
GitHub Username: j1c1m1b1

Shake Me Up

Description

An application that shows the user a random restaurant in a 5 km radius. The app also shows the reviews and relevant information about the places. Shake your phone and find a restaurant in seconds.

Intended User

People who are undecided about what to eat, and want to go to a place near them.

Features

- Stores application data
- Uses accelerometer to detect a shake motion
- Uses user location
- Vibrates

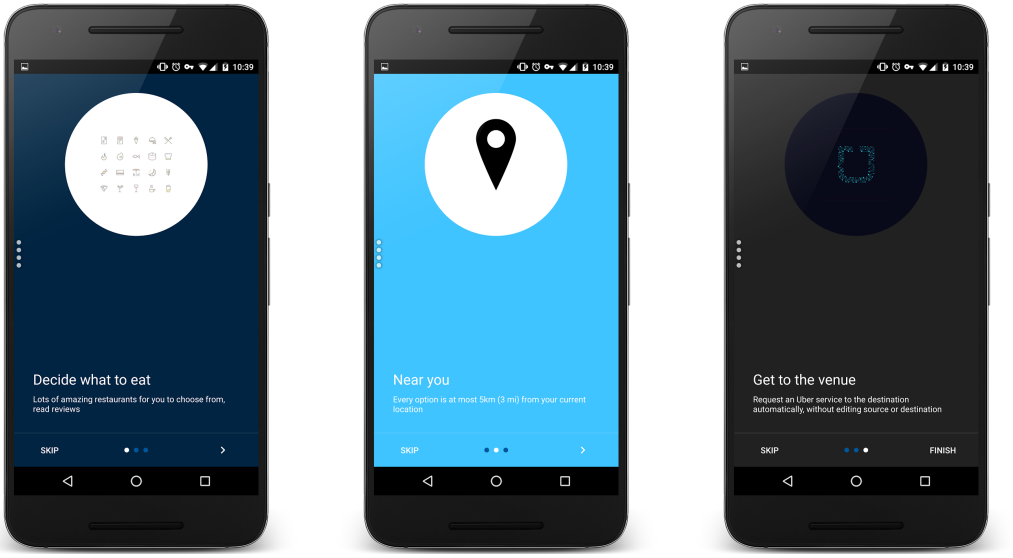
User Interface Mocks

These can be created by hand (take a photo of your drawings and insert them in this flow), or using a program like Photoshop or Balsamiq.

Splash Screen

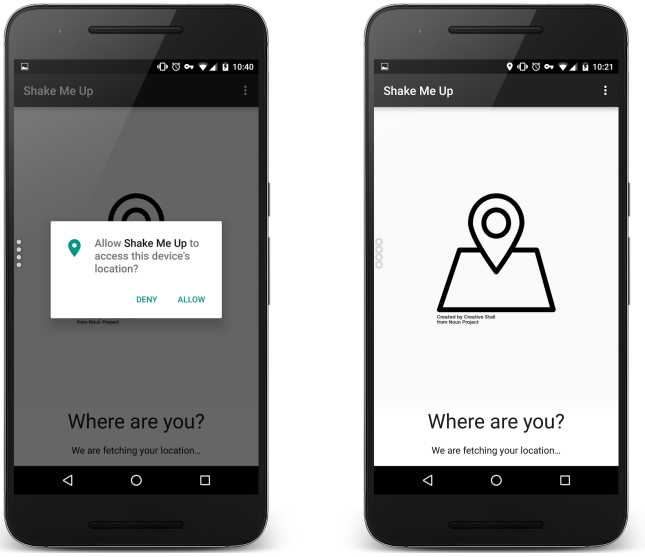
Splash screen of the app

Onboarding



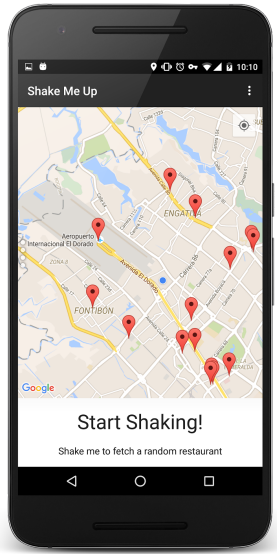
Screens that describe the features of the app.

Map Screen - Obtaining Location



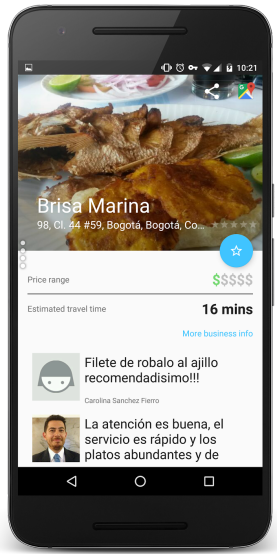
Screen that notifies the user that location is being obtained. (A permission dialog will be shown if the user's device OS version is after API 23: Marshmallow)

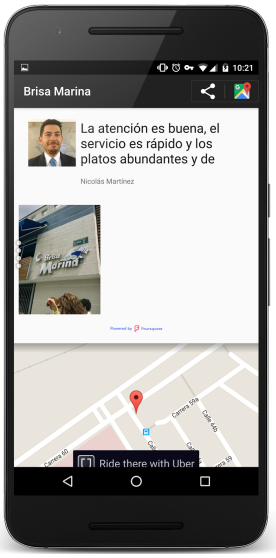
Map Screen - All Nearby Venues



After the location has been obtained a map is shown with markers to nearby locations

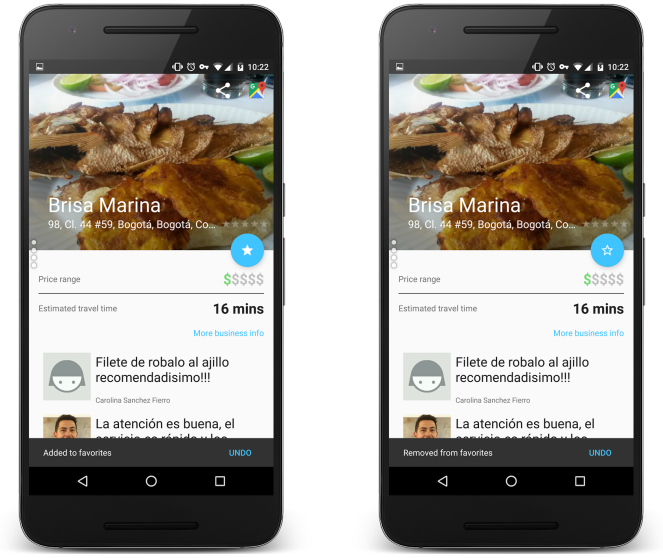
Place Screen





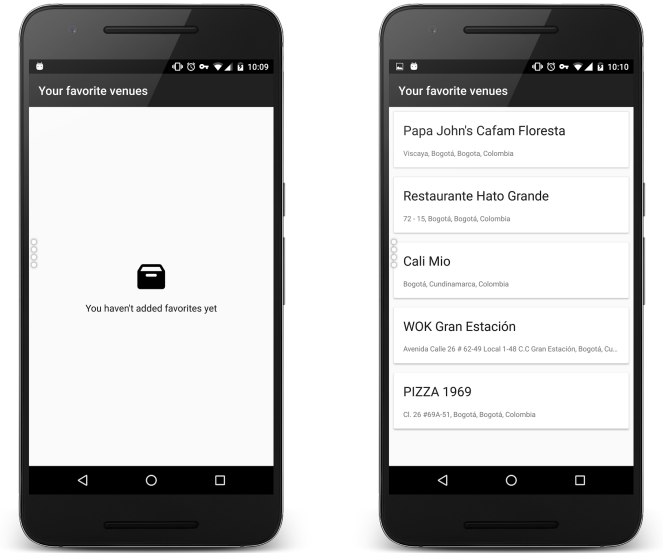
Scrolling activity that shows relevant information about a location. It provides a button to request an Uber ride to get the user to the venue.

Favorite Switch



Clicking the Floating Action button will display a Snackbar notifying that the venue was added or removed from the favorites list.

Favorites List



This screen shows the venues has added as favorites giving fast access to re-visit them

Key Considerations

How will your app handle data persistence?

- A boolean value indicating if the onboarding must not be shown again will be saved in the application's private Shared Preference

- I will build a ContentProvider to handle data persistence and save the user's favorite places, so they can access the information about them without transferring information from the network.

Corner cases

- When a request fails the user must be notified of the cause of the problem

Libraries

- Android design library to support the Material Design paradigm
- Maps and Location libraries needed to obtain the user's location and display a map of the locations of the places shown in the app
- Glide to handling and caching images.
- ORMlite to handle some of the data persistence.
- Uber SDK to show a "Ride with Uber button"
- OkHttp to handle requests

Next Steps: Required Tasks

This is the section where you can take the main features of your app (declared above) and decompose them into tangible technical tasks that you can complete incrementally until you have a finished app.

Task 1: Project Setup

Libraries Setup

1. Import the latest versions of the AppCompat and Design libraries
2. Import the latest versions of the Google Play Services maps and location libraries
3. Import Glide, OkHttp and Uber SDK

Google API Setup

1. Go to the Google Developers Console
2. Login with an existing Google account
3. Create a new Project
4. Follow the Guide in Create an API Key
5. Get an Additional Key for Google Places Web Service
6. Add both keys as **string resources** in the `strings.xml` file of your **values** res subfolder

Uber Setup

1. Register your app at Uber Developer Site
2. Obtain the **Uber client ID** and create a string resource with its value

Foursquare Setup

1. Go to Foursquare for Developers Site
2. Create a new application
3. Obtain the **Foursquare Client ID and Client Secret**
4. Create **string resources** for both values

Task 2: Implement UI for Each Activity and Fragment

1. Build the UI for the `SplashActivity`
2. Build the UI for the **Onboarding**
    a. Create the UI for `OnboardingActivity`
    b. Create the UI for the **3 Onboarding fragments**
        i. For the all the fragments name all the common elements with the **same** IDs
        ii. Add additional fragment layouts for **landscape orientation**
3. Create the UI for `MainActivity` and separate the activity file from the content
    a. Add a MapFragment to the layout.
4. Create the UI for the `PlaceActivity` separating the app bar layout from the specific content of the place layout
    a. Create an additional **`AppTheme.Dialog`** theme and make **`AppTheme.NoActionBar`** as the **parent**
    b. Create an additional `styles.xml` for the **`values-sw600dp`** directory, add the **`AppTheme.Dialog`** style to this file and set **`Theme.AppCompat.Light.Dialog`** as its **parent**
5. Create the UI for **`FavoritePlacesActivity`** and **favorite place item**.

Task 3: Declare the APIs Calls

Create the Requests class to handle all app requests through OkHttp APIs

- Google Maps requests:
  - Google Maps Distance Matrix API
  - Google Places API - Nearby Places
  - Google Places API - Place Details
  - Google Maps Geocoding API
- Foursquare calls:
  - Venues search
  - Venue details by ID

- Create a Requests class that contains all methods and fields to make all API calls
- **`Requests.java`**
  1. Add all paths, parameters names and generic values as constants
  2. Create a **private static field** of the **OkHttp** client and initialize it.
  3. Create a **private static generic method** that returns a URL.
  4. Create a **private static generic method** that creates a generic request.
  5. Create public static methods for every call to be made that passes its specific path and parameters. Each of this method will obtain a URL using the method created in **step 3** and calls the API with the method created at **step 4**.

- Create an **interface** that handles the failure or success of every call, to be added as parameter to the **`callApi`** method in the **Requests** class.
- Create a **`Parser`** class containing public static methods formats the responses obtained by the API calls and returns usable values to be used by the application components.

Task 4: Implement a Shake Detector

- Create the class and make it implement `SensorEventListener`.
- Create a **public inner interface** called `OnShakeListener` and declare a single method called `onShake()`.
- Create a **private field** of `OnShakeListener` and generate a setter for it.
- Override the methods of `SensorEventListener` and detect a shake event in the **`onSensorChanged()`** method.
  - If a shake event is detected call the **`onShake()`** method of your `OnShakeListener` field.

Task 5: Implement BaseActivity.java

Implement an Activity to handle the location features of the app.

- Create the file and make it extend from AppComptActivity.
- Make it implement from `ConnectionCallbacks`, `OnConnectionFailedListener` and `LocationListener`.
- Implement the methods of the declared interfaces.
- Add the run time permissions implementation to adjust for newest versions of Android
    - Create a **protected** method called **`onPermissionsSucceeded`** and add and call it once the user has accepted permissions
- Create a method called `getPlaces()` that makes an static call to the **`Requests`** class to obtain nearby places, if a response is obtained **parse** the returned answer with the `Parser` class, to **get the ID** of a **random place**.
- Implement the shake detector
    - Add a **private field** of the `ShakeDetector` class.
    - Create a method called `startUpAccelerometer()` that obtains the accelerometer sensor, initialises and sets an anonymous inner `OnShakeListener` implementation to the Shake Detector field.
    - Inside the implementation of the **`onShake()`** method, **make the device vibrate** for 200 milliseconds and **call `getPlaces()`**.

Task 6: Create the Main Activity

- Create a java class called `MainActivity`, extend `BaseActivity` and implement `OnMapReadyCallback`.
- Declare the Activity in the manifest.
- Create private fields of `MapFragment`.
- On the `onCreate(...)` method set its corresponding content view and initialize the UI fields.
    - Instantiate MapFragment.
- **Override** the `onPermissionsAccepted()` method of `BaseActivity`, call the **super** and then call the method `getMapAsync(...)` on the `MapFragment` field passing the **current instance** as a parameter.
- Create a private GoogleMap field
- Override onMapReady() method and set the `GoogleMap` parameter as your field.
- Override `onLocationChanged(...)` method of `BaseAcitivity` to center the `GoogleMap` camera to the user location.

Task 7: Create the On Boarding Activity

- Create a java class called `OnBoardingActivity`, extend `AppCompatActivity`.
- Declare the Activity in the manifest.
- Create fields for its UI components

- On the `onCreate(...)` method set its corresponding content view and initialize the UI fields.
- Create a click listener that:
    - Puts the value of **false** in a boolean called `FIRST` in the application's shared preferences.
    - Starts an Intent to the Main Activity and then finishes the current one.
- Obtain the app's shared preferences and instantiate a final boolean that indicates if it is the first time opening the app.
- Create a TimerTask that declares an Intent
    - Create a ternary operator upon a Class<?> variable that validates the boolean obtained from the shared preferences.
    - If the boolean is true set the Class<?> as OnBoardingActivity.class; set it as MainActivity.class on the contrary.

Task 8: Create the Splash Activity

- Create a java class called `SplashActivity`, extend `AppCompatActivity`.
- Declare the Activity in the manifest and add an intent filter with the `MAIN` action and the `LAUNCHER` category so it is the first shown when the app starts.
- On the `onCreate(...)` method set its corresponding content view.
- Obtain the **app's shared preferences** and instantiate a **final boolean** that indicates if it is the first time opening the app.
- Create a `TimerTask` that declares an Intent
    - Create a ternary operator upon a `Class<?>` variable that validates the boolean obtained from the shared preferences.
    - If the boolean is true set the `Class<?>` as `OnBoardingActivity.class`; set it as `MainActivity.class` on the contrary.
    - Create an **explicit `Intent`** passing the resulting `Class<?>` instance as the second parameter.
    - **Start an activity** with the created intent and **finish the current activity**.
- Create a `Timer` that schedules the `TimerTask` **after 3 seconds**.

Task 9: Create the Place Activity and modify Base Activity

- Create a java class called `PlaceActivity`, extend `BaseActivity`.
- Initialize its UI contents
- Create a private String field called `placeID`
- In the `onCreate()` method call `getIntent()` and initialize `placeID` with the `getString()` method of the Intent class.
- Create a menu layout resource for the activity and add an invisible menu item to share the place.

- Call the methods on the Requests class to obtain the place details from Google and Foursquare to update the UI contents.
    - Once the content is loaded update the share Menu Item to make it visible and so it has the Intent to the place location.
- Modify the method `getPlaces()` of the BaseActivity class:
    - Start `PlaceActivity` passing the place ID as an Intent extra.
    - Call the `finish()` method of Activity

Task 10: Create Place and Tip objects

- Create a class that contains the relevant information about a place.
- Create a class that contains the relevant information about a Tip.

Task 11: Create a ContentProvider and modify PlaceActivity

- Create a `Contract` **class**, that **describes the database schema**
- Create a `SQLite Helper` class, that **extends from** **`SQLiteOpenHelper`**
    - Add the `onCreate()` and `onUpgrade()` methods so they create and update the database accordingly.
    - **Access the contract class** to write the SQL statements to create necessary tables.
- Create a class that extends from `ContentProvider`
    - **Add a `URI matcher`** to **switch from different URIs sent to the `ContentResolver`**
    - **Implement the CRUD methods**, matching the correct `URIs`, so transactions are applied to the correct tables and columns.
- Create a custom `AsyncTaskLoader` that **deletes or inserts a place** from or to **the favorites table** and **returns a String** with a message **indicating the success or failure of the operation**.
    - Add **4 private fields** to the class: A **`URI`**, a **`Context`**, a **`Place`** and a **`boolean` called `delete`**.
    - Add 4 parameters of its **constructor**: a **`URI`**, a **`Context`**, a **`Place`** and a **`boolean`**. **Set the corresponding class fields** in the body of the method.
    - In the **`loadInBackground()`** method:
        - **Create a String** variable called **`resultMessage`** and set it to **a String resoruce** containing an error message.
        - Check the value of the **`delete` boolean**
        - If **`delete`'s value is equals to `true`** call the **`delete()`** method of the **content resolver instance** obtaining it from the context and **passing the URI as a parameter**.
            - If the transaction **was successful** set `resultMessage` **from string resources** indicating that the place was successfully deleted

        - On the contrary:
            - **Map the place parameter as `ContentValues`** using the Contract.
            - **Call the `insert()` method** passing the `URI` and a `ContentValues` parameters.
            - If the transaction was successful set `resultMessage` from string resources indicating that it was successfully inserted
        - **Return `resultMessage`**
- Update the `PlaceActivity` class
    - Update the **`activity_place` layout file(s)** adding an **anchored floating action button**
    - Add a **private boolean field** called **`isFavorite`** that will be **`true`** if the place was saved as a favorite or **`false`** on the contrary.
    - Make `PlaceActivity` **implement** `LoaderCallbacks`
        - **Create two IDs** to identify the database loaders
            - **`QUERY_LOADER_ID`** for the loader that checks if the current place is a favorite.
            - **`TRANSACTION_LOADER_ID`** for the loader that inserts or deletes a place from the favorites table.
        - In the **`onCreateLoader()`** method **check the ID parameter** so it matches `TRANSACTION_LOADER_ID`.
            - If it does, **check `isFavorite`**. If the place is favorite create a `URI` to delete the place from the favorites table; on the contrary create a `URI` that inserts it.
            - **Return a new instance** of your custom `AsyncTaskLoader`
        - **Else check if it matches `QUERY_LOADER_ID`** create a `URI` that **obtains a place** form the favorites table **matching the place ID**.
            - Return a new `CursorLoader` **instance** that performs the query.
        - In the **`onLoadFinished()` check the ID** of the `Loader` parameter
            - If it matches **`QUERY_LOADER_ID`** cast the `Object` parameter to a `Cursor`.
                - **Check if the cursor is `null` or empty**, if it isn't change the value of **`isFavorite`** to **`true`** and **initialize the content UI accordingly**.
                - **If the cursor is null or empty**, **load** the place details from the **Google and Foursquare APIs**.
            - If it matches **`TRANSACTION_LOADER_ID` cast the `Object` parameter to `String`** and **display the message**.
    - On the **`onCreate()`** method call the **`initLoader()`** method passing **QUERY_LOADER_ID**, `null` and the current instance as parameters.
    - Create a method to switch the action to perform when the button is pressed.
        - **Check if the current place is a favorite** and set the value as a **boolean** variable.

- Call the **`initLoader()`** method from the LoaderManager passing the **TRANSACTION_LOADER_ID**, `null` and the current instance as parameters.

Task 12: Create FavoritePlacesActivity

- Create a **`OnItemClickListener` interface** and declare a void method called **`onItemClick(String placeId)`**
- Create a `FavoritePlacesAdapter` class that **extends** from `Adapter`
  - Add a **private `OnItemClickListener` field** and **add it as a parameter to the constructor** of the class.
  - Add a **private `Context` add it as a parameter to the constructor** of the class.
  - Add a **`ViewHolder` inner class** that **extends** `ViewHolder`
  - Add a `Cursor` **private field** to the adapter and **create a setter method** for the field. **Call `nofifyDataSetChanged()` after the cursor is set**.
  - Implement the adapter methods to inflate and bind a place to the ViewHolder values.
  - **Add a `OnClickListener()` implementation to the `viewItem` instance** of the `ViewHolder` that **calls `onItemClick(String)`** of the `OnItemClickListener` instance of the adapter.
- Create the `FavoritePlacesActivity` class that **extends `AppCompatActivity`** and i**mplements `OnItemClickListener`**.
- Add a `FavoritePlacesAdapter` private field to the class.
- Initialize the adapter and the UI contents of the activity.
  - **Initialize the adapter passing the current activity instance as a parameter**
  - Initialize the `RecyclerView` that displays a list of places and set the **adapter**
- In the onItemClick(String placeID) implementation **start an explicit Intent to `PlaceActivity`** passing **`placeID` as an Intent Extra**.
- Make `FavoritePlacesActivity` **implement** `LoaderCallbacks`
  - In the **`onCreate()`** method **call `initLoader()` from the LoaderManager instance** of the activity.
  - In the method **`onCreateLoader()` return a new `CursorLoader`** that queries all the favorite places.
  - In the method onLoadFinished() pass the obtained cursor as a parameter of the setCursor() method of the adapter field of the activity.
  - In the method onLoaderReset() pass null as the parameter of the setCursor() method of the adapter field of the activity.