GitHub Username: Ahmed-Alnabhan

App Name: WiFi Fingerprint

Description

This app enables users to scan WiFi networks and collect data about the access points. The app displays access points and network data in a customizable way and provides charts to support the visualization of data. Furthermore, the app enables users to store WiFi information in a SQLite database and retrieve the chosen data in a form of JSON, XML or CSV file and share the retrieved file.

Intended User

- WiFi network users
- WiFi signals researchers
- Indoor localization and navigation systems builders that use the WiFi network in the working area.

Features

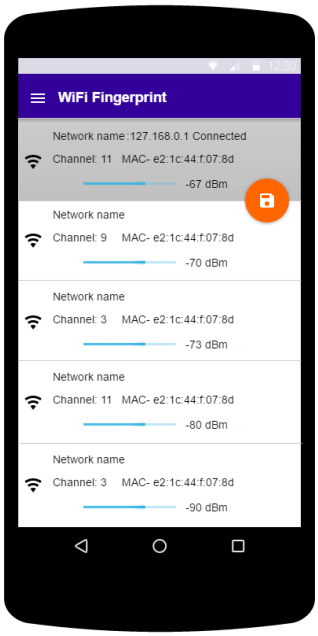
The app provides the following features:

- The app displays information about the currently connected network such as Network name, SSID, IP address and access point's MAC address
- The app displays a list of access points in the range of the smart phone and information about each access point such as Network name, SSID and access point's MAC address.
- The app allows the user to see more information about each access point in the list by taping an access point. The data shown in the access point detail activity includes:
    - SSID (wireless network identifier)
    - AP MAC Address
    - Channel
    - Image of represents signal strength (RSSI).
    - Image of lock or open network.
    - 5GHz or 2.4GHz
    - Frequency
    - AP Manufacturer
    - Type of security protocols such as WPA2, WPA …etc.
- The app stores information of the logged access points into a SQLite database and then retrieve the needed information in a file.
- The app provides three formats of files that retrieve logged access points information: JSON, XML, and CSV.
- Users can share the file that stores the access point's information with other users.
- Users can sort the list of access points on the main activity by:
    - Naturally (As read by the smart phone)
    - Alphabetically
    - Signal strength

- Users can choose to display all access points, only 5GHz or only 2.4GHz.
- Users can use settings to switch between dark and light themes.
- Users can see information about the currently used smart phone such as device brand, manufacturer and version and Android OS version, and type.
- The App uses charts to visualize the signal strength of each access point.
- User can use settings to change the default location where the files are saved

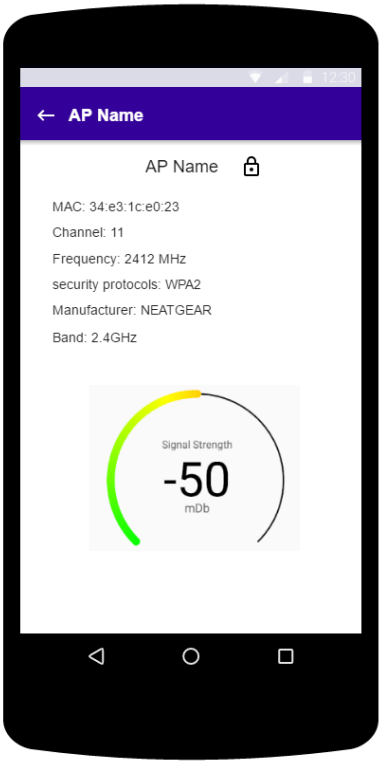
User Interface Mocks

When the user launches the WiFi Fingerprint app, the following screen will show up:



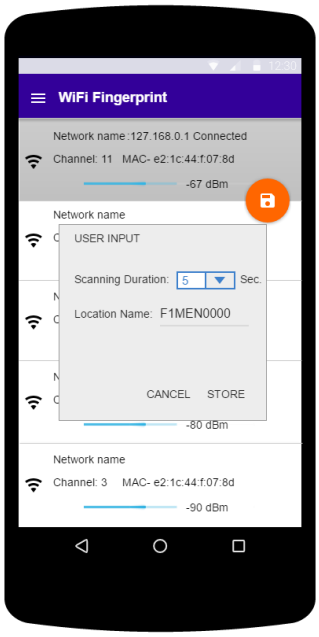
Screen 1

This main activity contains a list of access points that the device can read in its range. The dark item above the list represents the AP that is connected to the device. When a user taps on any item on the APs list, an AP detail screen will show up as follows:



Screen 2

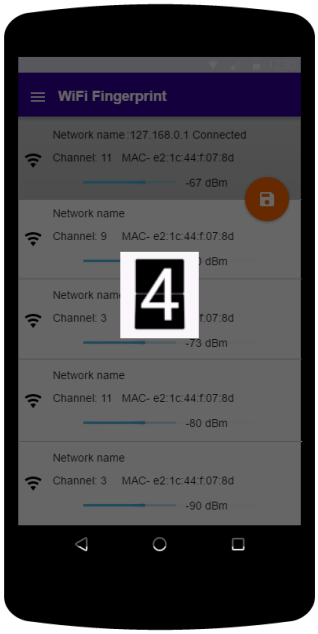
In Screen 1, when a user taps the Floating Action Button FAB, the following screen shows up:



Screen 3

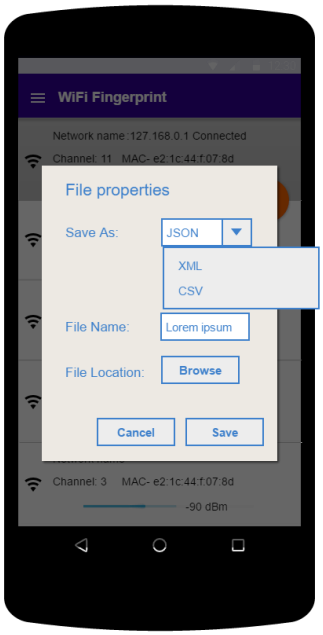
In Screen 3, the user can enter an interval of AP's info reading and a name for the location where they are reading the data. The user can cancel the process or store the readings in a SQLite database using the location name and for the period of time.

In Screen 3, if the user taps the Store button, a timer will show up:



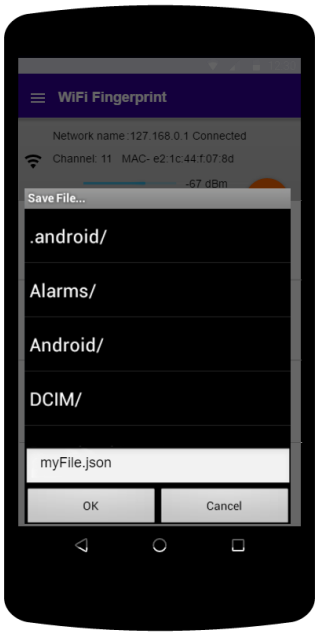
Screen 4

If the timer in Screen 4 ends and storing data process is successful, a screen of saving APs info from the database into a JSON, XML or SCV file shows:



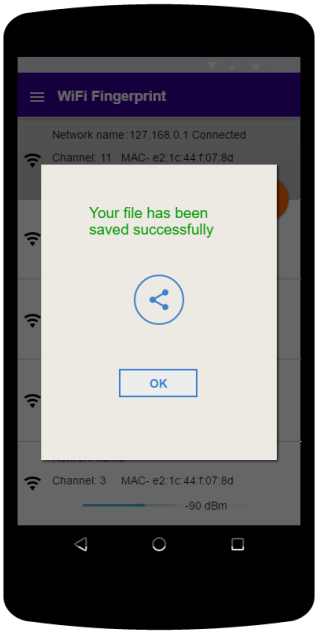
Screen 5

In Screen 5, the user can cancel saving data into a file or specify the file format, name, and location (or accept the default location) and tap the Save button.
If the user taps the Browse button, the following screen will show up to allow the user select a location to save the file:



Screen 6

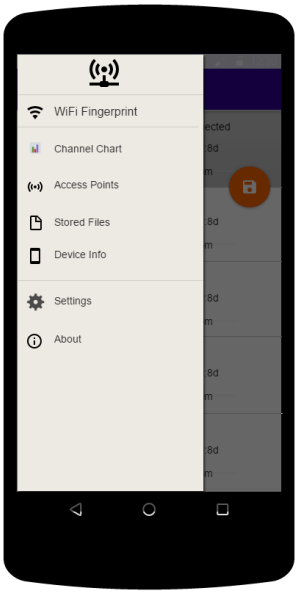
If the file is saved successfully, the following Screen 6 will show up:



Screen 7

In Screen 6, the user can share the file with others.

Returning to Screen 1, when the user taps the hamburger button, a drawer shows on the left side of the screen:

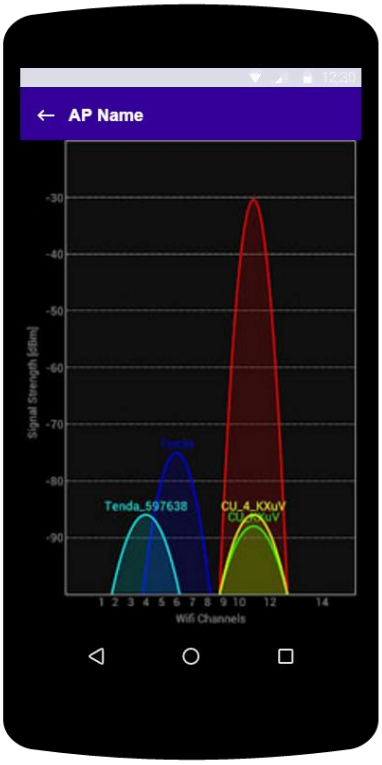


Screen 8

Tapping the WiFi Fingerprint item in the drawer (Screen 7) results taking the user to the main Screen 1.

Tapping the Access Points item in the drawer (Screen 7) results taking the user to the main Screen 1.

Tapping the Channel Chat item in the drawer (Screen 7) results showing the following screen:



Screen 9

Tapping a Stored Files item in the drawer (Screen 7) results showing a list of stored files as illustrated in the following screen:

Screen 10

The list of files in Screen 9 shows file names and locations and allows the user to share any file.

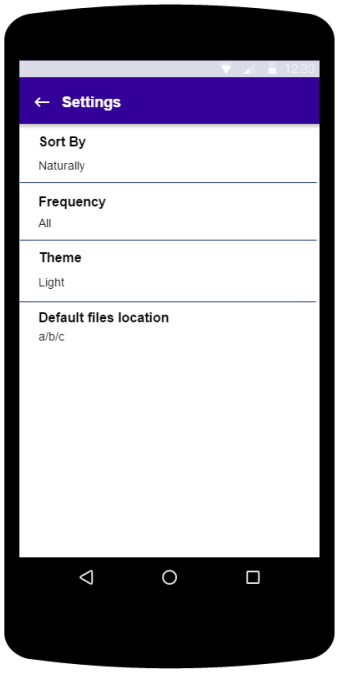
When the user taps Device Info item in the drawer (Screen 7), a screen of device information will show the following screen:

Screen 11

When the user taps the About item in the drawer (Screen 7), information about the WiFi Fingerprint app is displayed on the screen:

Screen 12

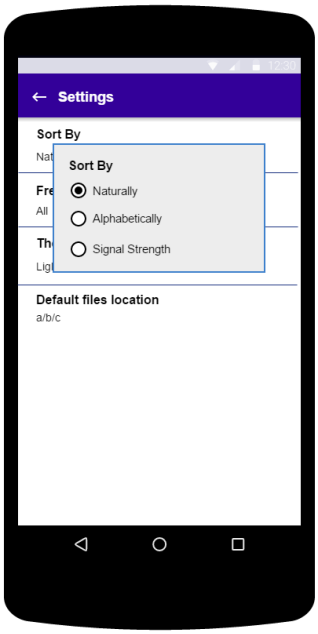
When the user taps the Settings item in the drawer (Screen 7), a setting screen will show up:



Screen 13

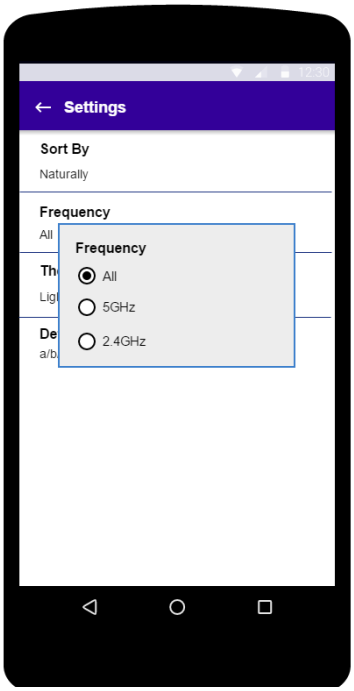
The user can perform the following tasks using the Settings Screen 12:

- Sort the APs list as illustrated in the following screen:



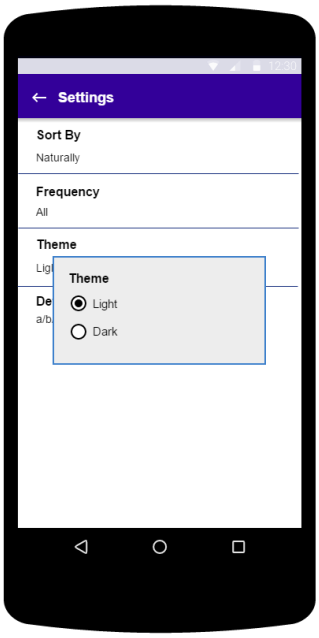
Screen 14

- Display the APs list based on the AP's frequency:



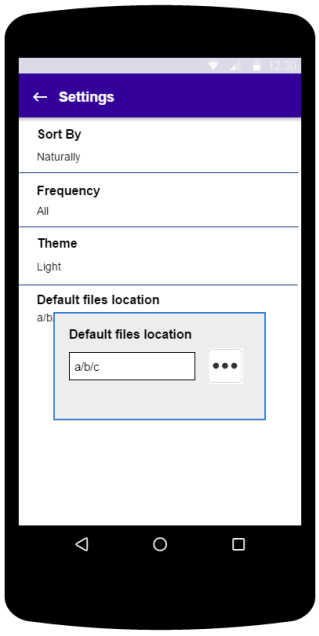
Screen 15

- Choose a light or a dark theme for the app:



Screen 16

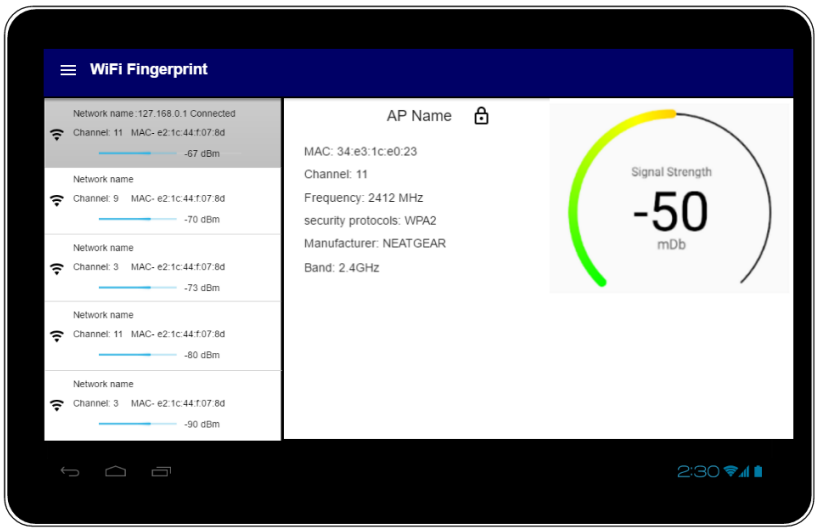
- Choose a new default location to save files:



Screen 17

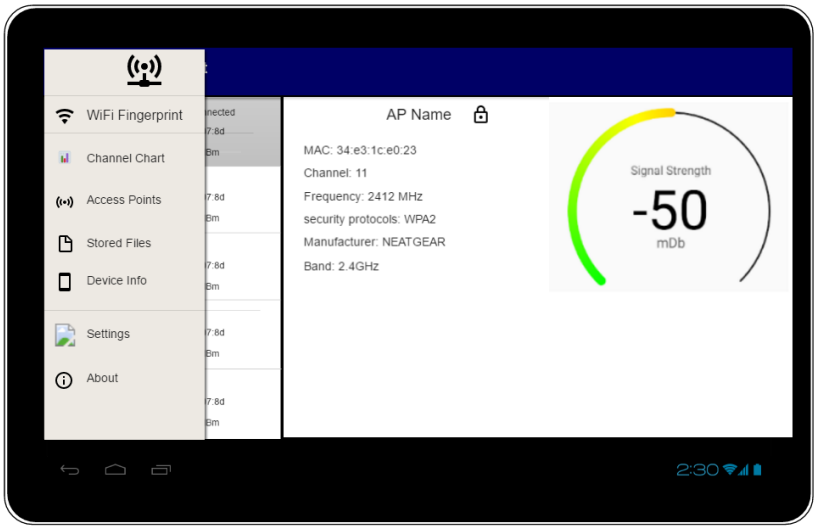
Tablets

- When the user launches the app on a tablet the following screen will show up:



Screen 18

- When the user taps the hamburger icon on the toolbar, the following screen will show up:

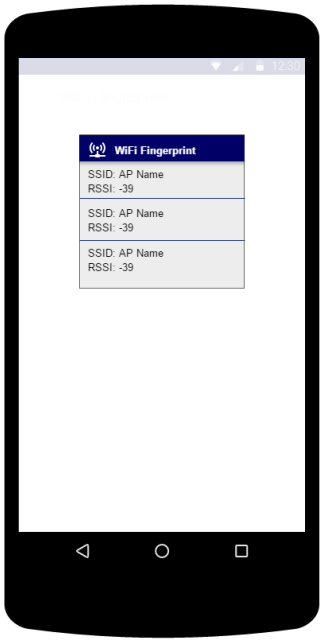


Screen 19

- The rest of screens on the tablet are similar to the corresponding screens on the phone.

App Widget

The app widget is illustrated in the following screen:



Screen 20

Key Considerations

How will WiFi Fingerprint app handle data persistence?

WiFi Fingerprint will handle the AP data using a content provider with a SQLite database at the back. The app will use shared preferences to store the app settings and use files to store distributable AP information. The app supports JSON, XML and CSV format to save the AP data.

Edge or corner cases in the UX.

If the WiFi is turned off by mistake or the airplane mode is active, the app will not be able to read data from access points, therefore, the app will check the device's WiFi connectivity when it is launched and turn it on if it is already off.

Libraries

The app will use the following libraries:

- Android Support library that includes: RecyclerView, AppCompat, and Design.
- ButterKnife library to perform injection on the views and listeners. ButterKnife is lightweight and it allows the developer to focus on the logic instead of glue code. Furthermore, it reduces the code and makes it clean and more readable.
- MPAndroidChart library to create the channel chart. This library is easy to use and provides 8 chart types with animation and predefined color templates.
- Espresso library to implement the UI testing.

Google Play Services or other external services.

The app will use two Google Play Services:

- Google Mobile Ads service. This service can be used by including its dependency in the build.gradle file of the app:
  Compile 'com.google.android.gms:play-services-ads:11.2.0'
- Google Analytics service.

Next Steps: Required Tasks

Task 1: Project Setup

1. The app will use the WifiManager API to manage the WiFi connectivity and reading information. Therefore, the following permissions must be added to the manifest.xml file:
   ```
   <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
   <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
   <uses-permission
   android:name="android.permission.ACCESS_COARSE_LOCATION"/>
   ```

2. A BroadcastReciever should be registered because the app will scan a list of WiFi networks.

3. Create a timer that is used to scan the wifi networks every second. We also need a countdown timer for storing wifi data in the database for a duration that the user specifies.

4. A SQLite database must be created that contains the following tables and fields:

   4.1. location
      4.1.1. _id
      4.1.2. Name

   4.2. wifi_network
      4.2.1. _id
      4.2.2. ssid

   4.3. access_point

        4.3.1. _id
        4.3.2. Location_name
        4.3.3. ssid
        4.3.4. rssi
        4.3.5. frequency
        4.3.6. mac_address
        4.3.7. channel
        4.3.8. isLocked
        4.3.9. manufacturer
        4.3.10. security_protocol
        4.3.11. ip_address

    4.4. file
        4.4.1. _id
        4.4.2. name
        4.4.3. location
        4.4.4. type
5. Write a function that checks the wifi status and turns it on if it is already off.

Task 2: Implement UI for Each Activity and Fragment

- Build UI for access point's list activity. On phones, this activity has only one fragment that uses a RecyclerView to display the list of access points. On tablets, this activity has another fragment to display access point's detail.
- Build UI for access point detail activity. On phones, this activity has a fragment that displays data of a selected access point.
- Build UI for Channel chart activity.
- Build UI for Device info activity.
- Build UI for Files list activity.
- Build UI for the Settings activity.
- Build UI for the About activity.

Task 3: Implement the Content Provider that writes and reads data from the SQLite database described in Task1 above.

- Create a Content Provider.
- Create a Contract class.
- Create the UriMatcher definition.
- Implement callbacks.
- Implement the CRUD methods.
- Add the content provider to the AndroidManifest.xml.
- Create loaders to link the Content Provider with UI elements and load data from the Content provider to the views.

Task 4: Implement scanning AP data and writing that data into the database for a period of time using the content provider.

- Implement FAB click listener
- Implement scanning the WiFi networks using the WifiManager class.

- Implement a countdown timer that takes its upper value from the user.
- Write data on the fly (every second) to the database using the content provider.

Task 5: Implement the APs list activity

- Read ScanResult.
- Create A ResyclerView
- Populate the WiFi networks info into the RecyclerView
- Create A dark item to display the currently connected AP.

Task 6: Implement the selected AP detail activity

- Create an intent in the MainActivity
- Put the information of the selected AP into that intent.
- Start the AP detail activity using the intent.

Task 7: Implement marshalling objects to JSON, XML and CVS formats

- Create an AsyncTask that reads the AP's data from the database using Content Provider.
- Based on the user's choice, marshal the object obtained from the content provider to a corresponding file format:
    - Implement marshalling an object(s) to a JSON file using GSON.
    - Implement marshalling an object(s) to an XML file using JAXB.
    - Implement marshalling an object(s) to a CSV file using Jackson.
- Save the produced file in the selected location.
- Save file info in the database.

Task 8: Create the Channel chart using the MPAndroidChart library

- Read the AP's data on the fly using the WiFiManager.
- Draw a chart that visualize the APs and their RSSIs for every channel.

Task 9: Implement the Settings screen of the app

- Store the settings values in sharedPreferences.
- Change the captions based on the user selection.
- Create the Sort By item.
- Sort the APs list based on the user's choice.
- Create the Frequency item.
- Display only the APs with the selected frequency.
- Create the Theme item.
- Change the theme of the app based on the selected theme.
- Create the Default File Location item.
- Set the new location entered by the user as the default file location.

Task 10: Implement the side drawer

- Create an item that opens the main activity.
- Create an item that opens the Channel chart screen.
- Create an item that opens the APs list screen.
- Create an item that opens the Device Info screen.
- Create an item that opens the Files list screen.
- Create an item that opens the Settings screen.
- Create an item that opens the About App screen

Task 11: Implement the Files list screen

- Read the file info from the database.
- Display file name and location
- Create a share button and functionality for each file.

Task 12: Implement the Device info screen

- Read Device info using Build class.
- Display info on the screen.

Task 13: Implement the About App screen

- Display information about the app and what it does.
- Display the app version.

Task 14: Implement the scanning APs Screen

- Create an activity that allows a user to provide scanning duration.
- Create a text edit that accepts the location name (where the reading takes place)

Task 15: Implement the File info Screen

- Create an activity that allows a user to provide:
    - File name.
    - File type.
    - File location or use the default location

Task 16: Implement the App widget

- Create an app widget that includes the app name.
- Create a list view that list the APs.
- Display the SSID of each AP.
- Display the RSSI of each AP.
- Create a click listener for the app widget that opens the main activity.

Task 17: Implement the UI testing using Espresso

- Create UI test cases.

- Run UI test cases.

Task 18: Implement Google Play Services

- Create the Google Mobile Ad service.
- Create the Analytics service.

Task 19: Implement Release signing configuration

- Configure, build, and test release version
- Create a signed APK file.
- Submit/ release the app.