Kurs programowania aplikacji mobilnych
Sprawozdanie: Aplikacja modowa do katalogowania i losowania zestawów ubrań

Data: 24/04/2022
Ewa Namysł, Adrianna Pilawa
Informatyka stosowana, III rok
link do repozytorium: https://github.com/namysl/wirtualna-szafa-android-app

1. Założenia:

Głównym przeznaczeniem aplikacji jest możliwość katalogowania oraz losowania zestawów ubrań, natomiast jako podstawowe funkcjonalności aplikacji wybrano:

- dodawanie zdjęcia,
- usuwanie zdjęcia,
- losowanie zestawów ubrań,
- galeria zdjęć,
- dodawanie tagów do zdjęć.

Aplikacja mobilna ma być napisana w Javie na platformę Android.

2. Funkcjonalności obecne w tej wersji aplikacji:

Użytkownik ma możliwość dodania zdjęcia z galerii lub zrobienia zdjęcia bezpośrednio z poziomu aplikacji. Przy zapisywaniu zdjęcia należy dodać odpowiedni tag oraz kolor elementu ubioru, które wybierane są z listy rozwijalnej z predefiniowanymi wartościami. Ponadto aplikację zabezpieczono przed zapisaniem pustego pola lub sytuacją, kiedy nie dodano zdjęcia. W takim przypadku wyskakuje pop up ze stosowną informacją.

Użytkownik ma możliwość przeglądania zdjęć w przewijalnej horyzontalnej galerii, w której znajdują się wszystkie dodane przez niego elementy. Pod każdym zdjęciem znajduje się przycisk *USUŃ ELEMENT*, dzięki któremu może usunąć wybrany element. Jeśli nie dodano jeszcze ani jednego elementu, w galerii pojawi się stosowna informacja.

Użytkownik ma możliwość filtrowania dodanych przez niego elementów, a następnie przeglądania wyników. Może to zrobić wybierając tag, kolor lub oba pola. Poruszanie się po kolejnych elementów umożliwiają dwa guziki.

Użytkownik ma możliwość przeglądania wylosowanych zestawów z dodanych elementów. Aby zobaczyć nowy zestaw, należy kliknąć *LOSUJ PONOWNIE*. Jeśli użytkownik przejrzał wszystkie zestawy, na ekranie pojawi się pop up z informacją, a możliwe kombinacje ubrań będą wyświetlane od początku. Ponadto użytkownik ma możliwość zapisania wybranego zestawu.

Użytkownik może przeglądać i usuwać zapisane zestawy. Podobnie jak w przypadku filtrowania ubrań, poruszanie się umożliwiają dwa przyciski. Jeśli nie zapisano żadnego zestawu, pojawia się informacja o tym.

Użytkownik ma możliwość założenia konta oraz logowania, niestety rozwinięcie funkcjonalności związanej z API nie zostało ukończone.

Ponadto użytkownik może zmienić kolorystykę aplikacji. Dostępny jest jasny oraz ciemny motyw.

3. Harmonogram prac:

	Ewa Namysł	Adrianna Pilawa
Tydzień 1	• Rozpoczęcie pracy nad GUI.	• Rozpoczęcie pracy nad funkcją dodającą zdjęcia.
Tydzień 2	• Rozbudowa GUI, • praca nad uzyskaniem uprawnień dla aparatu oraz galerii.	• Rozpoczęcie pracy nad API.
Tydzień 3	• Dalsza rozbudowa GUI, • ukończono funkcję dla uprawnień oraz funkcję dodającą zdjęcia, • stworzono pierwszą wersję galerii, • jasny/ciemny motyw aplikacji, • zapisywanie zdjęć w pamięci wewnętrznej.	• Kontynuowanie pracy nad API.
Tydzień 4 oraz 5	• Poprawki GUI, rozbudowa o listy rozwijalne, • stworzenie lokalnej bazy danych oraz implementacja, • dodanie w galerii możliwości usuwania zdjęć, • dodanie możliwości filtrowania po tagach i kolorach, • funkcja losująca zestawy, • funkcja zapisująca zestawy.	• Dalsza praca nad API, • implementacja API, • funkcja logowania i rejestracji.

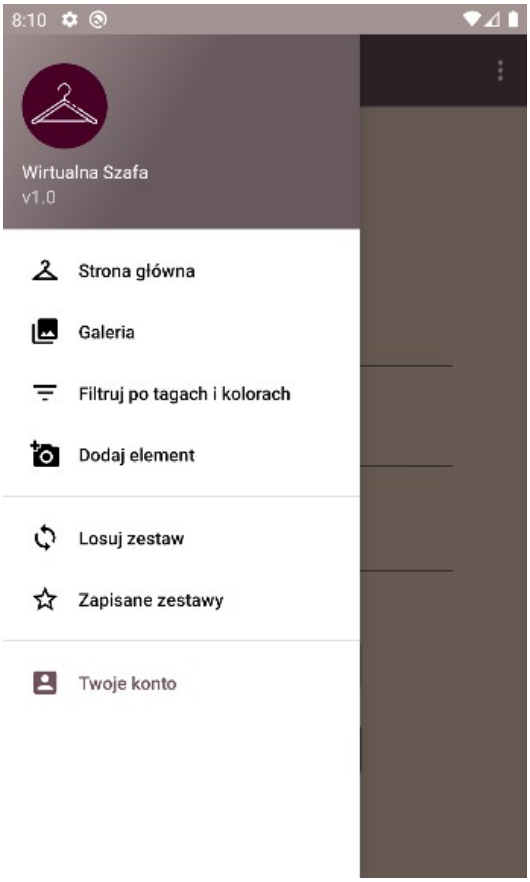
4. Wygląd aplikacji:

Widok strony głównej

jasny motyw

ciemny motyw

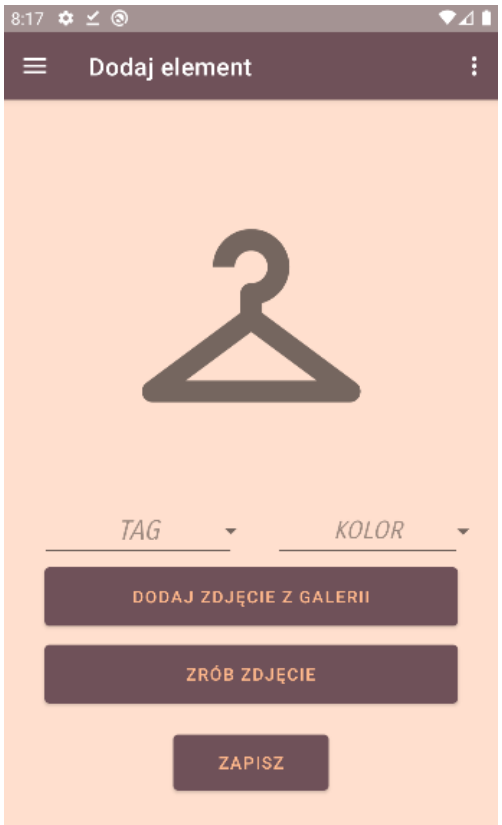
Widok panelu nawigacyjnego

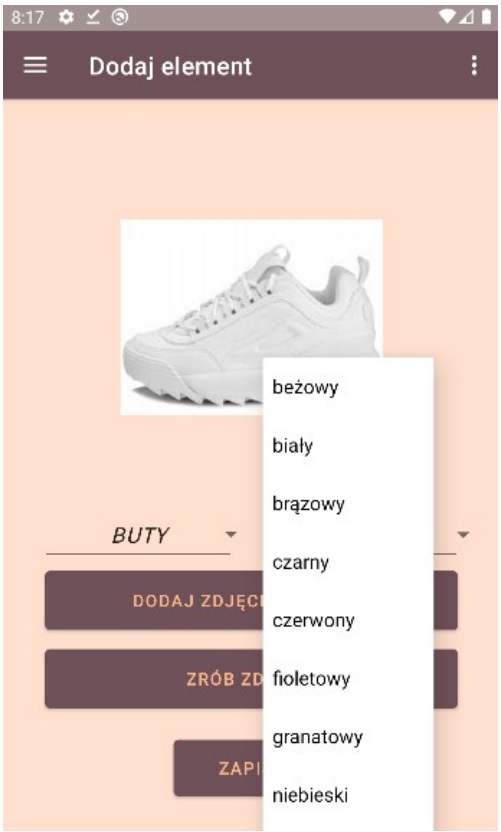


Widok galerii

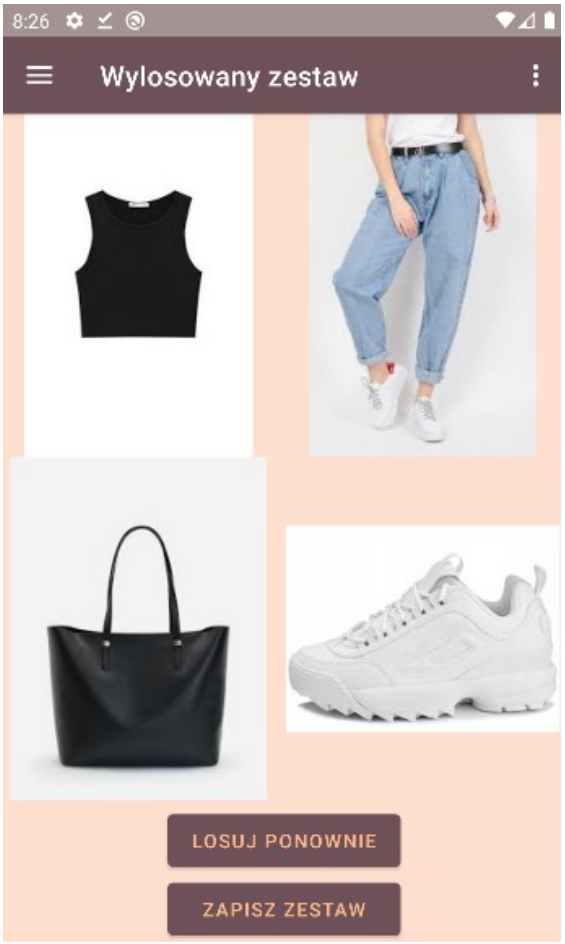
Filtrowanie po tagach i kolorach

Dodawanie elementu

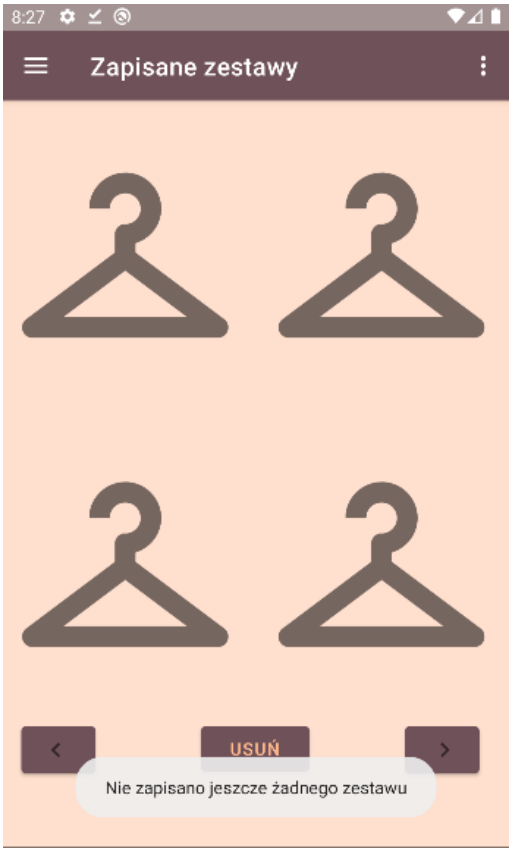




Losowanie zestawów



Widok zapisanych zestawów
(w tym przypadku przedstawienie sytuacji, w której nie zapisano żadnego)



Widok logowania/rejestracji

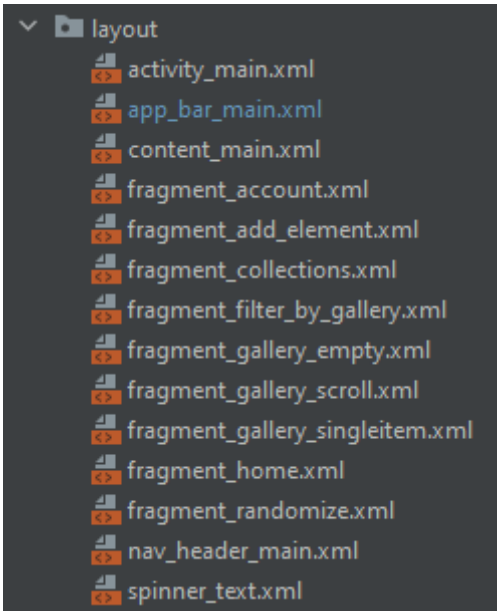
5. Wykorzystane języki, funkcje oraz zewnętrzne biblioteki:

Aplikacja napisana została na platformę Android w języku Java. Dodatkowo wykorzystano możliwości bibliotek takich jak Retrofit, Room do stworzenia lokalnej bazy danych oraz Google Guava do losowania (a w zasadzie tworzenia produktu kartezjańskiego z obiektów bazy danych, a następnie ich przetasowania przy pomocy funkcji z pakietu Collections).

Z kolei API zostało napisane w języku PHP za pomocą Framework Laravel 8. Znajdują się w nim metody: rejestracji, logowania, dodawania nowych zdjęć ubrań przez użytkownika, jak również metody umożliwiające zwrócenie użytkownika, jego zdjęć ubrań oraz tagi danych zdjęć.

6. Opis działania najważniejszych funkcjonalności oraz fragmenty kodu:

Każda zakładka w panelu nawigacyjnym ma własny plik xml ze zdefiniowanymi polami tekstowymi, guzikami i innymi niezbędnymi elementami GUI.



Aby zakładki mogły znaleźć się w panelu nawigacyjnym należy je dodatkowo zdefiniować w pliku *mobile_navigation.xml* oraz dodać do *AppBarConfiguration.Builder* w *MainActivity.java* jak poniżej:

```
mAppBarConfiguration = new AppBarConfiguration.Builder(
        R.id.nav_home, R.id.nav_gallery, R.id.nav_filter_by_gallery, R.id.nav_add_element,
        R.id.nav_randomize, R.id.nav_collections, R.id.nav_account)
        .setOpenableLayout(drawer)
        .build();
```

Każda z zakładek musi wywołać metodę *onCreateView*, która rozpoczyna proces tworzenia widoku. Tam wywołujemy elementy, które zdefiniowaliśmy w plikach xml.

6.1. Zakładka *Twoje konto*:

W zakładce tej można stworzyć konto lub zalogować się.

Cała komunikacja aplikacji z API została zaimplementowana przy użyciu biblioteki Retrofit, która zmienia HTTP API w interfejs Javy. Za pomocą GsonBuilder() przetwarzane są Gsonowe zapytania w obiekty.

W projekcie została użyta też RxJava do zarządzania wątkami, aby umożliwić pracę w tle w czasie użytkowania aplikacji i obserwację odpowiedzi.

Za pomocą wyrażeń regularnych sprawdzana jest również poprawność adresu mailowego.

```
public void login(String username, String password) {
    Disposable request = remoteService.login(username, password)
            .doOnSubscribe(disposable -> loading.postValue(true))
            .doOnTerminate(() -> loading.postValue(false))
            .subscribe(token -> {
                if(token != null && !token.isEmpty()) {
                    authorized.postValue(token);
                }
            }, throwable -> {
                authorized.postValue(null);
            });

    compositeDisposable.add(request);
}
```

funkcja obsługująca logowanie

```
public class RemoteService {

    private static final String BASE_URL = "http://adrinna-pilawa-314-a-2022-04.int.heag.live/api/";
    private final RemoteServiceInterface client;

    public RemoteService() {
        OkHttpClient.Builder builder = new OkHttpClient.Builder();

        // Add the interceptor to OkHttpClient
        HttpLoggingInterceptor httpLoggingInterceptor = new HttpLoggingInterceptor();
        httpLoggingInterceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
        builder.networkInterceptors().add(httpLoggingInterceptor);
        //builder.networkInterceptors().add(interceptor)
        OkHttpClient okHttpClient = builder.build();

        Gson gson = new GsonBuilder()
                .setLenient()
                .create();

        Retrofit retrofit = (new Retrofit.Builder())
                .baseUrl(BASE_URL)
                .addConverterFactory(GsonConverterFactory.create(gson))
                .addCallAdapterFactory(RxJava2CallAdapterFactory.create())
                .client(okHttpClient)
                .build();

        client = retrofit.create(RemoteServiceInterface.class);
    }
```

część klasy obsługującej API

6.2. Zakładka *Dodaj element:*

Listy rozwijalne (spinnery) umożliwiają wybór odpowiednich opcji.

Dzięki

```
spinner_color = rootView.findViewById(R.id.spinner_color);
ArrayAdapter<CharSequence> staticAdapter_color = ArrayAdapter.createFromResource(
        rootView.getContext(),
        R.array.add_color, //array of strings
        R.layout.spinner_text); //layout

staticAdapter_color.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinner_color.setAdapter(staticAdapter_color);
spinner_color.setSelection( position: 14,  animate: true);

spinner_color.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
        color_editT.setText((String) parent.getItemAtPosition(position));
    }
```

setOnItemSelectedListener wybrana przez użytkownika wartość przekazywana jest dalej do pola tekstowego, a następnie przesyłana ze zdjęciem do bazy danych.

Jeśli chodzi o wczytywanie lub robienie zdjęcia, konieczne było stworzenie metod *checkAndRequestPermission, onActivityResult* oraz *onRequestPermissionsResult,* które sprawdzają uprawnienia przyznane aplikacji. Jeśli aplikacja jest uruchamiana po raz pierwszy, to przy wybieraniu zdjęcia lub otwarciu w aplikacji aparatu wyskakuje pop up i prosi o przyznanie uprawnień.

Jeśli to przebiegło bez problemów, zdjęcie jest zapisywane jako plik w pamięci wewnętrznej, a następnie przesyłane asynchronicznie jako nowy obiekt do lokalnej bazy *WardrobeDB.*

6.3 Baza danych*:*

W aplikacji wykorzystano bibliotekę Room, która umożliwia stworzenie lokalnej bazy danych SQLite. Baza ma dwie tabele – *WardrobeDB* oraz *SavedCollectionsDB.*

Pierwsza służy do zapisywania nowych dodanych elementów oraz posiada kolumny *path*, *tag*, *color*. *Path* to ścieżka do zdjęcia w pamięci wewnętrznej. W *WardrobeDAO* zapisane są zapytania SQL oraz podstawowe metody (insert, delete, update).

Tabela

```
@Dao
public interface WardrobeDAO{
    //show all items in default manner
    @Query("SELECT * FROM WardrobeDB")
    List<WardrobeDB> getAll();

    //show all items in descending order by id, so new items first
    @Query("SELECT * FROM WardrobeDB ORDER BY id DESC ")
    List<WardrobeDB> getAllDesc();

    //only show items with given tag, e.g. shoes
    @Query("SELECT * FROM WardrobeDB WHERE tag=:tag")
    List<WardrobeDB> getClothesByTag(String tag);

    //only show items with given color, e.g. black
    @Query("SELECT * FROM WardrobeDB WHERE color=:color")
    List<WardrobeDB> getClothesByColor(String color);

    //tags and colors
    @Query("SELECT * FROM WardrobeDB WHERE tag=:tag AND color=:color")
    List<WardrobeDB> getClothesByTagAndColor(String tag, String color);
```

zapytania SQL pierwszej tabeli

SavedCollectionsDB służy do zapisania losowych zestawów wybranych przez użytkownika, kolumnami są cztery ścieżki do zdjęć zestawu. Podobnie jak w poprzednim przypadku, w *SavedCollectionsDAO* zapisane są zapytania SQL.

Sama baza danych wraz z tabelami zdefiniowana jest w *AppDB*, a obsługiwana jest przez instancję *ClientDB.*

6.4 Filtrowanie po tagach, zapisane kolekcje i/lub kolorach oraz galeria:

Jeśli chodzi o filtrowanie, to również użyto spinnerów, podobnie jak przy dodawaniu elementów. Wybrane wartości wykorzystywane są w zapytaniu SQL, a następnie zwracana jest lista z obiektami. Z obiektów za pomocą gettera dostajemy ścieżkę do zdjęcia, dzięki czemu możemy je wyświetlić.

```
class LoadFromDB extends AsyncTask<Void, Void, List<WardrobeDB>> {
    @Override
    protected List<WardrobeDB> doInBackground(Void... voids) {
        //retrieve data from DB
        WardrobeDAO dao = ClientDB.getInstance(getContext()).getAppDatabase().wardrobeDAO();
        found_clothes = dao.getClothesByTagAndColor(tag_editT.getText().toString(),
                                                    color_editT.getText().toString());

        return found_clothes;
    }
```

klasa wysyłająca zapytanie (tag+kolor), zwracająca listę obiektów

W przypadku galerii działa to podobnie, to z tą różnicą, że zwracane są wszystkie obiekty z bazy, które następnie w pętli dodawane są do kolejnych wywołań *imageView* i *TextView*, aby możliwe było horyzontalne scrollowanie. Możliwe jest także usuwanie – zdjęcie usuwane jest z bazy oraz z pamięci wewnętrznej.

Wygląda to podobnie w przypadku zapisanych zestawów z tabeli *SavedCollectionsDB,* z tą różnicą, że usuwanie zestawu usuwa go tylko z tabeli *SavedCollectionsDB,* natomiast indywidualnie elementy nadal istnieją w *WardrobeDB* oraz pamięci wewnętrznej.

6.5 Losowanie zestawów:

Wszystkie obiekty w tabeli *WardrobeDB* dodawane są do listy. W metodzie pomocniczej *populate_and_shuffle_list* tworzona jest dodatkowa lista, wypełniana liczbami od 0 do rozmiaru produktu kartezjańskiego (funkcja *cartesianProduct* z biblioteki Google Guava) wszystkich obiektów z tabeli. Następnie elementy są mieszane. Dzięki temu użycie modułu Random odbywa się tylko dwa razy – przy wyświetleniu zestawów po raz pierwszy oraz przy ponownym wczytaniu zestawów, jeśli użytkownik obejrzał już wszystkie.

```
private void populate_and_shuffle_list(List <Integer> list_to_populate, int picked_random){
    for(int i=0; i<cartesian_product_size; i++){
        list_to_populate.add(i);
    }
    list_to_populate.remove(picked_random);
    Collections.shuffle(list_to_populate);
}
```

Poza tymi przypadkami, kolejne zestawy wyświetlane są w kolejności z przemieszanej listy.