł i to one będą decydować o ruchu sieciowym w najbliższej przyszłości.

Rosnąca popularność urządzeń mobilnych wiąże się z korzystaniem z aplikacji dostępnych na nich. Na rynku mobilnym obecnie mamy zjawisko duopolu, który nigdy nie był do tej pory tak wyraźny. Zdecydowany prym na rynku wiodą urządzenia z systemem Android mające w 2017 roku 85,9% udziałów, niemalże całą resztę urządzeń obsługuje system iOS (14% udziałów) [5]. Nierówną walkę o udział na rynku mobilnym z wyżej wymienionymi markami przegrały znane firmy takie jak Nokia, Blackberry czy Microsoft. Jak widać Android jest zdecydowanym liderem wśród urządzeń mobilnych, więc aplikacje dedykowane dla tego systemu mają największy popyt.

## 1.2 Branża turystyczna

Branża turystyczna jest jedną z najlepiej rozwijających się gałęzi usług. Z biegiem czasu coraz więcej państw otwiera swoje granice, ułatwia podróże, a linie lotnicze raczą klientów coraz lepszymi ofertami. Poszerzenie strefy Schengen w Europie oraz lepsze i tańsze połączenia do krajów egzotycznych, sprawiają że liczba lotów na przestrzeni ostatnich 20 lat wzrosła niemal trzykrotnie [1].

Podróże są nieodłączną częścią współczesnego społeczeństwa. Najlepiej świadczyć o tym może niedawno pobity rekord świata – 29 czerwca 2018 r. jednocześnie na niebie znajdowało się ponad 19 tys. samolotów pasażerskich [2], natomiast w przeciągu 24 godzin tego samego dnia odnotowano ponad 200 tysięcy lotów [2], co było jedynym takim przypadkiem w historii lotów pasażerskich.

Jak mówią dane z 2017 roku, głównym celem wyjazdów zagranicznych w Polsce jest turystyka i wypoczynek [3]. Wyjazdy turystyczne wymagają odpowiedniego przygotowania, zarówno pod względem formalnym (dokumenty podróży, wizy) jak i pod względem turystycznym. Wiele osób nie wie jakie atrakcje czekają je w miejscu docelowym, co niezbędnego powinni ze sobą zabrać i w jakiej ilości. Ważnym elementem układania planu podróży jest również prognoza pogody na czas wyjaz-

du. Sprawdzanie wszystkich składowych elementów potrzebnych do zorganizowania podróży marzeń, może być zadaniem zbyt czasochłonnym lub problematycznym. Z analizy powyższych danych można wywnioskować, że rozwiązanie korzystające z dominacji Androida na rynku oraz szerzenia się popularności podróży może cieszyć się dużym zainteresowaniem i popytem.

### **1.3 Cel i zakres pracy**

Celem pracy jest stworzenie aplikacji mobilnej w systemie Android (wersje od 6.0 wzwyż) wspomagającej proces planowania podróży, sprawdzanie prognozy pogody w miejscu docelowym, generowanie na podstawie wybranych aktywności edytowalnych list rzeczy niezbędnych do spakowania oraz priorytetyzowanie produktów na nich, dodawanie i edycja własnych produktów oraz kalkulowanie wagi bagażu na podstawie wygenerowanej listy.

Zakres obejmuje następujące zagadnienia:

- Analizę dostępnych na rynku istniejących rozwiązań z zakresu organizacji i planowania podróży
- Zapoznanie się z narzędziami, które służą do tworzenia oprogramowania na system Android
- Stworzenie i zaprojektowanie lokalnej, relacyjnej bazy danych korzystając z wbudowanej biblioteki SQLite
- Stworzenie aplikacji wspomagającej planowanie podróży na urządzenia z systemem Android
- Wygenerowanie pliku z rozszerzeniem .apk, a następnie instalacja aplikacji na urządzeniu obsługującym system Android
- Porównanie utworzonej aplikacji z rozwiązaniami dostępnymi na rynku pod kątem wyznaczonych kryteriów, za pomocą ankiety wśród potencjalnych użytkowników

W pracy przyjęto następujące założenia

- Do stworzenia aplikacji zostało użyte środowisko programistyczne dla Androida opartego na IntelliJ IDEA- Android Studio. Aplikacja zostanie stworzona z wykorzystaniem API Android w wersji 24
- Stworzona aplikacja będzie rozszerzeniem funkcjonalności rozwiązań komercyjnych dostępnych w sklepie Google Play

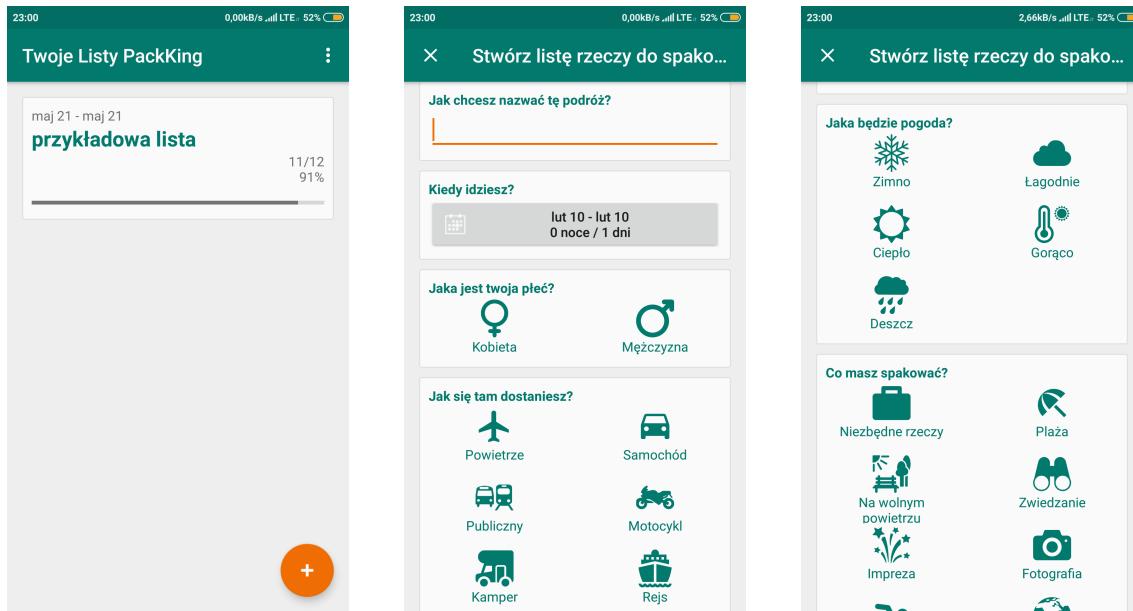
- Do instalacji oraz uruchomienia stworzonej aplikacji zostały wykorzystane telefony: Xiaomi Redmi Note 4, Huawei P10 oraz Samsung Galaxy A5.

## 2 Analiza istniejących rozwiązań

Podróże są popularnym tematem, popularne są również aplikacje wspomagające ich organizację. Dostępnych jest wiele rozwiązań w postaci organizerów podróży, w mojej analizie chciałem się skupić na najpopularniejszych oraz najbardziej elastycznych. Następnie dokonam porównania funkcji wybranych aplikacji, żeby na koniec móc je zestawić ze swoim rozwiązań. Wybranymi przeze mnie aplikacjami są *PackKing* [6], *PackPoint travel packing list* [7] oraz *My Luggage Checklist* [8].

### 2.1 PackKing

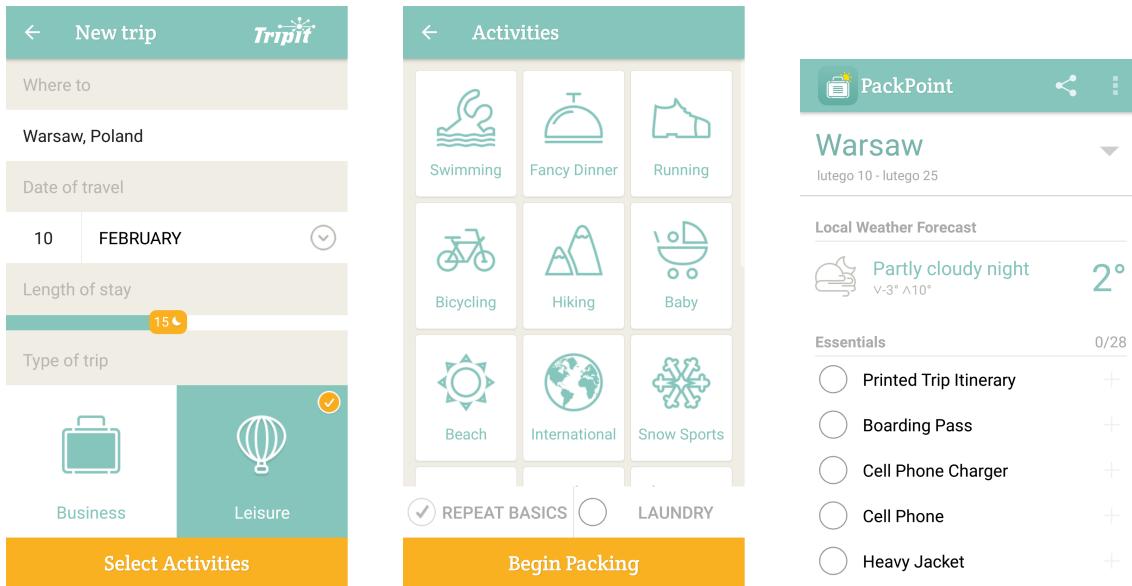
*PackKing* jest popularną aplikacją mającą ponad 100 tys. pobrań oraz 7 tys. opinii w Sklepie Play [6]. Jest to aplikacja bardzo rozbudowana, posiada obszerną bazę danych z wyszczególnionymi aktywnościami, pozwala w niemal dowolny sposób manipulować wygenerowaną listą rzeczy do spakowania, która generuje się na podstawie wybranej długości podróży. Posiada funkcję dodawania własnych przedmiotów oraz wyboru pogody w miejscu docelowym, jednak pogodę użytkownik wybiera sam co pozostawia margines błędu.



Rysunek 2.1: Zrzuty ekranu, prezentujące aplikację PackKing. Ekran główny (po lewej), tworzenie listy rzeczy do spakowania (po środku) wybieranie aktywności w miejscu docelowym (po prawej). Źródło: opracowanie własne

## 2.2 PackPoint travel packing list

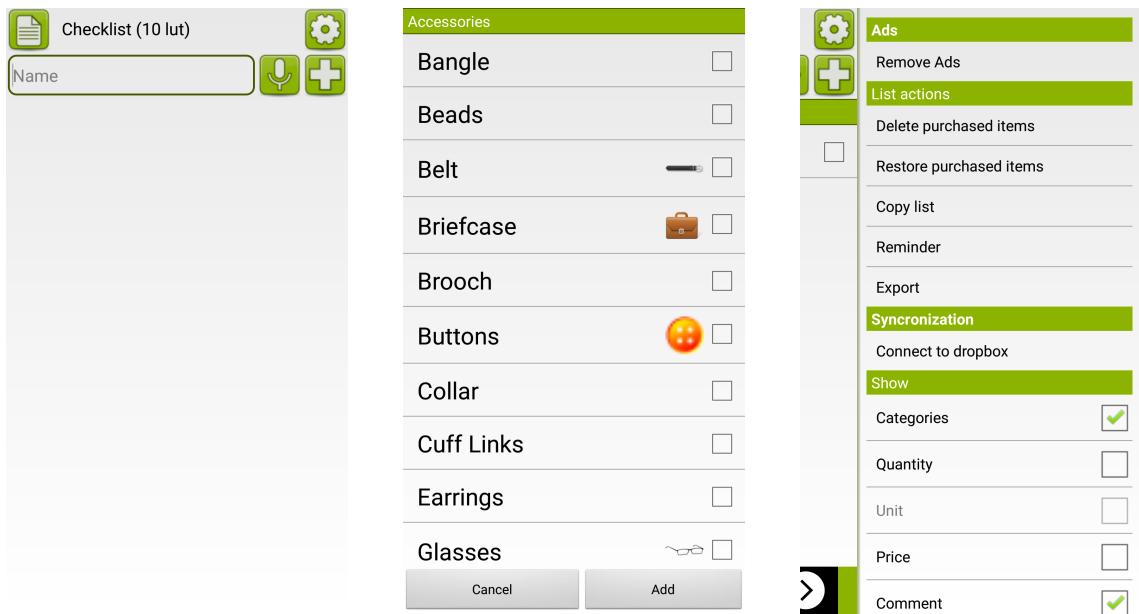
Najpopularniejszą aplikacją, którą będę analizował w mojej pracy jest *PackPoint travel packing list*, z ponad milionem pobrań i 26 tys. opinii w Sklepie Play [7]. Aplikacja poza dobrze wykonanym i intuicyjnym interfejsem, posiada podpięte API od Google Maps oraz API Google Weather, które w połączeniu pokazują nam prognozę pogody w miejscu docelowym naszej podróży. Nie tak rozbudowana możliwość dodawania własnych produktów jak w przypadku *PackKing*, jednak spełnia swoją podstawową rolę. Możliwości edytowania szablonów list nie byłem w stanie sprawdzić, z racji tego że jest to ograniczenie wyłącznie dla wersji premium.



Rysunek 2.2: Zrzuty ekranu, prezentujące aplikację PackPoint. Ekran tworzenia wyjazdki (po lewej), wybieranie aktywności (po środku) wygenerowana lista rzeczy do spakowania(po prawej). Źródło: opracowanie własne

## 2.3 My Luggage Checklist

Ostatnim wybranym przeze mnie rozwiązaniem jest aplikacja *My Luggage Checklist*. Jest ona najmniej popularna z tych przeze mnie opisywanych, ponad 10 tys. pobrań i 150 opinii, jednak po przeanalizowaniu reszty dostępnych aplikacji uznałem, że znajduje się ona na trzecim miejscu pod względem liczby pobrań i pozytywnych recenzji [8]. Aplikacja nie raczy nas tak schludnym interfejsem jak dwie poprzednie, jest prosta a interfejs wygląda przestarzale, posiada ona jednak parę przydatnych funkcji, takich jak określanie ceny produktów, dodawanie ikon do produktów czy przydatna funkcja synchronizacji z serwisem Dropbox.



Rysunek 2.3: Zrzuty ekranu, prezentujące aplikację My Luggage Checklist. Ekran startowy(po lewej), dostępne przedmioty (po środku) pasek boczny menu(po prawej). Źródło: opracowanie własne

## 2.4 Porównanie istniejących rozwiązań

Na koniec rozdziału dokonam porównania poszczególnych rozwiązań pod względem posiadanych funkcjonalności. Porównanie zawarte zostanie w tabeli 2.1.

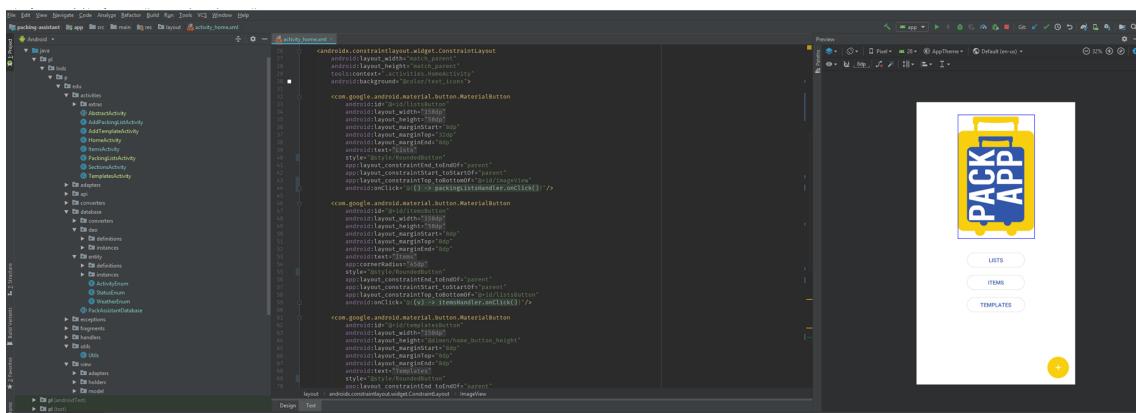
Tabela 2.1: Porównanie dostępnych funkcji najpopularniejszych aplikacji wspomagających planowanie podróży

Funkcjonalność	PackKing	PackPoint	My Luggage Checklist
Generowanie listy rzeczy do spakowania na podstawie aktywności w miejscu docelowym	dostępne	dostępne	niedostępne
Możliwość dodawania własnych przedmiotów	dostępne	dostępne	dostępne
Możliwość dodawania własnych aktywności	niedostępne	dostępne	niedostępne
Możliwość edytowania listy przedmiotów do spakowania	dostępne	dostępne	dostępne
Możliwość tworzenia własnych szablonów list do pakowania	dostępne	dostępne	niedostępne
Eksport listy do pliku PDF	dostępne	dostępne	dostępne
Synchronizacja z zewnętrznymi serwisami	niedostępne	niedostępne	dostępne
Możliwość wyboru pogody dla jakiej użytkownik chce wygenerować listę	dostępne	niedostępne	niedostępne
Sprawdzanie prognozy pogody w miejscu docelowym	niedostępne	dostępne	niedostępne
Generowanie list pakowania w zależności od pogody w miejscu docelowym	niedostępne	dostępne	niedostępne
Generowanie list pakowania w zależności od długości pobytu w miejscu docelowym	dostępne	dostępne	niedostępne
Możliwość odhaczania przedmiotów zapakowanych	dostępne	dostępne	dostępne
Archiwizowanie odbytych podróży	dostępne	dostępne	dostępne

### 3 Opis narzędzi, praktyk programistycznych i programów potrzebnych do realizacji projektu

#### 3.1 Android Studio

Android Studio jest zintegrowanym środowiskiem programistycznym, pozwalającym na tworzenie oprogramowania w systemie Android. Oparty jest na środowisku IntelliJ IDEA i korzysta z jego edycji kodu oraz narzędzi programistycznych. Ponadto środowisko wyposażone jest w Android SDK (Software Development Kit, w moim projekcie używany w wersji 24), który pozwala korzystać z takich narzędzi jak: biblioteki, emulator (możliwość sprawdzania działania aplikacji na różnych urządzeniach, bez konieczności posiadania ich fizycznie), debugger, poradniki, rozbudowany edytor tekstu oraz pozwala korzystać z systemu budowy projektów opartym na Gradle [9]. Android Studio został wykorzystany do stworzenia aplikacji (Rysunek 3.1) zgodnie z wcześniej stworzonym projektem.



Rysunek 3.1: Zrzut ekranu prezentujący widok projektu aplikacji w Android Studio.  
Źródło: opracowanie własne

##### 3.1.1 Room Persistance Library

Android Studio wykorzystuje bazę danych SQLite [11], jednak do wykorzystania w pełni jej możliwości stworzono bibliotekę Room Persistence Library [12], która jest wykorzystana przy budowie aplikacji. Biblioteka Room dostarcza abstrakcyjnej warstwy nad SQLite, dzięki której programista nie musi implementować własnych klas w celu używania bazy danych. Dzięki użyciu Room Persistence Library zapytania SQL sprawdzane są pod kątem poprawności w czasie komplikacji, co zmniejsza prawdopodobieństwo wystąpienia złej kwerendy, a w wyniku zastosowania systemu anotacji, kod jest bardziej czytelny. Ponadto użycie Room Persistence Library sprawia, że użytkownik nie musi implementować klasycznego wzorca projektowego MVC

(pol. *Model-Widok-Kontroler*) stosowanego w aplikacjach do Androida. Biblioteka Room podzielona jest na trzy główne komponenty:

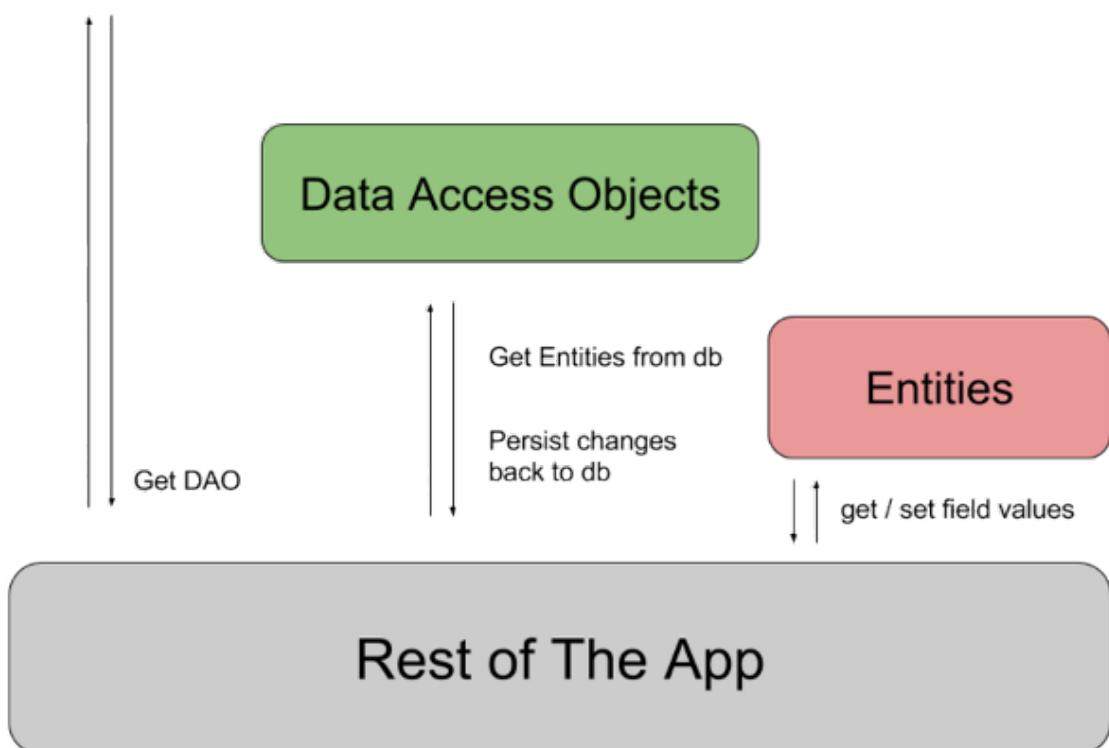
- Database (pol. *baza danych*) - zawiera właściciela bazy danych i jest głównym punktem dostępu do połączenia. Komponent ten oznacza się za pomocą adnotacji `@Database`, a klasa w ten sposób oznaczona powinna spełnić następujące wymagania:
  - być klasą abstrakcyjną, która rozszerza abstrakcyjną klasę `RoomDatabase`
  - zawierać listę encji powiązanych z bazą danych, przy czym lista encji musi być podana jako argument adnotacji, wraz z wersją, np.  
`@Database (entities = { example.class }, version=1 )`
  - zawierać abstrakcyjną, bezargumentową metodę, która zwraca obiekt klasy, której przypisano adnotację `@Dao`
  - z racji wysokich kosztów pamięciowych instancja bazy danych musi być zaimplementowana używając wzorca projektowego singletonu, co zagwarantuje stworzenie tylko jednej instancji danej klasy oraz zapewni globalny dostęp do stworzonego obiektu

W czasie wykonywania można uzyskaćinstancję Database, wywołując metodę `Room.databaseBuilder()` lub `Room.inMemoryDatabaseBuilder()`.

- Entity (pol. *encja*) - jest analogiczna do tabeli w bazie danych
- DAO (Data Access Object, pol. *obiekt dostępu do danych*) - w nim zawarte są kwerendy, dzięki którym uzyskujemy dostęp do bazy danych

Rysunek 3.2 pokazuje powiązania między poszczególnymi komponentami Room Persistence Library.

# Room Database



Rysunek 3.2: Diagram prezentujący związek między komponentami w biliotece Room. Źródło: [12]

### 3.1.1.1 Komponent Database w aplikacji

Komponent *Database* w aplikacji został zaimplementowany zgodnie z wymaganiami wymienionymi w rozdziale 3.1.1. Klasą odpowiedzialną za bazę danych, jest klasa `PackAssistantDatabase`. Listing 3.1 pokazuje spełnienie pierwszego założenia - klasa jest abstrakcyjna oraz rozszerza klasę `RoomDatabase`. Ponadto na załączonym listingu widać założenie drugie - w liście argumentów adnotacji `@Database` jest zawarta lista encji bazy danych.

Listing 3.1: Definicja klasy `PackAssistantDatabase`

```
@Database(entities = {  
    ItemDefinition.class,  
    SectionDefinition.class,  
})
```

```

PackingListDefinition.class,
SectionItemDefinition.class,
PackingListSectionDefinition.class,
ItemImage.class,
SectionInstance.class,
PackingListInstance.class,
SectionItemInstance.class,
PackingListSectionInstance.class}, version = 10, exportSchema = false)
@TypeConverters({DateTypeConverter.class, ActivityEnumTypeConverter.class,
StatusEnumTypeConverter.class, WeatherEnumTypeConverter.class})
public abstract class PackAssistantDatabase extends RoomDatabase {

...
}

```

Listing 3.2 zawiera z kolei spełnienie dwóch kolejnych założeń - abstrakcyjne, bezargumentowe metody, które zwracają obiekt typu `@Dao` oraz zaimplementowanie klasy wg wzorca singletonu.

Listing 3.2: Abstrakcyjne metody oraz użycie wzorca singletonu w klasie `PackAssistantDatabase`

```

public abstract class PackAssistantDatabase extends RoomDatabase {

    private static final String DB_NAME = "pack-assistant-db";

    private static PackAssistantDatabase INSTANCE;

    public abstract ItemDefinitionsDao itemDefinitionsDao();

    public abstract SectionDefinitionsDao sectionDefinitionsDao();

    public abstract PackingListDefinitionsDao packingListDefinitionsDao();

    public abstract SectionItemDefinitionsDao sectionItemDefinitionsDao();

    public abstract PackingListSectionDefinitionsDao packingListSectionDefinitionsDao();

    public abstract PackingListInstancesDao packingListInstancesDao();

    public abstract SectionInstancesDao sectionInstancesDao();
}

```

```

public abstract ItemInstancesDao itemInstancesDao();

public abstract PackingListSectionInstancesDao packingListSectionInstancesDao();

public abstract SectionItemInstancesDao sectionItemInstancesDao();

public synchronized static PackAssistantDatabase getInstance(Context context) {
    if (INSTANCE == null) {
        INSTANCE = buildDatabase(context);
    }

    return INSTANCE;
}

```

Ostatnim istotnym elementem klasy `PackAssistantDatabase` jest metoda tworząca bazę danych, zgodnie z procedurą opisaną w rozdziale 3.1.1, odbywa się to za pomocą użycia metody `Room.databaseBuilder()` (Listing 3.3).

Listing 3.3: Metoda `buildDatabase` w klasie `PackAssistantDatabase`

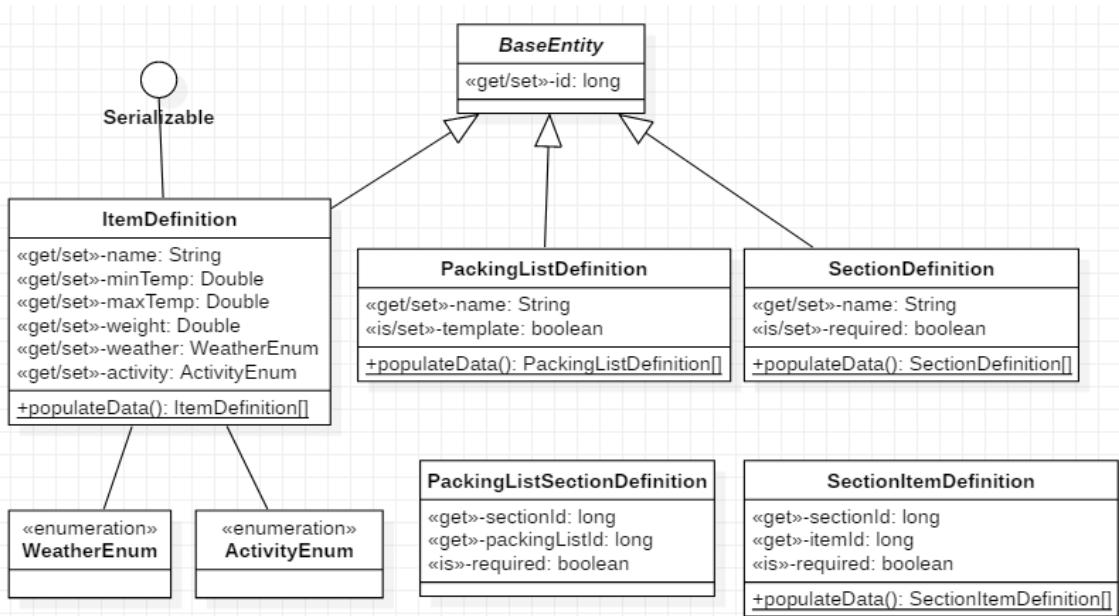
```

private static PackAssistantDatabase buildDatabase(final Context context) {
    return Room.databaseBuilder(context, PackAssistantDatabase.class, DB_NAME)
        .addCallback(new Callback() {
            @Override
            public void onCreate(@NonNull SupportSQLiteDatabase db) {
                super.onCreate(db);
                Executors.newSingleThreadScheduledExecutor().execute(new Runnable
                    () {
                        @Override
                        public void run() {
                            getInstance(context).itemDefinitionsDao().insertAll(
                                ItemDefinition.populateData());
                            getInstance(context).sectionDefinitionsDao().insertAll(
                                SectionDefinition.populateData());
                            getInstance(context).packingListDefinitionsDao().insertAll(
                                PackingListDefinition.populateData());
                            getInstance(context).sectionItemDefinitionsDao().insertAll(
                                SectionItemDefinition.populateData());
                        }
                    });
            }
        })
        .fallbackToDestructiveMigration()
        .build();
}

```

### 3.1.1.2 Komponenty Entity w aplikacji

Komponent *Entity* jest używany w Room Persistence Library jako odpowiednik tabeli w bazie danych. W moim programie na komponent ten składa się klasa abstrakcyjna  *BaseEntity* rozszerzana przez klasy odpowiedzialne za listy, przedmioty oraz sekcje do których one są przypisane. Ponadto do tego komponentu można zaliczyć klasy odpowiedzialne za przypisanie przedmiotów do sekcji oraz przypisanie sekcji do list pakowania. Wszystko to widoczne jest na diagramie UML (Rysunek 3.3).



Rysunek 3.3: Diagram UML przedstawiający komponent Entities. Źródło: opracowanie własne w programie StarUML [13]

Rekordy do tabel baz danych dodawane są w kodach klas Java. Na listingu 3.4 widać fragment klasy *ItemDefinition*, której statyczna metoda  *populateData()* odpowiedzialna jest za dodawanie przedmiotów do bazy danych.

Listing 3.4: Konstruktor klasy *ItemDefinition* oraz jej statyczna metoda  *populateData()*

```

public ItemDefinition(String name, Double maxTemp, Double minTemp, Double weight,
    WeatherEnum weather, ActivityEnum activity) {
    this(name, activity, minTemp, maxTemp);
    this.weather = weather;
    this.weight = weight;
}
  
```

```

...
public static ItemDefinition[] populateData() {
    return new ItemDefinition[]{
        new ItemDefinition("Passport"),
        new ItemDefinition("Money"),
        new ItemDefinition("ID Card"),
        new ItemDefinition("Bike shoes", ActivityEnum.RIDING_A_BIKE),
        new ItemDefinition("Running shoes", ActivityEnum.JOGGING),
        new ItemDefinition("Basketball shoes", ActivityEnum.
            PLAYING_BASKETBALL),
        new ItemDefinition("Football shoes", ActivityEnum.PLAYING_FOOTBALL),
        new ItemDefinition("Swimming trunks", ActivityEnum.SWIMMING),
        new ItemDefinition("Briefcase", ActivityEnum.OTHER, 5.0D, 15.0D),
        new ItemDefinition("Briefcase 2", ActivityEnum.OTHER, 16.0D, 25.0D),
        new ItemDefinition("Briefcase 3", ActivityEnum.OTHER, 26.0D, 35.0D),
        new ItemDefinition("Briefcase 4", ActivityEnum.OTHER, 36.0D, 45.0D),
        new ItemDefinition("Briefcase 5", ActivityEnum.OTHER, -0.6D, 4.0D),
        new ItemDefinition("For Sunny", ActivityEnum.OTHER, WeatherEnum.
            SUNNY),
        new ItemDefinition("For Windy", ActivityEnum.OTHER, WeatherEnum.
            WINDY),
        new ItemDefinition("For Snowy", ActivityEnum.OTHER, WeatherEnum.
            SNOWY),
        new ItemDefinition("For Stormy", ActivityEnum.OTHER, WeatherEnum.
            STORMY),
        new ItemDefinition("For Rainy", ActivityEnum.OTHER, WeatherEnum.
            RAINY)
    };
}

```

### 3.1.1.3 Komponenty Data Access Object w aplikacji

Komponentami DAO (Data Access Object) w stworzonej aplikacji są interfejsy, w których zawarte są kwerenty napisane w języku SQL (Listing 3.5) i to dzięki nim możemy operować na bazie danych.

Listing 3.5: Przykładowa kwerenda SQL z adnotacją @Query użyta w interfejsie oznaczonym adnotacją Dao

```

@Dao
public interface SectionDefinitionsDao {

```

```

    @Query("SELECT * FROM section_definitions")
    Flowable<List<SectionDefinition>> getAll();

    ...
}

```

Komponenty DAO umożliwiają również wstawianie i usuwanie rekordów do tabel bazy danych. Poniżej umieszczony listing interfejsu `ItemDefinitionsDao`, w którym widać wybieranie danych za pomocą kwerendy SELECT, a następnie dodawanie i usuwanie rekordów odpowiednimi metodami, przy użyciu adnotacji `@Insert` oraz `@Delete`(Listing 3.6).

Listing 3.6: Listing interfejsu ItemDefinitionsDao

```

@Dao
public interface ItemDefinitionsDao {

    @Query("SELECT * FROM items_definitions")
    Flowable<List<ItemDefinition>> getAll();

    @Query("SELECT i.id, i.name, i.max_temp, i.min_temp, i.weather, i.activity, i.weight " +
           "FROM items_definitions i " +
           "LEFT JOIN section_item_definitions si ON si.item_id = i.id " +
           "WHERE si.section_id = :id")
    List<ItemDefinition> getBySectionId(Long id);

    @Query("SELECT * FROM items_definitions def WHERE def.id = :id")
    ItemDefinition getById(long id);

    @Insert
    long[] insertAll (ItemDefinition ... itemDefinitions);

    @Query("SELECT i.id, i.name, i.max_temp, i.min_temp, i.weather, i.activity, i.weight " +
           "FROM items_definitions i " +
           "WHERE i.name = :name")
    ItemDefinition getByName(String name);

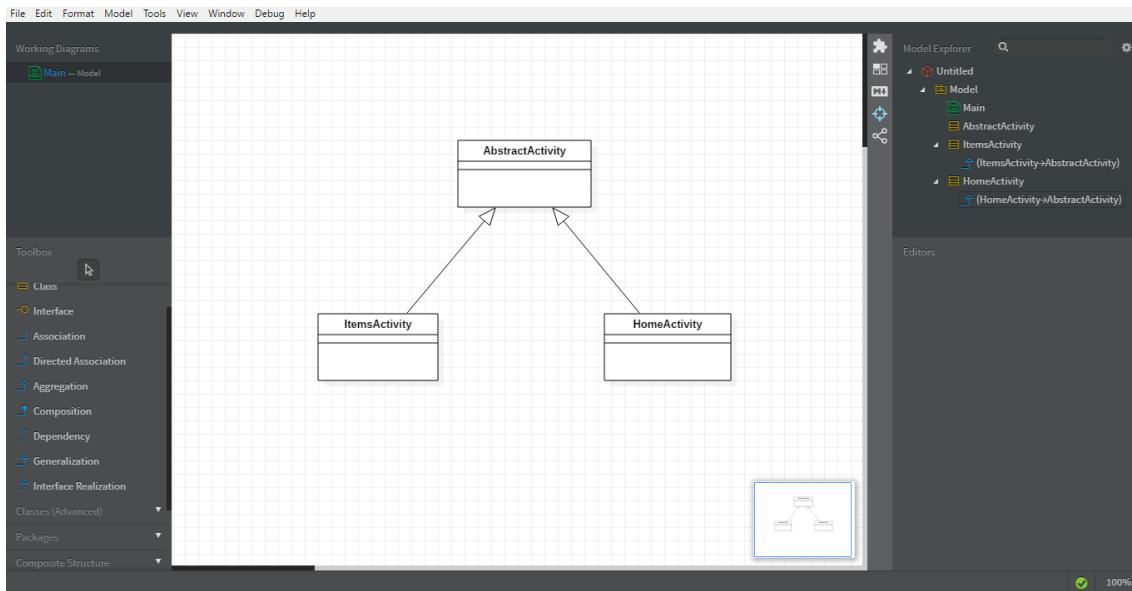
    @Delete
    void delete(ItemDefinition ... itemDefinition);

}

```

## 3.2 StarUML

StarUML to oprogramowanie pozwalające tworzyć diagramy UML (Unified Modeling Language). Pozwala w łatwy i intuicyjny sposób tworzyć między innymi diagramy: klas, sekwencji, przepaków użycia, stanów itd. StarUML jest programem typu „Drag and Drop”, co pozwala na szybkie konstruowanie diagramów (Rysunek 3.4). Wykorzystano program do zaprojektowania całej aplikacji.

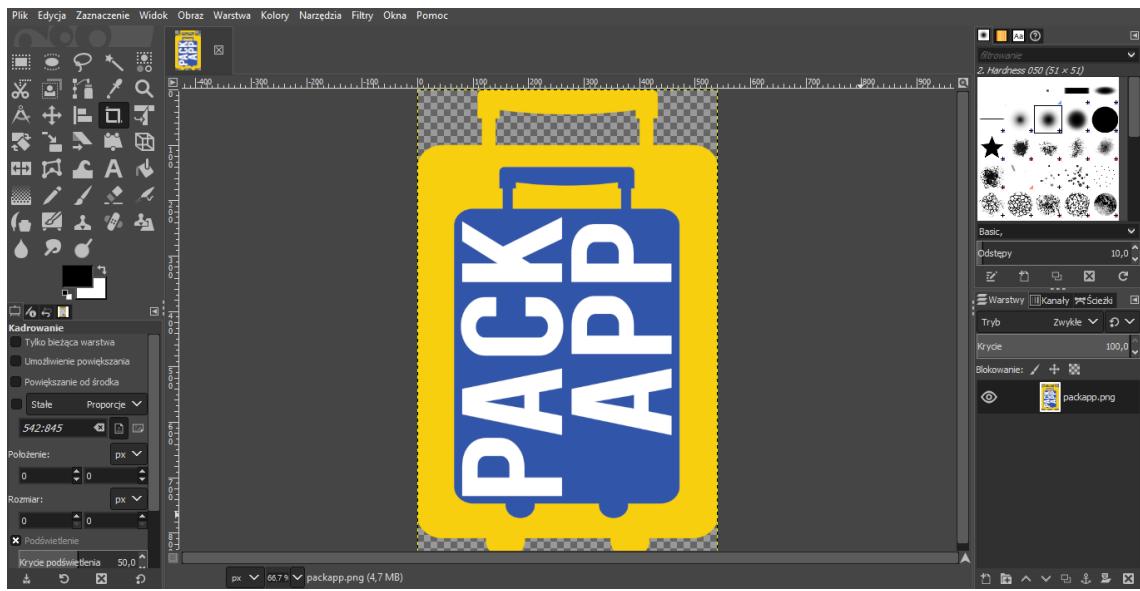


Rysunek 3.4: Zrzut ekranu prezentujący przykładowy diagram klas stworzony w StarUML [13]. Źródło: opracowanie własne

## 3.3 GIMP

GIMP (GNU Image Manipulation Program) jest darmowym programem służącym do tworzenia grafiki rozpowszechnionym na licencji GNU GPL. Pozwala na tworzenie prostych grafik czy logotypów. Jest również dobrą alternatywą dla płatnych programów takich jak np. Adobe Photoshop.

Mimo, że większość widoków w aplikacji stworzona została przy wykorzystaniu wbudowanych narzędzi Android Studio, zdecydowałem się stworzyć logotyp oraz ikonę aplikacji w programie GIMP (Rysunek 3.5).



Rysunek 3.5: Zrzut ekranu prezentujący widok programu GIMP. Źródło: opracowanie własne

## 4 Opis i realizacja aplikacji

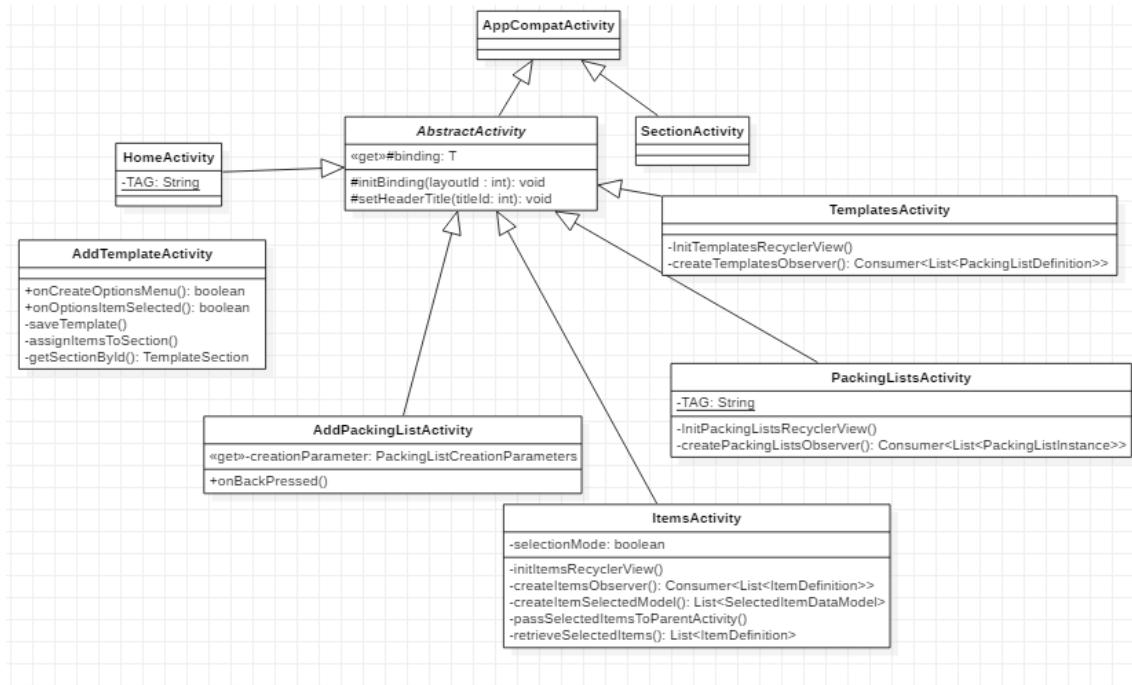
### 4.1 Założenia

Stworzono aplikację wspomagającą proces organizowania podróży, która umożliwia:

- Sprawdzenie pogody w miejscu docelowym za pomocą Weather API
- Tworzenie, edycja oraz zapis listy rzeczy do spakowania wraz z priorytetyzowaniem przedmiotów najważniejszych
- Tworzenie, edycja oraz zapis szablonów list rzeczy do spakowania, szablony zawierają sekcje do których przypisać można poszczególne przedmioty
- Tworzenie, edycja oraz zapis własnych przedmiotów do spakowania
- Zliczanie wagi netto naszego bagaża (istotne przy lotach pasażerskich)

### 4.2 Klasy aktywności

Wszystkie aktywności, które mają miejsce w aplikacji odbywają się dzięki klasom znajdującym się w pakiecie `activity`. W pakiecie tym zastosowana została technika *Data binding*, czyli stworzenie mechanizmu synchronizacji między źródłem danych, a ich odbiorcą. Sposób realizacji Data bindingu został przedstawiony za pomocą diagramu UML przedstawiającego klasy aktywności na rysunku 4.1.



Rysunek 4.1: Zrzut ekranu prezentujący diagram UML klas aktywności. Źródło: opracowanie własne

## 4.3 Funkcjonalności i interakcje

### 4.3.1 Główny ekran

Na ekranie głównym znajdują się logo aplikacji oraz 4 interaktywne przyciski. Przycisk *Lists* przenosi nas do ekranu, na którym widzimy dostępne listy (Rysunek 4.3), przycisk *Items* przenosi nas do ekranu przedmiotów (Rysunek 4.5), przycisk *Templates* przenosi nas do ekranu szablonów (Rysunek 4.6), natomiast okrągły przycisk z symbolem plusa, pozwala nam na stworzenie nowej podróży, przenosząc nas do ekranu generowania listy rzeczy do spakowania (Rysunek 4.7).

## Home



LISTS

ITEMS

TEMPLATES

+

Rysunek 4.2: Ekran główny aplikacji PackApp. Źródło: opracowanie własne

Widok ekranu głównego został stworzony w pliku XML - `activity_home.xml`, treść tego pliku została zwarta w Listingu 4.1.

Listing 4.1: Fragment widoku ekranu głównego aplikacji

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <data>

        <variable
            name="handler"
            type="pl.lodz.p.edu.handlers.ClickHandler" />

        <variable
            name="itemsHandler"
            type="pl.lodz.p.edu.handlers.ClickHandler" />

        <variable
            name="packingListsHandler"
            type="pl.lodz.p.edu.handlers.ClickHandler" />

        <variable
            name="templatesHandler"
            type="pl.lodz.p.edu.handlers.ClickHandler" />

    </data>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".activities.HomeActivity"
        android:background="@color/text_icons">

        <com.google.android.material.button.MaterialButton
            android:id="@+id/listsButton"
            android:layout_width="@dimen/home_button_width"
            android:layout_height="@dimen/home_button_height"
            android:layout_marginStart="8dp"
            android:layout_marginTop="32dp"
            android:layout_marginEnd="8dp"
            android:text="@string/lists_label"
            style="@style/RoundedButton"
```

```

        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageView"
        android:onClick="@{() -> packingListsHandler.onClick()}" />

    (...)

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="198dp"
        android:layout_height="320dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="32dp"
        android:layout_marginEnd="8dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/packing_list_logo" />

</androidx.constraintlayout.widget.ConstraintLayout>

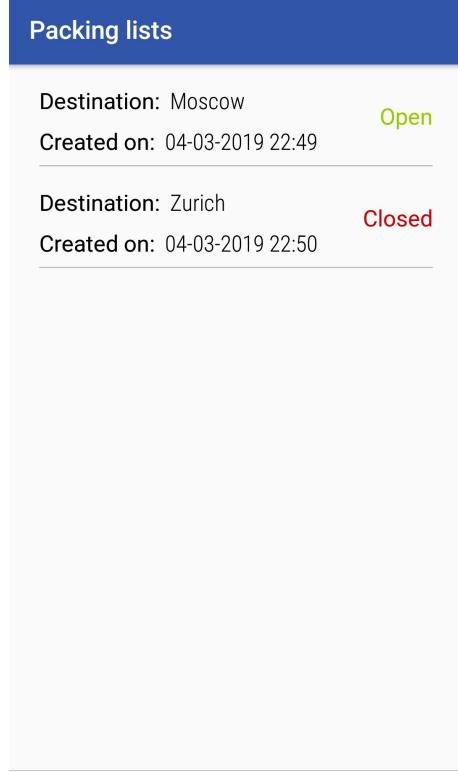
</layout>

```

Na załączonym listingu widać dodawanie przycisku *Lists* (pozostałe przyciski dodawane są analogicznie, zmienia się tylko ich treść i handler) oraz widoczne jest jak zostało dołączone logo aplikacji. Jak widać każdy przycisk ma przypisany unikalny numer identyfikujący, co pozwala później używać referencji do niego. Poza tym przycisk posiada rozmiar, treść, umiejscowienie, ale najważniejszym elementem jest linia `android:onClick="@() -> packingListsHandler.onClick()"`, to dzięki niej wywoływany zostaje handler, a po naciśnięciu przycisku użytkownik jest przenoszony do kolejnego ekranu.

#### 4.3.2 Ekran list

Po wybraniu przycisku *Lists* na ekranie głównym aplikacji (Rysunek 4.2) użytkownik zostaje przeniesiony do ekranu, w którym może przejrzeć stworzone przez siebie listy (Rysunek 4.3).



Rysunek 4.3: Zrzut ekranu pokazujący listy użytkownika. Źródło: opracowanie własne

Na tym ekranie użytkownik widzi stworzone przez siebie listy w postaci okienek, w których zawarty jest cel podróży, data stworzenia listy oraz jej status. Gdy status listy opisany jest zielonym napisem *Open* oznacza to, że lista jest otwarta i można dalej zaznaczać spakowane produkty natomiast czerwony napis *Closed* oznacza, że lista jest zamknięta i nie można dokonywać już żadnych interakcji. (Rysunek 4.4). Przejście ze statusu otwartego na zamknięty odbywa się za pomocą naciśnięcia przycisku *Close* na ekranie listy o statusie otwartym. Wartym odnotowania jest, że w prawym dolnym rogu listy mamy widoczną liczbę, mówiącą ile waży zaznaczona przez nas część przedmiotów, a ile cały bagaż.

Generate packing list	
Destination: Moscow	
Created on: 04-03-2019 22:49	
Status: Open	
General	
Passport *	<input type="checkbox"/>
Money *	<input type="checkbox"/>
ID Card *	<input type="checkbox"/>
<hr/>	
Other	
Briefcase 5	<input type="checkbox"/>
For Rainy	<input type="checkbox"/>
Weight [kg]: 0.0/11.0	
<input type="button" value="CLOSE"/> <input type="button" value="SAVE"/>	

Generate packing list	
Destination: Zurich	
Created on: 04-03-2019 22:50	
Status: Closed	
Odzież	
Bike shoes	<input checked="" type="checkbox"/>
Running shoes	<input checked="" type="checkbox"/>
Basketball shoes	<input checked="" type="checkbox"/>
<hr/>	
Parasol	
Passport	<input type="checkbox"/>
Money	<input type="checkbox"/>
ID Card	<input type="checkbox"/>
<hr/>	
Weight [kg]: 3.0/6.0	

Rysunek 4.4: Zrzuty ekranu, prezentujące listę o statusie *Open* (po lewej), oraz *Closed* (po prawej). Źródło: opracowanie własne

#### 4.3.3 Ekran przedmiotów

Po wybraniu przycisku *Items* na ekranie głównym aplikacji (Rysunek 4.2) użytkownik zostaje przeniesiony do ekranu, w którym może przejrzeć dostępne w bazie danych przedmioty (Rysunek 4.5). W kolumnie *Name* widzimy nazwę przedmiotu, w kolumnie *Weight* jego wagę, natomiast kolumna *Activity* mówi nam do jakiego rodzaju aktywności jest przedmiot przypisany, co jest istotne przy tworzeniu listy.

Items
Name: Passport
Weight [kg]: 1.0
Activity: -
Name: Money
Weight [kg]: 1.0
Activity: -
Name: ID Card
Weight [kg]: 1.0
Activity: -
Name: Bike shoes
Weight [kg]: 1.0
Activity: Playing golf
Name: Running shoes
Weight [kg]: 1.0
Activity: Jogging



Rysunek 4.5: Zrzut ekranu, prezentujący ekran dostępnego w bazie danych przedmiotów. Źródło: opracowanie własne

Naciśnięcie okrągłego przycisku z symbolem plusa, pozwoli nam na dodanie nowego przedmiotu (Rozdział 4.3.6). Aplikacja pobiera dane na temat przedmiotów z klasy `ItemDefinition` (Listing 3.4) w pliku `single_item_layout.xml`(Listing 4.2), który jest widokiem pojedynczego przedmiotu, natomiast cała lista przedmiotów, wraz z przyciskiem dodawania przedmiotu zawarta jest w pliku `activity_items.xml` (Fragment w listingu 4.3).

Listing 4.2: Fragment widoku pojedynczego elementu na liście przedmiotów (reszta pól wyświetalana jest analogicznie)

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">
    android:id="@+id/layout">

    <data>

        <variable
            name="obj"
            type="pl.lodz.p.edu.database.entity.definitions.ItemDefinition" />

    </data>
```

```

<androidx.constraintlayout.widget.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="4dp"
    android:onClick="@{(v) -> handler.onClick()}"
    android:paddingStart="8dp"
    android:paddingEnd="8dp">

    <TextView
        android:id="@+id/single_item_name_label_tv"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_weight="0.30"
        android:fontFamily="sans-serif-condensed-medium"
        android:text="@string/name_label_2"
        android:textColor="@android:color/black"
        android:textSize="18sp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    (...)

    </androidx.constraintlayout.widget.ConstraintLayout>
</layout>

```

Listing 4.3: Fragment widoku ekranu przedmiotów dostępnych w bazie danych pokazujący wyświetlanie nazwy przedmiotu (reszta pól wyświetalana jest analogicznie)

```

<androidx.constraintlayout.widget.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/items_recycler_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

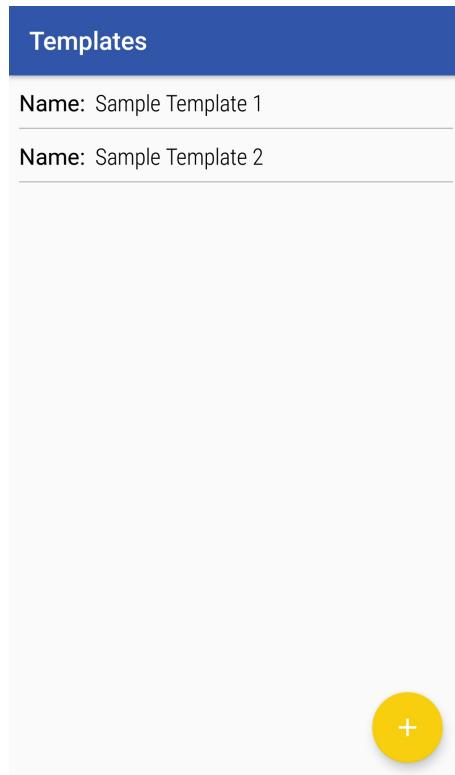
    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/floating_action_button"
        android:layout_width="wrap_content"

```

```
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|end"
        android:layout_margin="16dp"
        android:layout_marginEnd="16dp"
        android:layout_marginBottom="16dp"
        android:onClick="@{(v) -> handler.onClick()}"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:srcCompat="@drawable/ic_add_white_24dp" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

#### 4.3.4 Ekran szablonów

Po wybraniu przycisku *Templates* na ekranie głównym aplikacji (Rysunek 4.2) użytkownik zostaje przeniesiony do ekranu, w którym może zobaczyć stworzone przez siebie szablony (Rysunek 4.6).

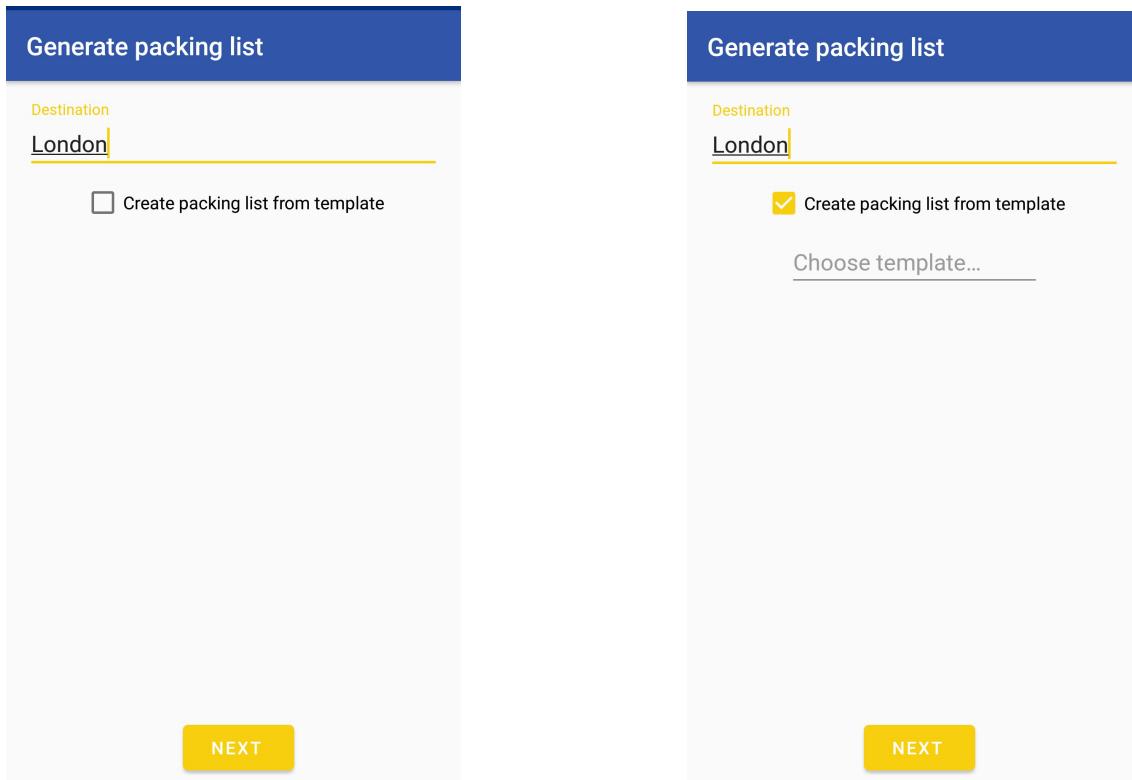


Rysunek 4.6: Zrzut ekranu, prezentujący ekran stworzonych przez użytkownika szablonów. Źródło: opracowanie własne.

Widok ten ogranicza się do pokazania listy dostępnych szablonów oraz analogicznie do poprzednich widoków posiada przycisk w prawym dolnym rogu umożliwiający przejście do ekranu tworzenia szablonu (Rozdział 4.3.8).

#### 4.3.5 Ekran generowania listy rzeczy do spakowania

Po wybraniu przycisku okrągłego przycisku z symbolem plusa na ekranie głównym aplikacji (Rysunek 4.2) użytkownik zostaje przeniesiony do ekranu, w którym może zacząć planować swoją podróż (Rysunek 4.7).



Rysunek 4.7: Zrzuty ekranu, prezentujące ekran wpisywania miejsca docelowego (po lewej) i ewentualne generowanie listy poprzez wybór szablonu (po prawej). Źródło: opracowanie własne

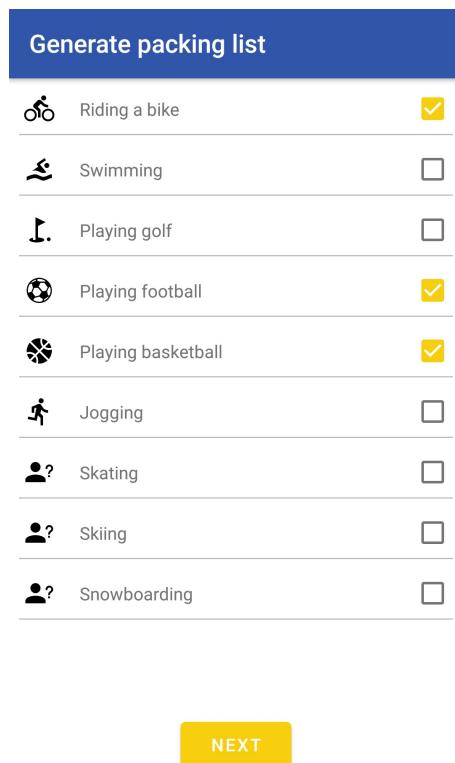
Pole tekstowe z wyborem szablonu automatycznie podpowiada nam nazwy szablonów, które użytkownik wcześniej stworzył, implementacja widoku dla takiego zachowania przedstawiona jest w listingu 4.4 pochodzący z pliku *fragment\_basic\_data\_packing\_list.xml*.

Listing 4.4: Fragment widoku pokazujący automatyczne wypełnianie pola wyboru szablonu

```
<AutoCompleteTextView  
    android:id="@+id/template_autocomplete_tv"  
    android:layout_width="200dp"  
    android:layout_height="wrap_content"  
    android:layout_marginStart="8dp"  
    android:layout_marginTop="8dp"
```

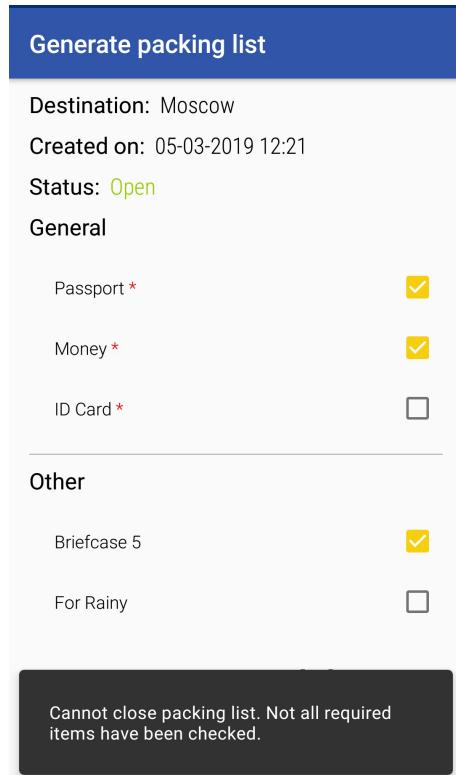
```
        android:layout_marginEnd="8dp"
        android: visibility = "invisible"
        android:hint="@string/choose_template"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/checkBox"
        android:text="@={converter.toString(params.template)}"/>
```

Następnym oknem, które pokazuje nam się podczas tworzenia naszej podróży, jest wybór aktywności, na podstawie których aplikacja wygeneruje nam przedmioty niezbędne do spakowania (Rysunek 4.8).



Rysunek 4.8: Zrzut ekranu, prezentujący możliwość wyboru aktywności. Źródło: opracowanie własne

Po wybraniu odpowiednich dla naszego wypoczynku aktywności, przechodzimy już do ekranu listy o statusie *Open* (Rysunek 4.4). Warto dodać, że priorytetyzacja elementów na liście, sprawia że nie możemy zmienić jej statusu na *Closed* dopóki nie zaznaczymy przedmiotów obowiązkowych, oznaczonych czerwoną gwiazdką (Rysunek 4.9).



Rysunek 4.9: Zrzut ekranu, prezentujący brak możliwości zamknięcia listy bez zaznaczenia przedmiotów obowiązkowych. Źródło: opracowanie własne

Za kontrolę priorytetu zaznaczenia przedmiotów najważniejszych odpowiada klasa `ClosePackingListHandler`, której najistotniejszy fragment widoczny jest w listingu 4.5.

Listing 4.5: Fragment kodu sprawdzającego czy elementy z nadanym priorytetem zostały odznaczone

```
public class ClosePackingListHandler extends AbstractPackingListHandler implements ClickHandler {

    public ClosePackingListHandler(CreatedPackingListFragment fragment) {
        super(fragment);
    }

    @Override
    public void onClick() {
        super.onClick();
        if (allRequiredItemsChecked()) {
            final ClosePackingListInstanceTask task = new ClosePackingListInstanceTask(
                fragment);
            task.execute(fragment.getPackingList());
        }
    }
}
```

```

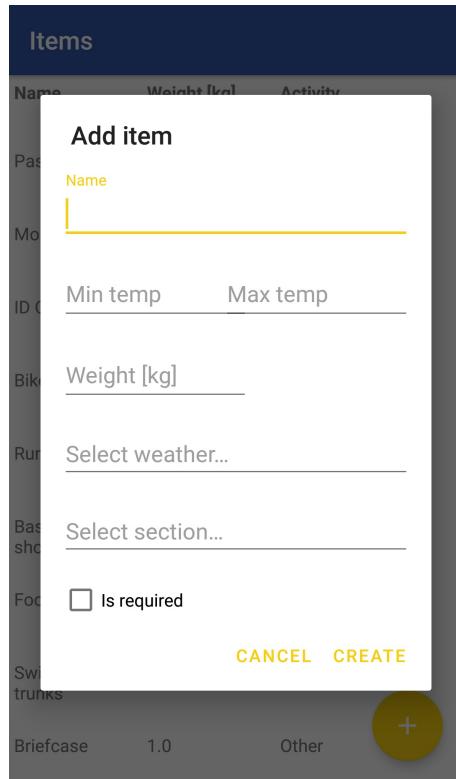
final FragmentActivity activity = fragment.getActivity();
if (activity != null) {
    activity . finish ();
}
else {
    Snackbar.make(fragment.getView(), R.string.required_items_not_checked_error,
        Snackbar.LENGTH_LONG).show();
}
}

private boolean allRequiredItemsChecked() {
    final PackingList packingList = fragment.getPackingList();
    for (Section section : packingList.getSections()) {
        for (Item item : section.getItems()) {
            if (item.getSectionItemInstance().isRequired() && !item.getInstance().
                isSelected ()) {
                return false;
            }
        }
    }
    return true;
}

```

#### 4.3.6 Dodawanie przedmiotu

Gdy na ekranie dostępnych w bazie danych przedmiotów (Rysunek 4.5) naciśniemy okrągły przycisk z symbolem plusa, zostaniemy przeniesieni do ekranu, w którym możemy do bazy danych dodać własny przedmiot (Rysunek 4.10).



Rysunek 4.10: Zrzut ekranu, prezentujący okno dodawania własnego przedmiotu do bazy danych. Źródło: opracowanie własne

Plikiem z widokiem tego okna jest plik *dialog\_add\_item.xml*(fragmenty w listingu 4.6). Okno zawiera pola tekstowe na nazwę oraz automatycznie uzupełniane pola pogody oraz aktywności. Ponadto pola w których możemy określić temperaturę w jakiej nasz przedmiot jest użyteczny oraz checkbox, który pozwala nam określić czy przedmiot zostanie dodany do listy tych obowiązkowych. W listingu 4.6 widzimy w jaki sposób zaimplementowane są okna z automatycznym wypełnianiem, checkbox oraz pole wagi, które posiada również konwerter na typ *Double* (Listing 4.7). Z kolei listing 4.8 przedstawia fragment klasy *ItemsActivity*, która odpowiedzialna jest za logikę dodawania przedmiotu.

Listing 4.6: Fragment kodu widoku okna dodawania przedmiotu

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <data>
        <import type="pl.lodz.p.edu.converters.DoubleConverter" />
        ...
    </data>
```

```

(...)

<com.google.android.material.textfield . TextInputEditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="number"
    android:text="@={DoubleConverter.toString(newItem.weight)}"
    android:singleLine="true"/>

(...)

<AutoCompleteTextView
    android:id="@+id/weather_autocomplete"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginTop="16dp"
    android:layout_marginEnd="16dp"
    android:hint="@string/select_weather_hint"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/weightInput"
    android:text="@={WeatherConverter.toString(newItem.weather)}"
    android:singleLine="true"/>

<AutoCompleteTextView
    android:id="@+id/activity_autocomplete"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginTop="16dp"
    android:layout_marginEnd="16dp"
    android:hint="@string/select_section_hint"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/weather_autocomplete"
    android:text="@={ActivityConverter.toString(newItem.activity)}"
    android:singleLine="true"/>

<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginTop="16dp"
    android:layout_marginEnd="16dp"
    android:checked="@{newItem.required}"
    android:text="@string/required_item_label"
    app:layout_constraintEnd_toEndOf="parent"

```

```

        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/activity_autocomplete" />

    </androidx.constraintlayout.widget.ConstraintLayout>
</layout>

```

Listing 4.7: Listing klasy DoubleConverter realizujące konwersję typu string na double (oraz konwersję na odwrót)

```

import androidx.databinding.InverseMethod;

public class DoubleConverter {

    private static final String DOT = ".";
    private static final String COMA = ",";
    private static final String EMPTY = "";

    @InverseMethod("toDouble")
    public static String toString(Double value) {
        return value == null ? EMPTY : value.toString();
    }

    public static Double toDouble(String value) {
        return Double.valueOf(value.replace(COMA, DOT));
    }

}

```

Listing 4.8: Fragment kodu klasy ItemsActivity

```

public class ItemsActivity extends AbstractActivity<ActivityItemsBinding> {

    private boolean selectionMode = false;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.initBinding(R.layout.activity_items);
        setHeaderTitle(R.string.items_title);

        final Intent intent = getIntent();
        if (intent != null) {

```

```

        this.selectionMode = intent.getBooleanExtra(ExtrasCodesEnum.
            SELECTION_MODE.name(), false);
    }

    this.binding.setHandler(new AddItemClickListener(this));
    initItemsRecyclerView();
}

@SuppressWarnings("CheckResult")
private void initItemsRecyclerView() {
    this.binding.itemsRecyclerView.setLayoutManager(new LinearLayoutManager(this));
    PackAssistantDatabase
        .getInstance(this)
        .itemDefinitionsDao()
        .getAll()
        .subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(createItemsObserver());
}

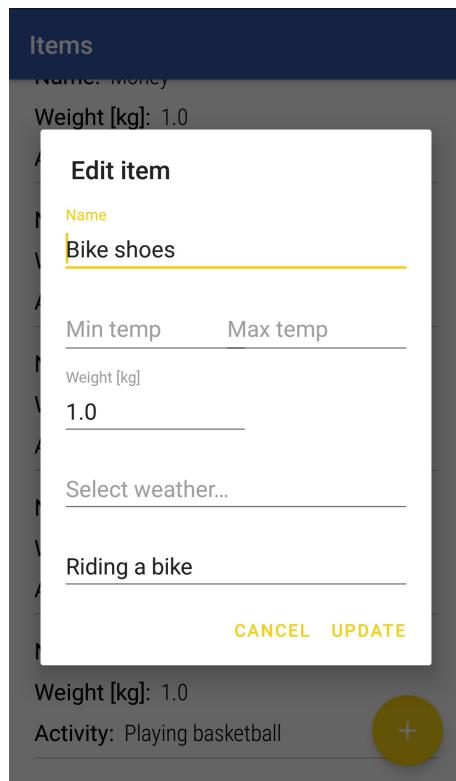
private Consumer<List<ItemDefinition>> createItemsObserver() {
    return new Consumer<List<ItemDefinition>>() {
        @Override
        public void accept(List<ItemDefinition> itemDefinitions) {
            if (selectionMode) {
                binding.itemsRecyclerView.setAdapter(new ViewAdapter<>(
                    createItemSelectedModel(itemDefinitions), R.layout.
                    single_item_selection_layout));
            } else {
                binding.itemsRecyclerView.setAdapter(new ViewAdapter<>(
                    itemDefinitions, R.layout.single_item_layout));
            }
        }
    };
}

private List<SelectedItemDataModel> createItemSelectedModel(List<ItemDefinition>
    itemDefinitions) {
    final List<SelectedItemDataModel> result = new ArrayList<>();
    for (ItemDefinition definition : itemDefinitions) {
        result.add(new SelectedItemDataModel(definition));
    }
    return result;
}

```

#### 4.3.7 Edycja przedmiotu

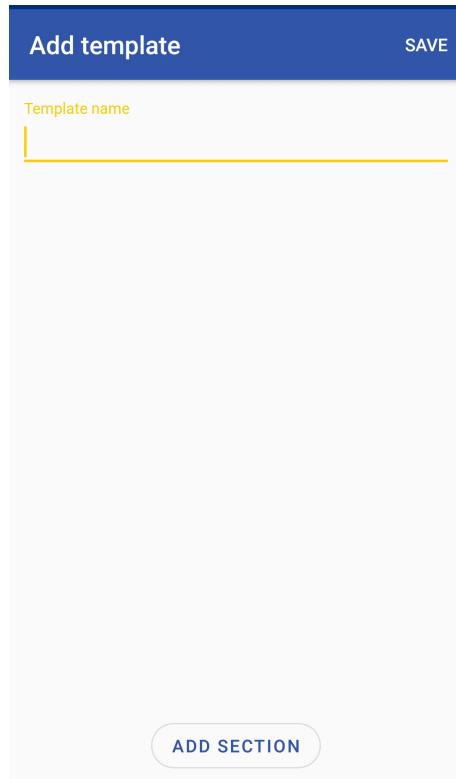
Ekran przedmiotów ponadto zawiera możliwość ich edycji. Po kliknięciu na któryś z przedmiotów dostępnych na liście (rysunek 4.5), zostajemy przeniesieni do ekranu edycji przedmiotu (rysunek 4.11). Okno to jest bardzo podobne do okna dodawania przedmiotu (rysunek 4.10), jednak zamiast przycisku *CREATE* mamy przycisk aktualizujący przedmiot *UPDATE*.



Rysunek 4.11: Zrzut ekranu, prezentujący okno edycji przedmiotu z bazy danych.  
Źródło: opracowanie własne

#### 4.3.8 Dodawanie szablonu

Gdy na ekranie listy stworzonych przez użytkownika szablonów (Rysunek 4.6) naciśniemy okrągły przycisk z symbolem plusa, zostaniemy przeniesieni do ekranu, w którym możemy dodać kolejny szablon do naszej bazy (Rysunek 4.12).



Rysunek 4.12: Zrzut ekranu, prezentujący okno dodawania własnego szablonu do bazy danych. Źródło: opracowanie własne

Logika dodawnia szablonu odbywa się w klasie `TemplatesActivity`(Listing 4.9).

Listing 4.9: Dodawanie szablonu w klasie `TemplatesActivity`

```
public class TemplatesActivity extends AbstractActivity<ActivityTemplatesBinding> {

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.initBinding(R.layout.activity_templates);
        setHeaderTitle(R.string.templates_title);

        binding.setHandler(new ViewTransitionClickHandler(this, AddTemplateActivity.class));
    }

    initTemplatesRecyclerView();
}

private void initTemplatesRecyclerView() {
    this.binding.templatesRecyclerView.setLayoutManager(new LinearLayoutManager(this));
}
final Disposable subscription = PackAssistantDatabase
```

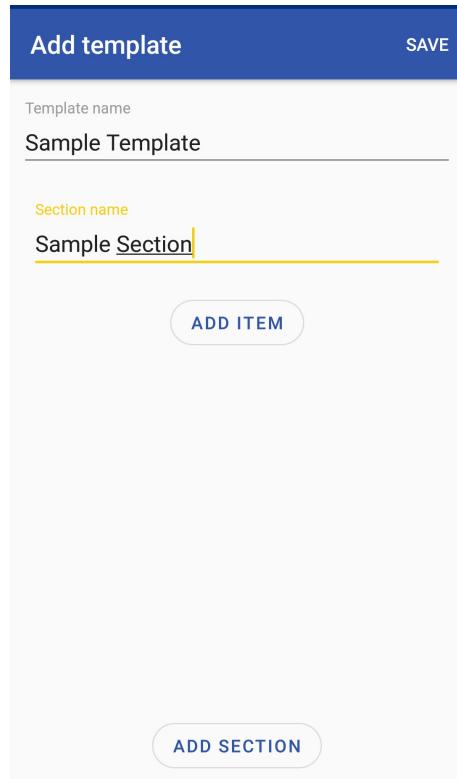
```

    .getInstance(this)
    .packingListDefinitionsDao()
    .getAll(true)
    .subscribeOn(Schedulers.io())
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(createTemplatesObserver());
}

private Consumer<List<PackingListDefinition>> createTemplatesObserver() {
    return new Consumer<List<PackingListDefinition>>() {
        @Override
        public void accept(List<PackingListDefinition> data) {
            binding.templatesRecyclerView.setAdapter(new ViewAdapter<>(data, R.
                layout.single_template_layout));
        }
    };
}
}

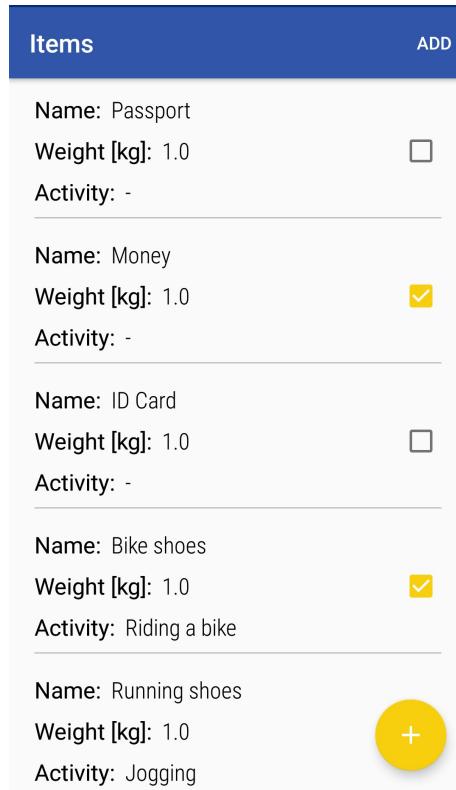
```

Na początku jedynym dostępnym polem tekstowym, jest pole nazwy szablonu. Po wpisaniu nazwy i naciśnięciu przycisku *Add Section*, będziemy mogli dodać sekcję, czyli podział przedmiotów naszego szablonu (np. spodnie i koszulka - sekcja odzież). Pojawi nam się pole tekstowe, w którym możemy wpisać nazwę sekcji (Rysunek 4.13).



Rysunek 4.13: Zrzut ekranu, prezentujący okno dodawania sekcji do szablonu. Źródło: opracowanie własne

Następnie do każdej sekcji, możemy dodać przedmioty z naszej bazy danych, klikając przycisk *Add Item*, który poprowadzi nas do bardzo podobnego okna jak na rysunku 4.5, jednak okno dodawania przedmiotu do sekcji posiada w prawym górnym rogu przycisk *Add* (Rysunek 4.14).



Rysunek 4.14: Zrzut ekranu, prezentujący okno dodawania przedmiotu do sekcji szablonu. Źródło: opracowanie własne

Dodawanie przedmiotu do sekcji odbywa się w klasie `AddTemplateActivity` za pomocą metody `assignItemsToSection(Listing 4.10)`.

Listing 4.10: Metoda `assignItemsToSection()` dodająca przedmiot do sekcji

```

private void assignItemsToSection(final Intent data) {
    List<ItemDefinition> items = (List<ItemDefinition>) data.getSerializableExtra(
        ExtrasCodesEnum.ITEMS.name());
    int sectionId = data.getIntExtra(ExtrasCodesEnum.SECTION_ID.name(), -1);

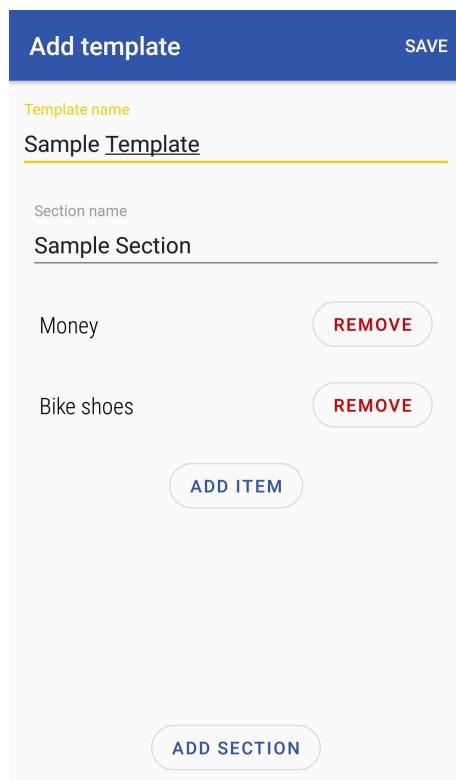
    if (sectionId >= 0 && items != null && items.size() > 0) {
        TemplateSection section = getSectionById(sectionId);
        if (section == null) {
            return;
        }

        section.getItems().addAll(mapToTemplateSectionItem(items));
        binding.templateSectionsRecyclerView.setAdapter(new
            TemplateSectionViewAdapter(template.getSections(), AddTemplateActivity.
            this));
        binding.getRoot().invalidate();
    }
}

```

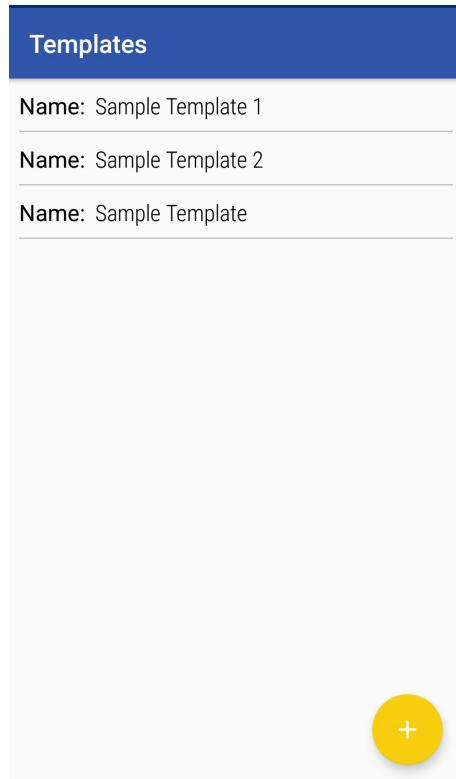
}

Następuje powrót do okna dodawania szablonu, w którym możemy jeszcze edytować sekcje oraz przedmioty do niej dodane (Rysunek 4.15).



Rysunek 4.15: Zrzut ekranu, prezentujący możliwość edycji szablonu przed zapisem.  
Źródło: opracowanie własne

Na końcu po naciśnięciu przycisku *Save* następuje zapis naszego szablonu do bazy danych oraz aktualizacja listy szablonów (Rysunek 4.16)



Rysunek 4.16: Zrzut ekranu, prezentujący zaktualizowaną o nową pozycję listę szablonów. Źródło: opracowanie własne

Listing 4.11: Metoda zapisująca szablony saveTemplate() znajdująca się w klasie AddTemplateActivity

```
private void saveTemplate() {  
    final SaveTemplateAsyncTask task = new SaveTemplateAsyncTask(this);  
    task.execute(template);  
    finish();  
}
```

## 5 Podsumowanie

Celem pracy było zaprojektowanie i implementacja aplikacji wspomagającej organizację podróży na urządzenia z systemem Android, rozszerzając jednocześnie rozwiązania alternatywne, dostępne w Sklepie Google Play. Cel pracy został osiągnięty. Dane aplikacji zapisywane są w pamięci urządzenia, używając relacyjnej bazy danych SQL, za pomocą wbudowanej w system Android biblioteki SQLite. Ponadto rekordy bazy danych możemy sami zmieniać, dodając właśnie przedmioty, listy oraz szablony przedmiotów do spakowania. Stworzone rozwiązanie wyróżnia się możliwością dodawania własnych szablonów pakowania, zliczania wagi bagażu co pozwoli uniknąć problemów z limitami na lotniskach oraz bardzo istotna opcja priorytetyzacji przedmiotów na liście, czyli braku możliwości zamknięcia listy, bez odnaczenia pozycji obowiązkowych np. paszportu, pieniędzy czy dowodu osobistego. Żeby zweryfikować pozyteczność stworzonego rozwiązania, postanowiłem porównać je z istniejącymi aplikacjami omówionymi w rozdziale 2. Porównanie zostało dokonane, biorąc pod uwagę następujące kryteria: liczba dostępnych przedmiotów, szybkość tworzenia listy pakowania, wygląd interfejsu użytkownika, liczba atrybutów przedmiotu, intuicyjność interfejsu użytkownika, zarządzanie szablonami przedmiotów do spakowania, zliczanie wagi bagażu, priorytetyzacja przedmiotów na liście. Porównanie zostało zrelizowane w formie ankiety, pięciu użytkowników zostało poproszonych o ocenę w skali 0-5 poszczególnego kryterium (maksymalna ocena wynosi 25, przy braku funkcjonalności 0 pkt.), następnie oceny zostały zsumowane i umieszczone w odpowiedniej komórce Tabeli 5.1.

Tabela 5.1: Porównanie stworzonego rozwiązania z dostępnymi w Sklepie Google Play

Kryterium	PackKing	PackPoint	MLC	PackApp
Liczba dostępnych przedmiotów	19	22	25	15
Liczba atrybutów przedmiotu	16	5	20	25
Szybkość tworzenia listy pakowania	18	19	23	21
Wygląd UI	18	22	5	20
Intuicyjność UI	18	19	11	20
Zarządzanie szablonami przedmiotów do spakowania	0	16	0	20
Zliczanie wagi bagażu	0	0	0	20
Przeglądanie stworzonych list	23	17	10	20
Priorytetyzacja przedmiotów na liście	0	0	0	20
Funkcja pogody w miejscu docelowym	10	25	0	15

Celem pracy było stworzenie aplikacji, która rozszerza istniejące rozwiązania o przydatne funkcjonalności i jak wynika z ankiety cel został osiągnięty. Użytkownicy pozytywnie ocenili zaimplementowane przezem mnie funkcjonalności i zgodnie przyznali, że z nimi aplikacja mogłaby konkurować z komercyjnymi rozwiązaniami w Sklepie Google Play. Taki wynik ankiety oznacza prawidłowe zdiagnozowanie problemów istniejących rozwiązań. Aplikacja oferuje unikalne funkcjonalności, jedna posiada wady, które wymagają zminimalizowania.

Największą według użytkowników wadą, jest brak możliwości eksportu listy do pliku PDF lub zewnętrznego serwisu typu DropBox. Użytkownicy uznali również, że baza danych jest zbyt mała, warto byłoby dodać gotowe przedmioty, aktywności oraz szablony. Istotną wadą jest również brak API do obsługiwanego lokalizacji, przez co użytkownik musi ręcznie wpisać prawidłową miejscowością docelową, proces wpisywania jest walidowany tylko przez zatwierdzenie nazwy przez użytkownika. Problem ten nie wynika z kłopotów implementacyjnych, ponieważ wdrożenie do programu takiego API jest w Android Studio czynnością prostą, jednak nie było

żadnych darmowych rozwiązań, które realizowałyby tę funkcję. Poza tym problem wystąpił również przy API obsługującym prognozę pogody udostępnionym przez Yahoo! [14], ponieważ prognoza jest znana tylko na najbliższe 9 dni i nie ma możliwości ręcznego wpisania daty okresu, który nas interesuje. Płatne rozwiązania np. te od Google mają takie możliwości i warto byłoby z nich skorzystać. To głównie w tych kategoriach użytkownicy podkreślili przewagę rozwiązań komercyjnych. W celu zminimalizowania wad, aplikację należałyby rozbudować.

Pierwszym elementem ulepszenia mojego rozwiązania będzie zwiększenie bazy danych według sugestii użytkowników, oraz jej typu, czyli wykorzystanie bazy zdalnej, zamiast lokalnej. Wraz z dodanym systemem obsługującym social media, pozwoli to użytkownikom na udostępnianie swoich zaplanowanych podróży, czy szablonów przedmiotów do spakowania, a co za tym idzie popularność aplikacji zwiększy się.

Najmocniejszą z unikalnych funkcjonalności mojej aplikacji jest zliczanie wagi bagażu. Zliczanie wagi polega na dodaniu wagi poszczególnych produktów, można to rozszerzyć o uwzględnienie wagi bagaża i informację jakiej wagi bagaż dopuszcza dany przewoźnik. W aplikacji również zostanie dodane komercyjne API obsługujące lokalizację użytkownika, dzięki takiemu systemowi można rozbudować aplikację o miejsca warte odwiedzenia w danym regionie: restauracje, zabytki, pomniki przyrody itp.

Priorytetyzacja również przypadła ankietowanym do gustu. Polega ona na braku możliwości zamknięcia listy pakowania, bez oznaczenia obowiązkowych produktów takich jak np. paszport lub dowód osobisty. Funkcjonalność tę można rozszerzyć dodając wartość priorytetu dla poszczególnych przedmiotów, co decydowałoby o kolejności pakowania przedmiotów i pozwoliłoby upewnić się, że nie zapominimy niczego istotnego.

## 6 Literatura

- [1] World Bank Open Data. Free and open access to global development data.  
<https://data.worldbank.org/> (dostęp 9.2.2019 r.)
- [2] Flightradar24 Live Air Traffic.  
<https://www.flightradar24.com/data/statistics> (dostęp 6.2.2019 r.)
- [3] Ministerstwo Sportu i Turystyki  
<https://www.msit.gov.pl/pl/turystyka/badania-rynku-turystycz/statystyka-komunikaty-i/7855,Charakterystyka-podrozy-mieszkancow-Polski-w-2017-r.html> (dostęp 9.2.2019 r.)
- [4] WeAreSocial UK – Digital in 2018: World's internet users pass the 4 billion mark  
<https://wearesocial.com/uk/blog/2018/01/global-digital-report-2018>  
(dostęp 9.02.2019 r.)
- [5] Spider's Web - Android i iOS rządzą.  
<https://www.spidersweb.pl/2018/02/android-ios-udzialy-rynkowe.html>  
(dostęp 9.02.2019 r.)
- [6] Aplikacja wspomagająca planowanie podróży PackKing  
<https://play.google.com/store/apps/details?id=com.adotis.packking&hl=pl> (dostęp 9.02.2019 r.)
- [7] Aplikacja wspomagająca planowanie podróży PackPoint travel packing list app  
<https://www.packpnt.com/>
- [8] Aplikacja wspomagająca pakowanie walizki My Luggage Checklist  
[https://play.google.com/store/apps/details?id=world.easysolution.myluggagechecklist&hl=en\\_US](https://play.google.com/store/apps/details?id=world.easysolution.myluggagechecklist&hl=en_US)
- [9] K. Pelgrims, *Gradle for Android. Automate the build process for your Android projects with Gradle*, Packt Publishing, 2015.
- [10] Dokumentacja systemu Android  
[developer.android.com](https://developer.android.com)
- [11] Dokumentacja biblioteki SQLite  
[sqlite.org](https://sqlite.org)
- [12] Opis biblioteki Room Persistence Library  
<https://developer.android.com/training/data-storage/room/#java>

[13] Dokumentacja programu StarUML

<https://docs.staruml.io/>

[14] Dokumentacja Yahoo Weather API

<https://developer.yahoo.com/weather/>

## 7 Spis rysunków

2.1	Zrzuty ekranu, prezentujące aplikację PackKing. Ekran główny (po lewej), tworzenie listy rzeczy do spakowania (po środku) wybieranie aktywności w miejscu docelowym (po prawej). Źródło: opracowanie własne . . . . .	5
2.2	Zrzuty ekranu, prezentujące aplikację PackPoint. Ekran tworzenia wycieczki (po lewej), wybieranie aktywności (po środku) wygenerowana lista rzeczy do spakowania(po prawej). Źródło: opracowanie własne . . . . .	6
2.3	Zrzuty ekranu, prezentujące aplikację My Luggage Checklist. Ekran startowy(po lewej), dostępne przedmioty (po środku) pasek boczny menu(po prawej). Źródło: opracowanie własne . . . . .	7
3.1	Zrzut ekranu prezentujący widok projektu aplikacji w Android Studio. Źródło: opracowanie własne . . . . .	9
3.2	Diagram prezentujący związek między komponentami w bibliotece Rōm. Źródło: [12] . . . . .	11
3.3	Diagram UML przedstawiający komponent Entities. Źródło: opracowanie własne w programie StarUML [13] . . . . .	14
3.4	Zrzut ekranu prezentujący przykładowy diagram klas stworzony w StarUML [13]. Źródło: opracowanie własne . . . . .	17
3.5	Zrzut ekranu prezentujący widok programu GIMP. Źródło: opracowanie własne . . . . .	18
4.1	Zrzut ekranu prezentujący diagram UML klas aktywności. Źródło: opracowanie własne . . . . .	20
4.2	Ekran główny aplikacji PackApp. Źródło: opracowanie własne . . . . .	21
4.3	Zrzut ekranu pokazujący listy użytkownika. Źródło: opracowanie własne . . . . .	24
4.4	Zrzuty ekranu, prezentujące listę o statusie <i>Open</i> (po lewej), oraz <i>Closed</i> (po prawej). Źródło: opracowanie własne . . . . .	25
4.5	Zrzut ekranu, prezentujący ekran dostępnych w bazie danych przedmiotów. Źródło: opracowanie własne . . . . .	26
4.6	Zrzut ekranu, prezentujący ekran stworzonych przez użytkownika szablonów. Źródło: opracowanie własne. . . . .	28
4.7	Zrzuty ekranu, prezentujące ekran wpisywania miejsca docelowego (po lewej) i ewentualne generowanie listy poprzez wybór szablonu (po prawej). Źródło: opracowanie własne . . . . .	29
4.8	Zrzut ekranu, prezentujący możliwość wyboru aktywności. Źródło: opracowanie własne . . . . .	30

4.9 Zrzut ekranu, prezentujący brak możliwości zamknięcia listy bez zaznaczenia przedmiotów obowiązkowych. Źródło: opracowanie własne . . . . .	31
4.10 Zrzut ekranu, prezentujący okno dodawania własnego przedmiotu do bazy danych. Źródło: opracowanie własne . . . . .	33
4.11 Zrzut ekranu, prezentujący okno edycji przedmiotu z bazy danych. Źródło: opracowanie własne . . . . .	37
4.12 Zrzut ekranu, prezentujący okno dodawania własnego szablonu do bazy danych. Źródło: opracowanie własne . . . . .	38
4.13 Zrzut ekranu, prezentujący okno dodawania sekcji do szablonu. Źródło: opracowanie własne . . . . .	40
4.14 Zrzut ekranu, prezentujący okno dodawania przedmiotu do sekcji szablonu. Źródło: opracowanie własne . . . . .	41
4.15 Zrzut ekranu, prezentujący możliwość edycji szablonu przed zapisem. Źródło: opracowanie własne . . . . .	42
4.16 Zrzut ekranu, prezentujący zaktualizowaną o nową pozycję listę szablonów. Źródło: opracowanie własne . . . . .	43

## **8 Spis tabel**

2.1 Porównanie dostępnych funkcji najpopularniejszych aplikacji wspomagających planowanie podróży . . . . .	8
5.1 Porównanie stworzonego rozwiązania z dostępnymi w Sklepie Google Play . . . . .	45

## 9 Spis listingów

3.1	Definicja klasy PackAssistantDatabase . . . . .	11
3.2	Abstrakcyjne metody oraz użycie wzorca singletonu w klasie PackAssistantDatabase . . . . .	12
3.3	Metoda <i>buildDatabase</i> w klasie PackAssistantDatabase . . . . .	13
3.4	Konstruktor klasy ItemDefinition oraz jej statyczna metoda <i>populateData()</i> . . . . .	14
3.5	Przykładowa kwerenda SQL z adnotacją @Query użyta w interfejsie oznaczonym adnotacją Dao . . . . .	15
3.6	Listing interfejsu ItemDefinitionsDao . . . . .	16
4.1	Fragment widoku ekranu głównego aplikacji . . . . .	22
4.2	Fragment widoku pojedynczego elementu na liście przedmiotów (reszta pól wyświetalana jest analogicznie) . . . . .	26
4.3	Fragment widoku ekranu przedmiotów dostępnych w bazie danych pokazujący wyświetlanie nazwy przedmiotu (reszta pól wyświetalana jest analogicznie) . . . . .	27
4.4	Fragment widoku pokazujący automatyczne wypełnianie pola wyboru szablonu . . . . .	29
4.5	Fragment kodu sprawdzającego czy elementy z nadanym priorytetem zostały odznaczone . . . . .	31
4.6	Fragment kodu widoku okna dodawania przedmiotu . . . . .	33
4.7	Listing klasy DoubleConverter realizujące konwersję typu string na double (oraz konwersję na odwrotnie) . . . . .	35
4.8	Fragment kodu klasy ItemsActivity . . . . .	35
4.9	Dodawanie szablonu w klasie TemplatesActivity . . . . .	38
4.10	Metoda assignItemsToSection() dodająca przedmiot do sekcji . . . . .	41
4.11	Metoda zapisująca szablon saveTemplate() znajdująca się w klasie AddTemplateActivity . . . . .	43