Creating Data-Driven Mobile Applications

COS80019

ASSIGNMENT 11.2D – Script for the presentation

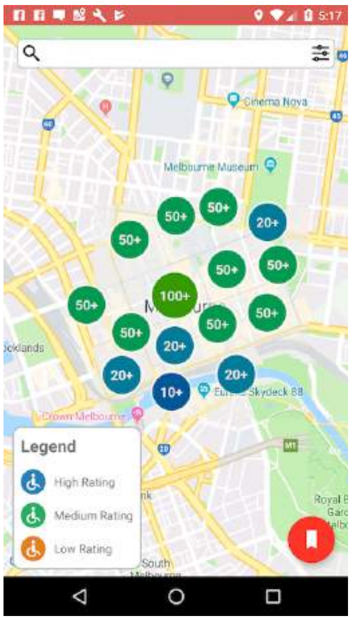
Accessibility Map in Melbourne

Student name: Duy Thuc Pham

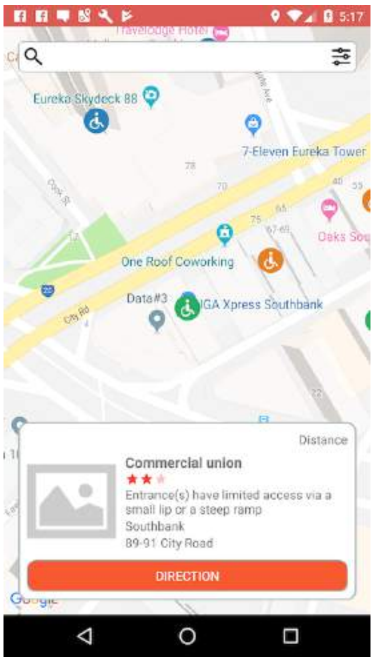
Student number: 101767225

App Functionalities

This application includes 5 main features: view accessible places, search accessible places, filter the places, bookmark places and get direction to the place with Google Map. The app starts with the Map screen.

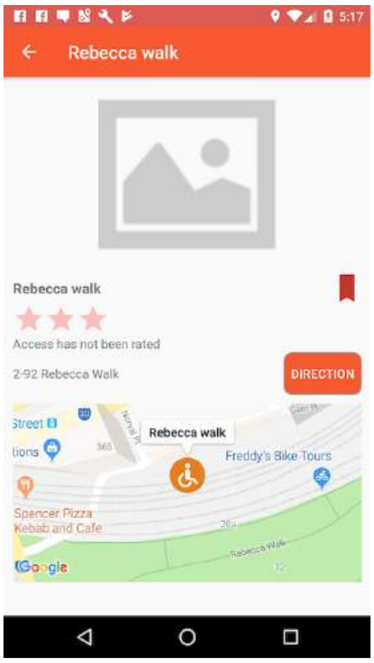
Map Screen	 Cinema Nova Melbourne Museum 50+ 20+ 100+ 10+ Docklands Crown Melbourne Eureka Skydeck 88 Legend High Rating Medium Rating Low Rating South Melbourne

The application asks the user for allowing the permission to access their GPS location so that the user is able to see accessible places around their current location. In this application, I have used the ACCESS_FINE_LOCATION to request the GPS location of the user. The map displays accessible places in 1km by default. The user can change the preferences to what they want in the filter screen. The map legend on the bottom left is used to explain the information of marker in this application. The user can go to Search screen, Filter screen and Bookmark screen in the map screen. Also, the by selecting the specific marker, the user can see the information. In the map screen, I have used the google map to display accessible places as markers. Also, in order to improve the user experience, the cluster view has been applied so that it helps reduce the number of markers displaying in the map.

Detail Window Screen	

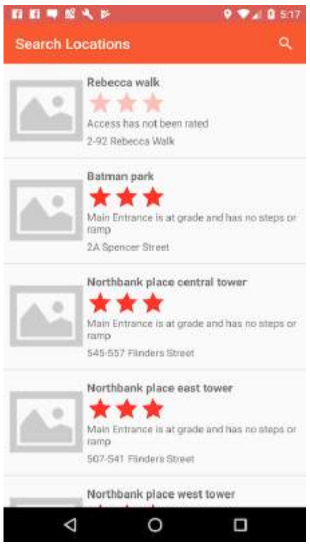
Distance
Commercial union
Entrance(s) have limited access via a small lip or a steep ramp
Southbank
89-91 City Road
DIRECTION

The information includes the name, rating, address, suburb and accessibility information of the place. The user is able to get the route to this location from their current location by select direction button. If the user wants to see the detail information of that place, he/she can simply click on the view to go to Detail screen.

Detail Location Screen	

In the detail screen, the user can see detail information of the place and able to bookmark this place by select bookmark button.

Regarding the search screen, the user can go to it by clicking on the search bar to enter this screen.

Search Screen	

5:17
Search Locations

Rebecca walk
Access has not been rated
2-92 Rebecca Walk

Batman park
Main Entrance is at grade and has no steps or ramp
2A Spencer Street

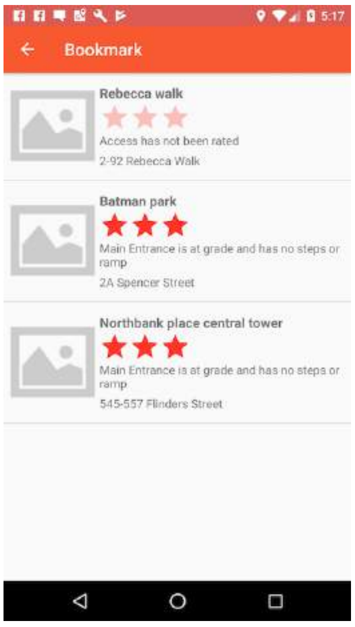
Northbank place central tower
Main Entrance is at grade and has no steps or ramp
545-557 Flinders Street

Northbank place east tower
Main Entrance is at grade and has no steps or ramp
507-541 Flinders Street

Northbank place west tower

In the search screen, it loads accessible places with pagination (10 places per page). The user can search the specific place that they want to go by name. They can refresh the screen by pull the screen to refresh. In addition, they can see the detail information of this screen by selecting on the cell.

All of the bookmark places are saved in the database and the user can see them by clicking on the bookmark button om Map screen. The user can remove the bookmark by simply swiping the row to delete it.

Bookmark Screen	

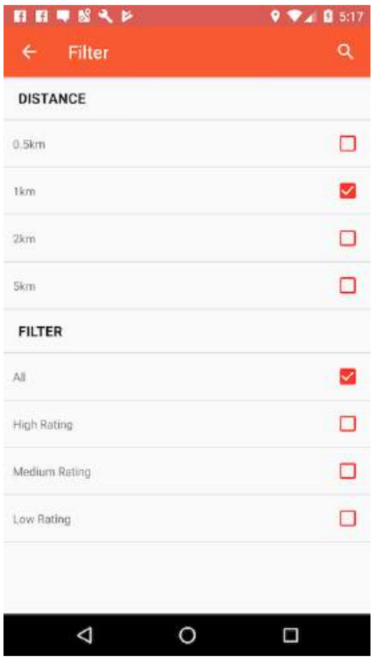
5:17
Bookmark

Rebecca walk
Access has not been rated
2-92 Rebecca Walk

Batman park
Main Entrance is at grade and has no steps or ramp
2A Spencer Street

Northbank place central tower
Main Entrance is at grade and has no steps or ramp
545-557 Flinders Street

To filter the application so that users can only see their places based on the ranking or radius, they can go to filter screen.

Filter Screen	 Filter DISTANCE 0.5km ☐ 1km ✓ 2km ☐ 5km ☐ FILTER All ✓ High Rating ☐ Medium Rating ☐ Low Rating ☐

These filter parameters will be saved in the user preferences of the application and used persistently to request APIs in the application.

APIs

In this application, I have used the Retrofit and the GSON to request API and parse the JSON to an object. The GSON is created by defining the JSON jey for the @SerializedName.

```java
public class Building implements Parcelable {
    @SerializedName("street_address")
    private String address;
    @SerializedName("lower_building_name")
    private String name;
    @SerializedName("block_id")
    private String blockId;
    @SerializedName("x_coordinate")
    private Double longitude;
    @SerializedName("y_coordinate")
    private Double latitude;
    @SerializedName("suburb")
    private String suburb;
    @SerializedName("accessibility_rating")
    private int rating;
    @SerializedName("accessibility_type")
    private String type;
    @SerializedName("accessibility_type_description")
    private String accessibilityDes;
}
```

Then to request the API and immediately parse the JSON in to the object, the Retrofit can define the object that response JSON will be parsed in the Interface which uses later to request the API

```
public interface RequestDataInterface {
    @Headers("X-App-Token: GxusT0nELv09s1GmHnb1osV1d")
    @GET("q8hp-qgps.json")
    Call<List<Building>> getBuildingInRange( @QueryMap(encoded = false) Map<String,
String> options);

    @Headers("X-App-Token: GxusT0nELv09s1GmHnb1osV1d")
    @GET("q8hp-qgps.json")
    Call<List<Building>> getBuildingByName( @QueryMap(encoded = false) Map<String,
String> options);
}
```

Then, I manage them in RequestAPIManager class and perform the request by simply invoking RetrofitClientInstance.getRetrofitInstance().create() method.

```
RequestDataInterface service =
RetrofitClientInstance.getRetrofitInstance().create(RequestDataInterface.class);
    Call<List<Building>> call = service.getBuildingInRange(data);
    call.enqueue(new Callback<List<Building>>() {
        @Override
        public void onResponse(Call<List<Building>> call, Response<List<Building>>
response) {
            if (response.isSuccessful()) {
                callback.onResponse(response.body());
            } else {
                String errorMessage =
ErrorHandlingClass.handlingRequestError(response.code());
                callback.onFailure(errorMessage);
            }
        }
        @Override
        public void onFailure(Call<List<Building>> call, Throwable t) {
            Log.e("Error", "network failure :( inform the user and possibly
retry");
            callback.onFailure("Network failure");
        }
    });
}
```

Database

In this application, Active Android ORM is used to deliver the effective database management. Active Android is quite easy to apply, perform query and also migrate the database. To use it, we should define the database name, version and model that we are using in this application in the Manifest.

```
<meta-data android:name="AA_DB_NAME" android:value="accessmap.db" />
<meta-data
    android:name="AA_DB_VERSION"
    android:value="1" />
<meta-data
    android:name="AA_MODELS"
    android:value="com.example.swinburne.accessiblitymap.Model.BuildingDA" />
```

Then, the table of it can be created by extends *Models* and defined the column information for the table.

```
@Table(name = "BuildingModel")
public class BuildingDA extends Model {
    @Column(name = "address")
    public String address;

    @Column(name = "name" , unique = true, onUniqueConflict =
Column.ConflictAction.REPLACE)
    public String name;

    @Column(name = "blockID")
    public int blockId;

    @Column(name = "longitude")
    public Double longitude;

    @Column(name = "latitude")
    public Double latitude;

    @Column(name = "suburb")
    public String suburb;

    @Column(name = "rating")
    public int rating;

    @Column(name = "type")
    public String type;

    @Column(name = "accessibilityDes")
    public String accessibilityDes;

    @Column(name = "imageURL")
    public String imageURL;
}
```

Finally, the table is managed by the DatabaseManager class which uses to centralize queries that used in this application. To perform the query, the user just calls the Select() method to perform query.

```
public static List<BuildingDA> getListBuilding() {
    return new Select()
            .from(BuildingDA.class)
            .execute();
}
```