Installation: Android Studio

1. Download and install android studio
2. Create an emulator and keep it ready

Overview:

- The app is structured as follows:
    - **3 activities:**
        - Main
        - Searchable
        - Details
    - **2 fragments:**
        - Favorites
        - Tabs

Implementation:

1. **App Icon and Splash Screen**
    - The app begins with a welcome screen which displays the icon. This is also where Darksky is credited for using their APIs and data. This screen is called Splash Screen
    - This screen is added in AppTheme.Launcher
2. **Home Screen**
    - This is the screen where the user's current location is fetched, and its weather details are displayed
    - The location is fetched using IP-API call
    - The latitude and longitude details must be fetched from the API call made to:
        - The latitude and longitude details must be fetched from the API call made to: http://ip-api.com/json
    - This screen contains 3 cards:
        - **Card 1:**
            - Icon
            - Temperature
            - Summary
            - City
            - On-click: clicking this card will open a new 'detailed weather information screen'
        - **Card 2:**
            - Humidity
            - WindSpeed
            - Visibility
            - Pressure
        - **Card 3:**
            - This card is used to display quick overview of next 7 days in tabular format
            - Scroll view is used to achieve this table
            - Each row contains:
                - Date
                - Icon
                - Minimum temperature
                - Maximum temperature

3. **Searching for a city:**
    - On top right side of the screen, there will be a search button which opens a textbox where the user can type name of a city
    - The user is provided with suggestions of city names using the places API
    - When the user taps on a suggestion, it is filled inside the search box and clicking enter/next takes the user to the next page
    - Before you get the data from your backend server, a progress bar should display on the screen
    - On the next page, the user will then be redirected to a new page/activity which will show the summary view
    - This component involves 2 things:
        - Implementing a searchable
        - Implementing autocomplete
    - **Favorites button:** This button will add/remove a city to/from the favorites. This can be implemented using a Floating Action Button. The icon of the button should also change based on whether the city currently belongs to favorites or not
4. **Detailed Weather Information View:**
    - This view has 3 fragments (tabs) and a tweet button in the action bar. This button click opens a new browser window with the twitter intent
    - The tabs are as below:
        - **Today:**
            - It contains 9 cards with icons and weather data:
                - Wind Speed
                - Pressure
                - Precipitation
                - Temperature
                - Icon
                - Humidity
                - Visibility
                - Cloud cover
                - Ozone
        - **Weekly:**
            - There are two components on this page: card and graph
            - **Card:**
                - Icon
                - Summary of the week temperature
            - **Graph:**
                - This view uses MPAndroidChart 3rd party library
                - It contains 2 line charts:
                    - Weekly minimum temperature
                    - Weekly maximum temperature
        - **Photos:**
            - This tab uses Picasso library to display pictures based on the city name
5. **Favorite cities:**
    - Here, the idea is to give users an ability to add a city to Favorites. The favorite cities are persisted even after the user has closed the app. The favorite cities are added as a Dynamic page/tab/fragment on the home page.
    - **Adding to favorite:**
        - Once a user has searched for a city, the summary view contains a FloatingActionButton which indicates "add to favorites" – icon of a map pin with a plus sign
    - **Removing from favorite:**

        - "Remove from favorites" is the same FloatingActionButton as add, but with a different icon of marker with a minus sign. This button appears on 2 places:
            - ❖ On the corresponding dynamically added page on the home screen
            - ❖ On the search result page when the city is searched again
    - On the search result page when the city is searched again. SharedPreferences is used to achieve this
    - The favorite cities are also indicated with dots to indicate the active tab
    - The favorites tab must update dynamically when the user comes back to home screen from a search query. E.g. If the favorites are empty and the user searches for Seattle, from the search summary view if the user clicks the "add to favorite" button and navigates back to the home screen, instead of just current location there should now also be the Seattle tab! And similarly, removing a favorite city must delete the tab
    - The dynamic tabs can be implemented in multiple ways
6. **Progress Bar:**
    - Every time the user must wait before he can see the data, you must display a progress bar

<u>Server side:</u>

- The server side is implemented in Node.js and hosted on AWS elastic beanstalk
- **Places Autocomplete API:**
    - To get the API key for this, follow the steps provided here: https://developers.google.com/places/web-service/get-api-key
    - To use the Autocomplete API, the below URL is used: https://maps.googleapis.com/maps/api/place/autocomplete/json?input=<query>&types=(cities)&language=en&key=[YOUR_GOOGLE_AUTOCOMPLETE_API]
    - An example of an HTTP request to the Places Autocomplete API Get Suggestion that searches for the city text field "Los" is shown below: https://maps.googleapis.com/maps/api/place/autocomplete/json?input=LOS&types=(cities)&language=en&key=[YOUR_GOOGLE_AUTOCOMPLETE_API]

- **Getting Google API key:**
    - Go to the Google Developers Console (https://console.developers.google.com/flows/enableapi?apiid=geocoding_backend&keyType=SERVER_SIDE&reusekey=true)
    - Create or select a project
    - Click Continue to Enable the API
    - Go to Credentials to get a Server key (and set the API Credentials)
    - Copy this key and use it in the code of this project in place of <GOOGLE_API_KEY>

- **Google Geocode API:**
    - The Google geocode API call looks like this: https://maps.googleapis.com/maps/api/geocode/json?address=[STREET,CITY,STATE]&key=[GOOGLE_API_KEY]

- **Getting user's current location:**
    - This is obtained using this url: http://ip-api.com/json

- **Getting DarkSky API key:**
    - Go to https://darksky.net/dev
    - Click on "Try For Free".

- Fill out the form to get the key
- Copy this key and use it in the code of this project in place of <YOUR_DARKSKY_API_KEY>

- **DarkSky API:**
    - This web service URL is constructed as follows: https://api.forecast.io/forecast/[YOUR_DARKSKY_API_KEY]/[LATITUDE,LONGITUDE]
    - Here LATITUDE and LONGITUDE corresponds to the values extracted from Google Geocode API. The response is in JSON format. Required details are obtained from the parsed JSON file

- **Getting Google Custom Search API:**
    - Follow the below documentation to use the Google custom search API key: https://developers.google.com/custom-search/json-api/v1/overview
    - The URL to be used further is this: https://www.googleapis.com/customsearch/v1?q=[CITY]%20City&cx=[YOUR_SEARCH_ENGINE_ID]&imgSize=huge&imgType=photo&num=8&searchType=image&key=[YOUR_GOOGLE_CUSTOM_API_KEY]

Note:

- All the icons/images used in the project are from: https://materialdesignicons.com/
- Run the "server.js" file on localhost (Port 8081) first and then deploy it on cloud. Replace the backend calls in java files accordingly