

6-2018

BOOK-HUNT! ANDROID MOBILE APPLICATION USING INDOOR POSITIONING TECHNOLOGY

Sneha Pantam

Follow this and additional works at: <https://scholarworks.lib.csusb.edu/etd>



Part of the [Computer and Systems Architecture Commons](#)

Recommended Citation

Pantam, Sneha, "BOOK-HUNT! ANDROID MOBILE APPLICATION USING INDOOR POSITIONING TECHNOLOGY" (2018). *Electronic Theses, Projects, and Dissertations*. 744.
<https://scholarworks.lib.csusb.edu/etd/744>

This Project is brought to you for free and open access by the Office of Graduate Studies at CSUSB ScholarWorks. It has been accepted for inclusion in Electronic Theses, Projects, and Dissertations by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.

BOOK-HUNT! ANDROID MOBILE APPLICATION USING
INDOOR POSITIONING TECHNOLOGY

A Project
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Sneha Pantam
June 2018

BOOK-HUNT! ANDROID MOBILE APPLICATION USING
INDOOR POSITIONING TECHNOLOGY

A Project
Presented to the
Faculty of
California State University,
San Bernardino

by
Sneha Pantam

June 2018

Approved by:

Dr. Tong Lai Yu, Advisor, Computer Science

Dr. Kerstin Voight, Committee Member

Dr. Qingquan Sun, Committee Member

© 2018 Sneha Pantam

ABSTRACT

Indoor Positioning System (IPS) focuses on locating objects inside Buildings. Till date, GPS has helped us obtain accurate locations outdoors. These locations have helped us in many ways like navigating to a destination point, tracking people etc. Indoor Positioning System aims at navigating and tracking objects inside buildings. [1] IndoorAtlas is a technology that works on the theory of Indoor Positioning System. Book-Hunt is an Android mobile application which majorly makes use of IndoorAtlas therefore making use of the technique of indoor tracking. This Android mobile application is designed for Libraries. It is designed specifically for John M. Pfau Library, CSUSB, to help the students locate a book in the Library. When a student selects a book, a marker is pointed towards the book and also on the student's current location. This application aims at saving time for student searching a particular book in the Library. Book-Hunt makes use of three tools Android Studio, Google Maps and IndoorAtlas.

ACKNOWLEDGEMENTS

I would like to express my very great appreciation to my advisor Dr. Tong Lai Yu, for all the help he has done throughout this project. He has been a constant helping hand towards the progress of this mobile application. He has advised me and helped me figuring out the errors in the code with a lot of patience. He also took time of his vacation during Summer and had helped me test the application in different situations. I heartily thank him for being such an amazing guide and imparting his valuable knowledge towards Android development. I am also grateful to Dr. Kerstin Voight and Dr. Qingquan Sun who agreed to be a part of my Master's project committee. They have been extremely supportive and helpful during this Project. I thank them for their advice and interest in this Project.

I am particularly grateful for the assistance given by Dr. Javier Torner, officer of Information Security for being very supportive in this journey. He has appointed me as a Student Assistant in his department and has always shown keen interest in my Project. He has always helped me and has constantly encouraged me. His motivating words have driven me towards the completion of my Project. I would also like to thank the Department of Computer Science at the California State University, San Bernardino. Finally, I would like to thank my Parents for supporting me.

TABLE OF CONTENTS

ABSTRACT	iii
ACKNOWLEDGEMENTS.....	iv
LIST OF FIGURES	vii
CHAPTER ONE: INTRODUCTION	1
CHAPTER TWO: DEVELOPMENT TOOLS AND ENVIRONMENT	
Android Studio	2
Features of Android.....	2
Indooratlas	5
What Is IndoorAtlas?	5
Features of IndoorAtlas	5
Advantages of IndoorAtlas	7
Google Maps	8
What Is Google Maps?.....	8
Features of Google Maps.....	8
Advantages:	9
CHAPTER THREE: SOFTWARE DEVELOPMENT	10
Indooratlas And Its Components	11
Indooratlas SDK.....	14
Interfaces of IndoorAtlas SDK.....	15
Classes of IndoorAtlas SDK.....	16
Google Map API	18
Displaying Maps On Screen.....	19

Adding Floor Plans As Overlay	20
Markers, Info Window And Polyline.....	21
Picasso Library	22
CHAPTER FOUR: IMPLEMENTATION	
Libraries And Techniques	23
IndoorAtlas-Android-SDK: 2.3.2	23
Sdk Setup	24
Event Listener Methods	30
Menu Types:	31
Constraint-Layout: 1.0.2.....	33
Subsampling-Scale-Image-View	34
Features.....	36
CHAPTER FIVE: DATA FLOW DIAGRAMS	
Data Flow Diagram of Indooratlas Dashboard	38
Description.....	39
Login:	39
Data Flow Diagram of User.....	47
CHAPTER SIX: UML DIAGRAMS	
Functions of Admin- Use Case Diagram.....	54
Functions of Indooratlas Web Application - Use Case Diagram	56
Functions of Google Map API- Use Case Diagram.....	58
Indooratlas SDK Package-Class Diagram	62
Interaction Between Mobile App And Server-Sequence Diagram.....	64

CHAPTER SEVEN: PROJECT DEVELOPMENT.....	66
Description.....	67
Integrate IndoorAtlas SDK	67
Integrate Google Map API	70
Fetch User's Location	71
Fetch Floor Plan from IndoorAtlas Cloud	75
Floor Detection	77
Add Floor Plan as an Overlay on Google Maps.....	78
Add Marker on User's Location.....	80
Adding a Marker on the Book's Location	82
Adding Polyline between Markers.....	86
CHAPTER EIGHT: LIMITATIONS OF BOOK-HUNT!.....	87
CHAPTER NINE: PROJECT SCREENS.....	88
CHAPTER TEN: CONCLUSION	98
APPENDIX A: ANDROID CODE	98
REFERENCES	137

LIST OF FIGURES

Figure 1. Architechtrual Representation	10
Figure 2. Class Diagram for IndoorAtlas SDK Package	14
Figure 3. Fragment containing Map Object in Layout file	19
Figure 4. Screenshot of Android Code to add Ground Overlay	20
Figure 5. Screenshot of Android Code to add Marker	21
Figure 6. Build Gradle Dependencies	24
Figure 7. Declare Sensors	25
Figure 8. User Permissions	26
Figure 9. Metadata Api Key	27
Figure 10. Picasso Dependencies	28
Figure 11. Picasso Load Function	28
Figure 12. Appcompact Dependency	29
Figure 13. Data Flow Diagram of IndoorAtlas Dashboard	38
Figure 14. IndoorAtlas Login Credentials	39
Figure 15. Create Location	40
Figure 16. Create Location Page in Web application.....	40
Figure 17. Locations added for Project.....	41
Figure 18. Add Floor Plan Dialog box.....	42
Figure 19. Floor plans added in the Project.....	43
Figure 20. Floorplan of Floor 1	43
Figure 21. Aligning Floor Plans.....	44

Figure 22. Adding Way points.....	45
Figure 23. Applications in IndoorAtlas	46
Figure 24. API key used for Book Hunt!	46
Figure 25. Data Flow Diagram of User	47
Figure 26. Checking Location on Floor plan	48
Figure 27. Toast to direct student towards the book.....	49
Figure 28. Mark my book button	50
Figure 29. Alert box confirming floor.....	50
Figure 30. Polyline showing navigation	51
Figure 31. Found my book button.....	52
Figure 32. Thank you message	52
Figure 33. To Search for another Book	53
Figure 34. Use case Diagram for Admin.....	54
Figure 35. Use case diagram for IndoorAtlas Web application	56
Figure 36. Use case Diagram for Google Maps API.....	58
Figure 37. Use case diagram for User.....	60
Figure 38. Class Diagram for IndoorAtlas Sdk Package.....	62
Figure 39. Sequence Diagram to show Interaction between Mobile Application and Server	64
Figure 40. Data Flow Diagram for Project Development	66
Figure 41. Adding Dependencies in Android Project	67
Figure 42. Adding sensors and Permissions to Android Manifest file	68

Figure 43. Syntax to add API key in Android Manifest file	68
Figure 44. Adding API key and secret to Android Manifest file	69
Figure 45. Adding Google Map dependencies.....	70
Figure 46. Adding API key to Manifest file	70
Figure 47. Runtime Permissions.....	71
Figure 48. Code for Runtime Permissions	72
Figure 49. Calling IALocationManager in onCreate().....	73
Figure 50. Fetching Location Updates	74
Figure 51. To stop Receiving Location Updates	74
Figure 52. Fetch floor plan code	76
Figure 53. Picasso download floor plan image code	76
Figure 54. Region detection code.....	77
Figure 55. Adding floor plan as Overlay	78
Figure 56. Exit region	79
Figure 57. Adding Marker	80
Figure 58. Marker in Mobile application.....	81
Figure 59. Book Waypoints in IndoorAtlas web app	82
Figure 60. Individual Activities for books	83
Figure 61. Waypoint Book Location.....	84
Figure 62. Adding Marker on Book's Location	84
Figure 63. Book Marker on Mobile application	85
Figure 64. Polyline in Mobile application	86

Figure 65. Screen of First Page	88
Figure 66. Screen of Help Page	89
Figure 67. Screen of First Page- Select a Book.....	90
Figure 68. Screen of Second Page- Toast to display floor number	91
Figure 69. Screen of Second Page.....	92
Figure 70. Screen of Second Page- Marker on User's Location.....	93
Figure 71. Screen of Second Page- Hitting Mark my book Button.....	94
Figure 72. Screen of Third Page- Marker on book's location.....	95
Figure 73. Screen of Third Page- Polyline between two markers	96
Figure 74. Screen of Third Page- After hitting Found my book button.....	97

CHAPTER ONE

INTRODUCTION

A Smartphone collects radio signals, geomagnetic fields, inertial sensor data, barometric pressure, camera data and other sensory information. Using these the Indoor positioning systems (IPS) locates people inside a building. This project is built using the IndoorAtlas SDK, which works on the theory of Hybrid Indoor Positioning Technology. [1]. This application makes use of IndoorAtlas which helps us fetch floor plans and detect regions as we move from one floor to another. The Application directs a student towards the floor of the book marks the book on the floor plan using a Google Map Marker, and after he reaches the appropriate floor. IndoorAtlas also provides us a Dashboard feature that enables users to upload floor plans to its server or cloud. We are able to align the floor plan to the exact coordinates of the building in a very easy manner. Its mobile app MapCreator 2 helps us map the area of the building.

Google Maps plays a very crucial role in this Project. Its Android API helps us display maps on screen, add markers on the locations (book and user) and also draws a route path with a functionality called as Polyline. [3]. This project will be very useful to the students in the John M. Pfau Library.

CHAPTER TWO

DEVELOPMENT TOOLS AND ENVIRONMENTS

Android Studio

For Google's Android operating system, Android Studio is the official integrated development environment (IDE). It is built on JetBrains' IntelliJ IDEA software and specifically designed for Android development. It is available for download on all Operating systems like Windows, macOS and Linux. Eclipse Android Development Tools (ADT) had been the primary IDE for Native Android applications. Android studio has been a replacement for Eclipse. [4]

Features of Android

The following features are provided in the current stable version:

- Gradle-based build support
- Android-specific refactoring and quick fixes

Lint tools to catch performance, usability, version compatibility and other problems

- ProGuard integration and app-signing capabilities

Advantages of Android Studio.

1. Instant Run helps Faster Deployment

Bringing incremental changes to an existing app code or resource is now easier and faster. The Instant Run feature enables us to witness the Code changes on the emulator or physical device in real-time without building a new APK (Android Application Package file) or restarting app each time.

2. Accurate and Easier Programming

Android Studio makes code writing faster and easier. As it is equipped

with an intelligent IntelliJ IDEA interface the code analysis has become more accurate.

3. Comparatively Faster Testing and Programming

In comparison to its predecessor the newly introduced emulator is three

times faster in I/O, CPU and RAM. The virtual testing environment has a user-friendly UI (User interface) and is faster in showing the changes in code. Every move of the developers is effectively read by the Sensor controls. Developers can use the multi-touch actions (pinch, pan, rotate and tilt) and also drag and drop APKs for quick installation.

4. Inclusive App Development using Cloud Test Lab Integration

Now the users can build for one and test on multiple devices using the

feature of Cloud Test Lab Integration. The compatibility and performance of an application can be compared on a wide range of physical Android devices within Android Studio.

Indooratlas

What Is IndoorAtlas?

IndoorAtlas makes use of Indoor positioning systems (IPS) to locate people or objects inside a building. A Smartphone collects radio signals, geomagnetic fields, inertial sensor data, barometric pressure, camera data and other sensory information. Using this data the Indoor Positioning Systems tracks people inside buildings. [1]

Features of IndoorAtlas

Way-Finding IndoorAtlas builds wayfinding capabilities in your mobile app and increases client satisfaction. This helps users to:

5. Find correct department or aisle in a store. Navigate quickly inside complex buildings
6. Avoid long waiting time and crowded areas
7. Save time finding an object or location.
8. Create an environment that is safe

Multi-Dot capability.

This feature lets multiple users and object locations to be visualized on a map and managed. This is great for team navigation and social networking as all the devices can be located at once in real-time while using a mobile app. This helps users to:

- Connect with their friends in the same location
- Find their co-worker
- Track people and remain safe

Proximity Marketing. This technology provides people with an underlying platform to build highly targeted proximity marketing within your mobile app. This helps users to:

- I. Receive coupons often
- II. Get alerts about product offers
- III. Save their time

Advantages of IndoorAtlas

It is Highly Scalable: Enterprise-grade cloud platform simplifies your infrastructure enabling you to scale cost effectively, quickly and securely

Reduced Costs: No need to purchase, install and maintain large amounts of costly infrastructure

Accurate Positioning: 2-3 meter accuracy with blue dot positioning

Ease of Deployment: Intuitive workflow enables easy implementation, designed with the developer in mind

Ubiquitous: Available on Android and iOS devices in any steel/concrete reinforced building.

Foundational: Technology can utilize Wi-Fi and Bluetooth for further optimization

Google Maps

What Is Google Maps?

Google offers a web mapping Service called the Google Maps. It offers maps in different set of views like the satellite view, panoramic view, street view for better User Interface. Google Maps provides us with an API which allows the users to embed maps into any website or mobile application. Route Planner is a main feature Google Maps which provides us four modes of transportation - Car, train, cycle or walk. [3]

Features of Google Maps

- i. Figuring out ways through big and confusing buildings can be a confusing task. Google Maps provides navigation through these buildings
- ii. It allows users to Download maps such that you can access them in situations where you are not connected to the Internet
- iii. It helps you keep track of what places you have visited in the last year.
- iv. Also helps users to book flight tickets

- v. Lets us see confusing road trips at a glance
- vi. It saves the home or work addresses
- vii. Get tickets for concerts and shows
- viii. Get directions using Route planner. Get Directions to any target or destination
- ix. Gives turn by turn navigations to any destination quickly. You can choose any mode of transportation
- x. Explore all the wonders of the world

Advantages:

- Full of information.
- Sharing benefits
- Multiple Modes of Transport

CHAPTER THREE

SOFTWARE DEVELOPMENT

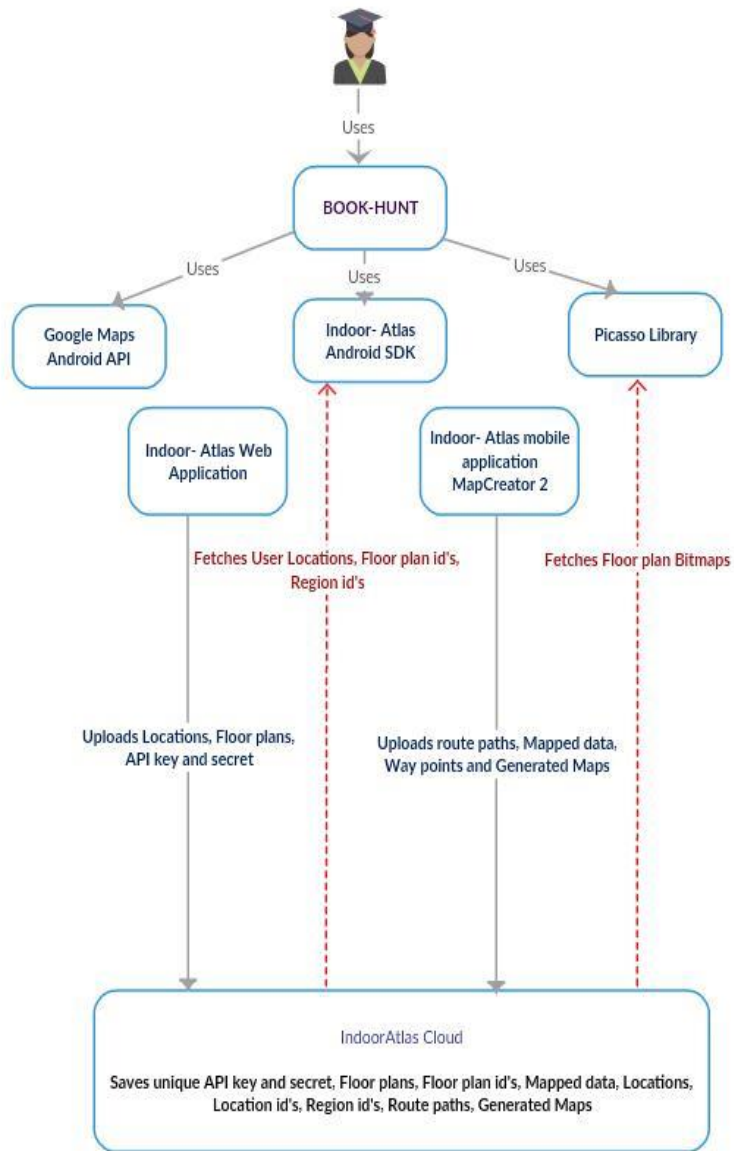


Figure 1. Architectural Representation

Book Hunt is an Android mobile application that aims at locating a book in the Library. This application is designed especially for the John M. Pfau Library at California State University, San Bernardino. As we have discussed earlier, searching for books in a library can be a challenging task. Fortunately, this application makes our task easier. As shown in the above figure, the major software components in Book Hunt are IndoorAtlas and Google Maps. This mobile application majorly focuses on locating the user inside a Building and locating the book inside a building. Firstly, we will discuss on how we locate the user inside a Building.

Indoor Positioning Systems is a technology which work on locating objects indoors. As the microwave signals do not pass inside buildings, GPS can only detect the location outdoors. The Indoor Positioning Systems have been working on detecting objects inside buildings using the radio signals and magnetic waves collected by a smart phone. [1]

Indooratlas And Its Components

IndoorAtlas is a technology that works on the theory of Indoor Positioning system. They make use of the Wi-Fi and Bluetooth inside a building and calculate the position of the user inside a building. IndoorAtlas has three main components that helps us detect user locations, they are IndoorAtlas web

application, IndoorAtlas MapCreator 2 and IndoorAtlas SDK. IndoorAtlas Web Application is a dashboard service which enables us to add locations of the building we need to Map. We are able to add only one location at a time to a Mobile application. Each Application has a unique API key and secret. This key distinguishes one mobile application to another. After creating the Location and generating the API key and secret, we upload floor plans. Each floor plan needs to be aligned carefully with the Co-ordinates. These floor plans are saved in the IndoorAtlas cloud and can be fetched by the Book-Hunt using the Picasso Library. [5]

IndoorAtlas MapCreator 2 is a Mobile application which maps the locations uploaded by the user. Mapping or Fingerprinting is the process of gathering signal data from a venue. This application must be installed from Google Play on an Android phone. This application helps us map the area of the location that has been added. You map the area by walking around and collecting data. This data is collected using Wi-Fi and Bluetooth signals in the smartphone. This data is sent automatically to the IndoorAtlas cloud. The IndoorAtlas cloud saves all the data uploaded by the user.

IndoorAtlas cloud is the Server setup by the IndoorAtlas Company, which distinguishes the data of all the users by its API key and id. The IndoorAtlas Cloud stores, the locations, the location data, the floor plans uploaded by the

user, the floor plan ids assigned, the Region numbers of the floor and their ids. This information has been uploaded by the user from the web application. The IndoorAtlas also stores the generated maps, the route paths and waypoints. This information is stored by the mobile application. [2]

IndoorAtlas SDK

IndoorAtlas SDK, is developed by IndoorAtlas to use the features of IndoorAtlas in the mobile applications. They have a cross platform SDK both for Android and iOS. In this project Android SDK has been used. The Main Features of this SDK include fetching the location data from IndoorAtlas Cloud. Fetching the information of the region and fetching the information of the floor. The Package of IndoorAtlas SDK has a set of Interfaces and Classes

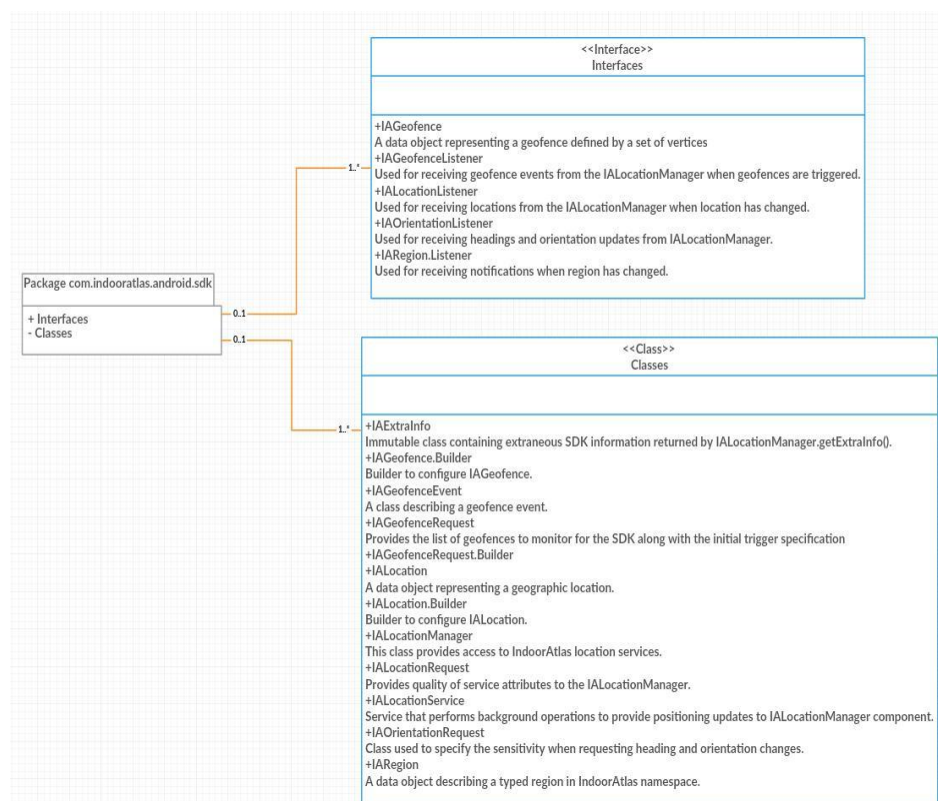


Figure 2. Class Diagram for IndoorAtlas SDK Package

Interfaces of IndoorAtlas SDK

- IAGeofence

A data object that represents a geofence by a set of vertices

- IAGeofenceListener

It is used to receive geofence events from the IALocationManager when the geofences are called

- IALocationListener

It is used for receiving locations from the IALocationManager when the location has been changed

- IAOrientationListener

It is used for receiving orientation updates and headings from IALocationManager

- IARegion.Listener

It is used for receiving notifications when the region has been changed

Classes of IndoorAtlas SDK

- IExtraInfo

It's an immutable class that contains extraneous SDK information which is returned by `ILocationManager.getExtraInfo()`.

- IGeofence.Builder

It's a builder to configure IGeofence.

- IGeofenceEvent

A class that describes a geofence event.

- IGeofenceRequest

It provides the list of geofences to monitor the SDK

- ILocation

A data object that represents a geographic location.

- `IALocation.Builder`
It's a builder that configures `IALocation`.
- `IALocationManager`
Provides access to IndoorAtlas location services.
- `IALocationRequest`
This class provides quality of service attributes to the `IALocationManager`.
- `IALocationService`
Service that performs background operations to provide positioning updates to
- `IALocationManager` component.
- `IAOrientationRequest`
Class used to specify the sensitivity when requesting heading and orientation changes.
- `IARegion`
A data object describing a typed region in IndoorAtlas namespace.

Google Map API

Google Map API plays a major role in this project. The user location that is collected from the IndoorAtlas Cloud needs to be displayed on Google Maps as the SDK retrieves Location objects from the Cloud. The Location objects are a set of coordinates having latitudes and longitude data. Google Maps API is known for displaying location of the coordinates on the map. [3]

The main functions of Google Map API in this project are:

- Displaying maps on screen
- Adding floor plans as Ground Overlay
- Adding Marker on user location
- Adding Marker on Book location
- Displaying Information about the book in Info window.
- Adding Polyline

Displaying Maps On Screen

The `onMapReady()` function is responsible to display maps on the screen. The layout file of the activity must have a `<fragment>`. This element must have a `SupportMapFragment` to act as a container for Google Maps and access the `Map` object.

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:map="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_alignParentBottom="true"
    android:layout_alignParentStart="true"
    tools:context="com.example.snehapantam.runtime.MapsActivity" />
```

Figure 3. Fragment containing Map Object in Layout file

Adding Floor Plans As Overlay

Google Maps API lets us add images on maps as overlays. This serves as an advantage to this project as we can fetch the floor plan from Cloud and add it as an overlay on Google Maps. [3]

```
private void setupGroundOverlay(IAFloorPlan floorPlan, Bitmap bitmap) {  
  
    if (mGroundOverlay != null) {  
        mGroundOverlay.remove();  
    }  
  
    if (mMap != null) {  
        BitmapDescriptor bitmapDescriptor = BitmapDescriptorFactory.fromBitmap(bitmap);  
        IALatLng iaLatLng = floorPlan.getCenter();  
        LatLng center = new LatLng(iaLatLng.latitude, iaLatLng.longitude);  
        GroundOverlayOptions fpOverlay = new GroundOverlayOptions()  
            .image(bitmapDescriptor)  
            .position(center, floorPlan.getWidthMeters(), floorPlan.getHeightMeters())  
            .bearing(floorPlan.getBearing());  
  
        mGroundOverlay = mMap.addGroundOverlay(fpOverlay);  
    }  
}
```

Figure 4. Screenshot of Android Code to add Ground Overlay

Markers, Info Window And Polyline

Markers are used for indicating geographic locations on map. Info Windows provide information to the marker. Using this feature of Google Maps, we add markers on the user's location and Book's location. As mentioned earlier, `CLLocationManager` fetches the location data as location objects. Google map places a standard icon on locations. In this project we have added custom markers designed separately. The icon that represents a user is a black circle dot and the icon that represents a book is a red symbol of a book. A Polyline represents a line from one marker to another to display navigation. [3]

```
Marker mMarker1 = mMap.addMarker(new MarkerOptions().position(mercury_book).title("Mercury").snippet("First shelf")
    .icon(BitmapDescriptorFactory.fromResource(R.mipmap.book_icon)));

mMarker1.showInfoWindow();
polylinemethod();
```

Figure 5. Screenshot of Android Code to add Marker

Picasso Library

To fetch floor plan from IndoorAtlas we use Picasso library. Picasso allows us to easily download images from target URLs. To sum it up, we fetch the floor plans from IndoorAtlas Cloud using Picasso, add them as a Ground Overlay on Google Map, Fetch locations from cloud using IndoorAtlas SDK, Add Marker on user locations and then add marker on the book location. We also add a polyline to show navigation route from user location to the book location. [5]

CHAPTER FOUR

IMPLEMENTATION

Libraries And Techniques

The Libraries used in this Mobile Application are:

IndoorAtlas-Android-SDK: 2.3.2

To run Indoor Atlas SDK on an Android device, you need

- Android SDK (minimum API 10: Gingerbread)
- Working Android phone with Wi-Fi connectivity. An emulator is not supported.
- Magnetometer and Gyroscope are preferred

Features of IndoorAtlas SDK.

- Location Updates
- Fetching floor plans from Indoor Atlas Cloud
- Floor Detection

Sdk Setup

Add SDK Dependency.

For new projects built with Gradle, IndoorAtlas recommends using AAR as it is easier to integrate. AAR contains both Java classes and *AndroidManifest.xml* template which gets merged into your application's *AndroidManifest.xml* during build process.

Add this to your *build.gradle* file.

Add this to your *build.gradle* file.

```
dependencies {  
    compile 'com.indooratlas.android:indooratlas-android-sdk:  
    {{site.versions.androidSdkVersion}}@aar'  
}  
repositories{  
    maven {  
        url "http://indooratlas-ltd.bintray.com/mvn-public"  
    }  
}
```

Figure 6. Build Gradle Dependencies

Declare Sensors.

The SDK uses three hardware sensors and Wi-Fi when available. It will function also without some of these sensors to a limited degree, but if your app requires full performance and you're willing to limit its device compatibility, declaring these in the manifest restricts the devices on which the SDK can be installed from Google Play. You can do this by adding the following declarations to your *AndroidManifest.xml* before *<application>* element.

```
<uses-feature android:name="android.hardware.sensor.accelerometer"
    android:required="true" />
<uses-feature android:name="android.hardware.sensor.compass"
    android:required="true" />
<uses-feature android:name="android.hardware.sensor.gyroscope"
    android:required="true" />
<uses-feature android:name="android.hardware.wifi"
    android:required="true" />
```

Figure 7. Declare Sensors

Enabling Beacon Support.

Beacons are not strictly necessary for positioning to work but enabling this feature is recommended for optimal performance.

During runtime, SDK checks if permissions are granted and that Bluetooth service is enabled by the user. Support for beacons is silently ignored if these conditions are not met.

As default, SDK *does not* add required permissions to scan beacons. To enable support, add the following permissions to your applications

AndroidManifest.xml:

```
<uses-permission android:name="android.permission.BLUETOOTH" />  
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

Figure 8. User Permissions

Add Credentials To The Android Manifest.

Every application which uses IndoorAtlas service needs a unique API Key and Secret strings, which you can manage with IndoorAtlas Applications.

Add your app credentials as *meta-data* attributes to *AndroidManifest.xml*

```
<application>
  <meta-data
    android:name="com.indooratlas.android.sdk.API_KEY"
    android:value="api-key-here"/>
  <meta-data
    android:name="com.indooratlas.android.sdk.API_SECRET"
    android:value="api-secret-here"/>
</application>
```

Figure 9. Metadata Api Key

Android Picasso.

Android Picasso is a powerful image downloading and caching library. It is an image loading/processing library developed and maintained by Square Inc. It's immensely popular since it often requires just one line of code and has a similar style of coding for each of its features. To use the android Picasso Library in your Android Studio project, add the following dependency in your build.gradle file.

```
compile 'com.squareup.picasso:picasso:2.5.2'
```

Figure 10. Picasso Dependencies

To load a image from URL in an ImageView using Picasso API, following code snippet is commonly used.

```
Picasso.with(context).load("http://cdn.journaldev.com/wp-content/uploads/2016/11/android-image-picker-project-structure.png").into(imageView)
```

Figure 11. Picasso Load Function

Appcompat V7.

This library adds support for the Toolbar interface design pattern (including support for the ActionBar design pattern). This library includes support for material design user interface implementations. Provide backward-compatible versions of Android framework APIs. This library is designed to be used with Android 1.6 (API level 4) Android 2.3 (API level 9) and higher. It includes the largest set of APIs compared to the other libraries, including support for application components, user interface features, accessibility, data handling, network connectivity, and programming utilities. [5]

- App bar, formerly known as Action bar in Android.
- AppBar is a special kind of toolbar that's used for branding, navigation, search, and actions.

Dependencies and prerequisites

1. Android 2.1 (API level 7) or higher

```
com.android.support:appcompat-v7:21.0.+
```

Figure 12. Appcompact Dependency

Event Listener Methods

- i. `onClick()` - From `View.OnClickListener`. Called when user either touches item (when in touch mode), or focuses upon item with navigation-keys or trackball and presses suitable "enter" key or presses down on trackball.
- ii. `onLongClick()` - From `View.OnLongClickListener`. Called when user either touches and holds item (when in touch mode), or focuses upon item with navigation-keys or trackball and presses and holds suitable "enter" key or presses and holds down on trackball (for one second).
- iii. `onFocusChange()` - From `View.OnFocusChangeListener`. Called when user navigates onto or away from item, using navigation-keys or trackball.
- iv. `onKeyDown()` - From `View.OnKeyListener`. Called when user is focused on item and presses a hardware key on device.
- v. `.onKeyUp()` - From `View.OnKeyListener`. Called when user is focused on item and releases a hardware key on device.
- vi. `.onTouchEvent()` - From `View.OnTouchListener`. Called when user performs an action qualified as a touch event, including a press, a release, or any movement gesture on screen (within bounds of item). [4]

Menu Types:

Options menu and app bar.

Options menu is primary collection of menu items for an activity. It's where you should place actions that have a global impact on app, such as “Search,” “Compose email,” and “Settings”. In Android 2.3 or lower, users can reveal options menu panel by pressing Menu button.

On Android 3.0 and higher, items from options menu are presented by app bar as a combination of on-screen action items and overflow options. Beginning with Android 3.0, Menu button is deprecated, so you should migrate toward using action bar (AppBar) to provide access to actions and other options. [4]

Context menu and contextual action mode

A context menu is a floating menu that appears when user performs a long-click on an element. It provides actions that affect selected content or context frame.

In Android 3.0 and higher, use contextual action mode to enable actions on selected content. This mode displays action items that affect selected content in bar at top of screen and allows user to select multiple items.

Popup menu.

Displays a list of items in a vertical list that's attached with view that invoked menu. It's good for providing an overflow of actions that relate to specific content or to provide options for a second part of a command.

Actions in a popup menu should not directly affect the corresponding content—that's what contextual actions are for. Rather, popup menu is for extended actions that relate to regions of content in your activity. [4]

Constraint-Layout: 1.0.2

According to android developers user guide, Constraint Layout allows you to create large and complex layouts with a flat view hierarchy (no nested view groups). It's similar to Relative Layout in that all views are laid out according to relationships between sibling views and the parent layout, but it's more flexible than Relative Layout and easier to use with Android Studio's Layout Editor. It involves simple drag and drop View widgets from the Palette to the designer editor. [5]

Constraint Layout Requirements.

- Android Studio version 2.2 and above
- Android min SDK 2.3

Subsampling-Scale-Image-View

This is a custom image view for Android, designed for photo galleries and displaying huge images. It includes all the standard gestures for zooming and panning images, and provides some extra useful features for animating the image position and scale and rotating the image.

The view is highly configurable, and designed for extension from the ground up. You can easily add your own overlays anchored to points on your image, and customise event detection. See the sample app and its source code for some examples.

The aim of this library is to solve some of the common problems when displaying large images in Android:

"Bitmap too large to be uploaded into a texture" errors caused by attempting to display images over 2048px wide or high. The view will automatically load images larger than this limit in tiles so no single bitmap is too large to display.

OutOfMemoryErrors caused by loading a large image into memory without subsampling. The view subsamples images, then loads high resolution tiles for the visible area as the user zooms in.

Features

Image display.

Display images from assets, resources, the file system or bitmaps

Automatically rotate images from the file system (e.g. the camera or gallery) according to EXIF

Manually rotate images in 90° increments

Display a region of the source image

Use a preview image while large images load and swap images at runtime

With tiling enabled:

Display huge images, larger than can be loaded into memory

Show high resolution detail on zooming in

Tested up to 20,000 x 20,000 px, though larger images are slower

Gesture detection

One finger pan

Two finger pinch to zoom

Quick scale (one finger zoom)

Pan while zooming

Fling momentum after panning

Double tap to zoom in and out

Options to disable pan and/or zoom gestures

Animation

Public methods for animating the scale and center

Customizable duration and easing

Optional uninterruptible animations

Overridable event detection

Supports OnClickListener and OnLongClickListener

Supports interception of events

using GestureDetector and onTouchListener

CHAPTER FIVE

DATA FLOW DIAGRAMS

Data Flow Diagram of IndoorAtlas Dashboard

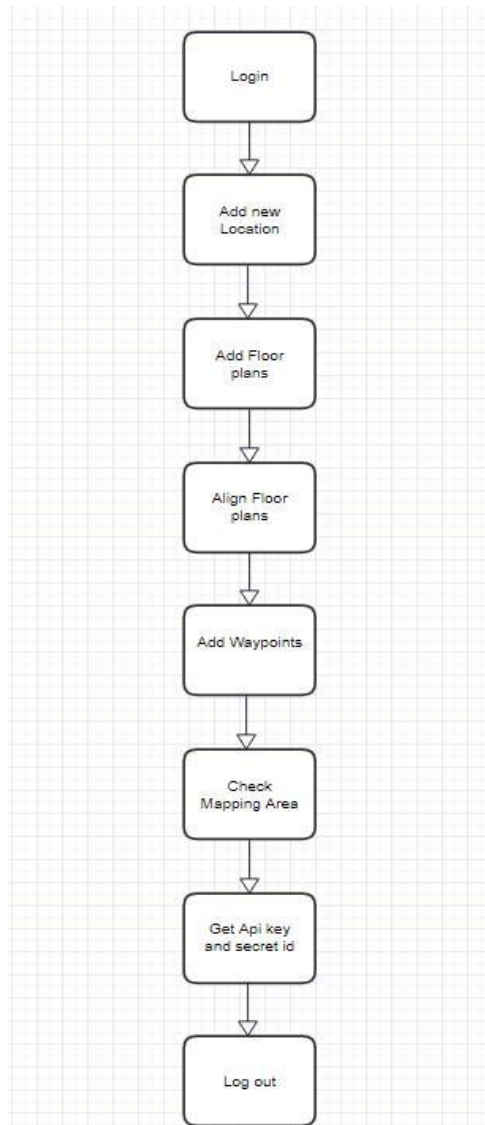


Figure 13. Data Flow Diagram of IndoorAtlas Dashboard

Description

IndoorAtlas Dashboard is a web portal which manages the floor plans, maps, API keys and Applications of IndoorAtlas. These steps describe us how the IndoorAtlas Dashboard works:

- Login:

The first step is to create a Create an Account with IndoorAtlas. Use the Credentials to Login into their Dashboard. These are the credentials used in this project.

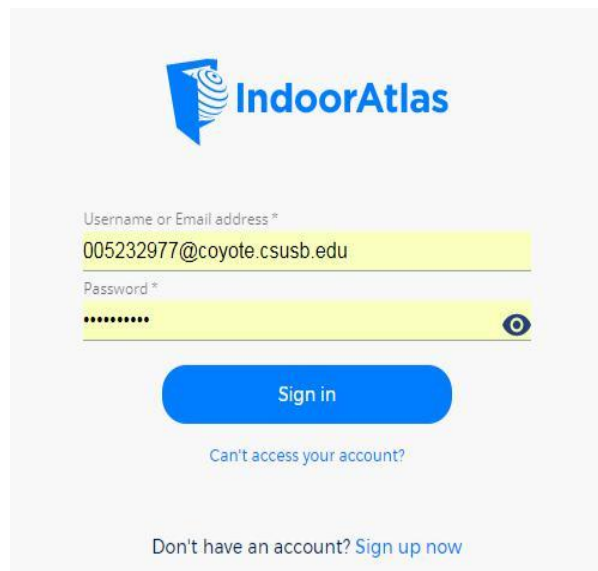
The image shows the IndoorAtlas login interface. At the top is the IndoorAtlas logo, which consists of a blue stylized 'I' icon followed by the text 'IndoorAtlas' in blue. Below the logo are two input fields. The first field is labeled 'Username or Email address *' and contains the text '005232977@coyote.csusb.edu'. The second field is labeled 'Password *' and contains a series of dots, with a small eye icon to its right. Below the password field is a blue rounded rectangular button with the text 'Sign in'. Underneath the button is a link that says 'Can't access your account?'. At the bottom of the form is another link that says 'Don't have an account? Sign up now'.

Figure 14. IndoorAtlas Login Credentials

- Add New Location

Next, We Add the Location we wish to use in our application. We do this by clicking the button named as 'Create Location'.

Locations

[+ Create location](#)

Figure 15. Create Location

Locations can be added by creating manually from floorplan images or importing from Micello.

[← Create location](#)

This is the building or other venue where you place your floor plans. Don't create copies of the same location! You can place multiple floor plans into one location.

Name *

Be as descriptive and accurate as possible, e.g. My Company Inc. / ABC Shopping Mall

Address

Champagne Avenue, Kern County, California

You can drag the marker on the map to choose the address or use the text field for searching.

Description (optional)

Latitude *

34.9286

Longitude *

-118.4052

[Cancel](#) [Submit](#)

MapBox

+

-

CHAMPAGNE AVE.

Figure 16. Create Location Page in Web application

The Location that has been used in this project is John M. Pfau Library located at California State University San Bernardino.

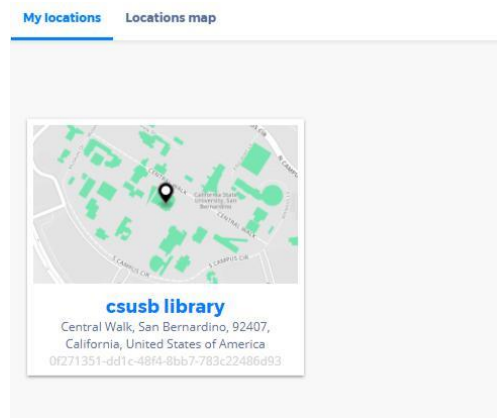


Figure 17. Locations added for Project

- Add Floor plans

Floorplans are used to define Indoor coordinate frames on which one can do mapping. In this web portal, the floorplans are positioned on the world map to bind our Indoor Locations to the standard global coordinate system, WGS-84. All floorplans must be given a floor number which determines correct vertical ordering within a location.

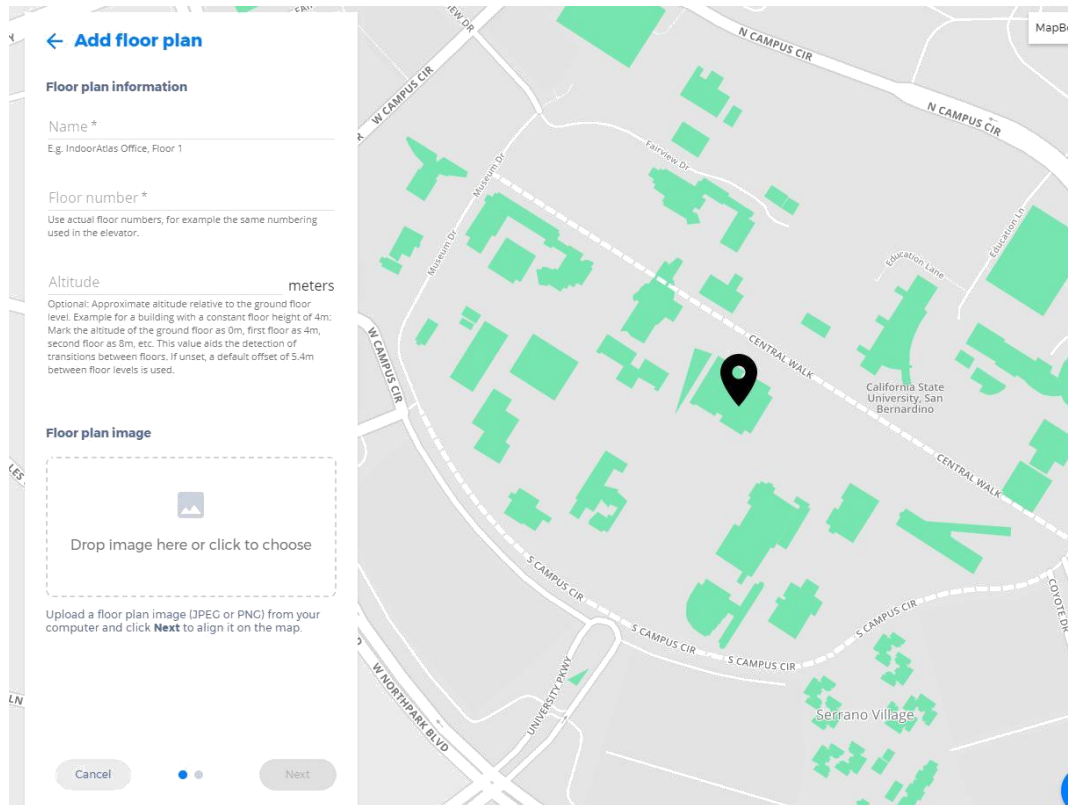


Figure 18. Add Floor Plan Dialog box

In this project, we have uploaded the floorplans of four floors.

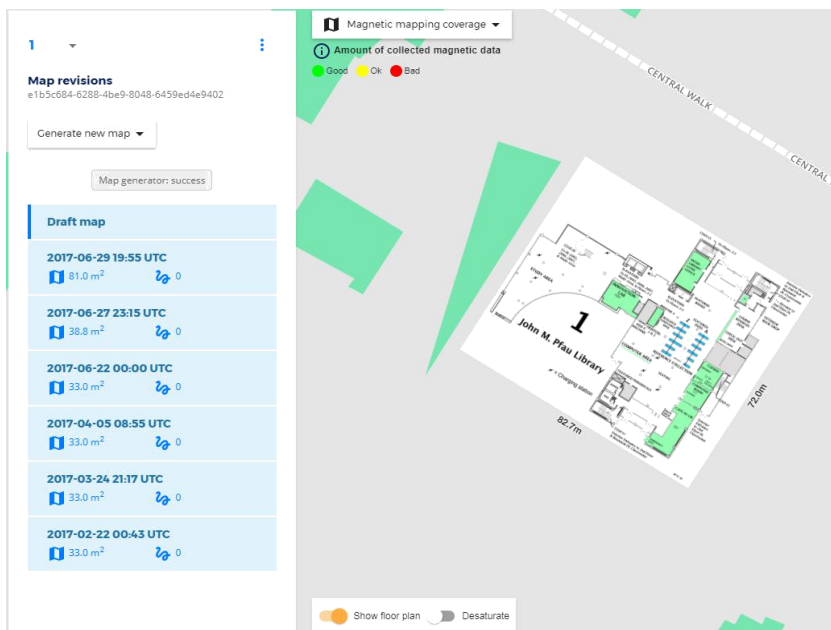


Figure 19. Floor plans added in the Project

Each floor has its unique floorId which is useful in our android code to distinguish between other Floor plans.



Figure 20. Floorplan of Floor 1

- Align Floorplans

After uploading the Floor plans we align them. The alignment needs to be perfect to obtain accurate mapping.

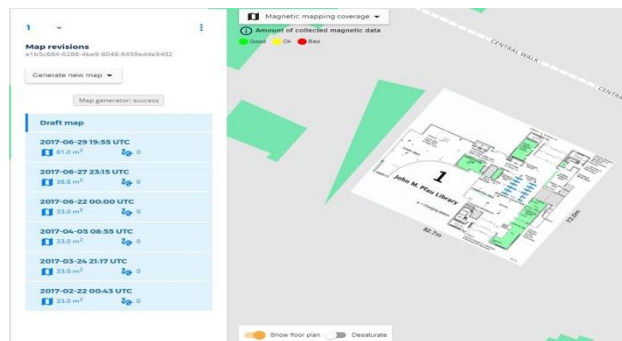


Figure 21. Aligning Floor Plans

- Adding Way points

After aligning the Floor plans we need to add Waypoints. Before starting the mapping you should plan the route you will walk to collect the data. The waypoints act as confirmation points for your data collection paths.

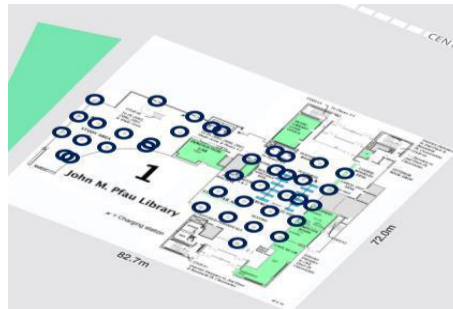


Figure 22. Adding Way points

- Check Mapping Area

After mapping the area on the Waypoints marked, we can review how the routes are formed in the Dashboard. If all the routes look correct you can generate a map.

- Get API keys and Secret id:

You should always create a new Application and API key for each new application you deploy with IndoorAtlas, in order to manage distribution and provide additional account security for your usage.

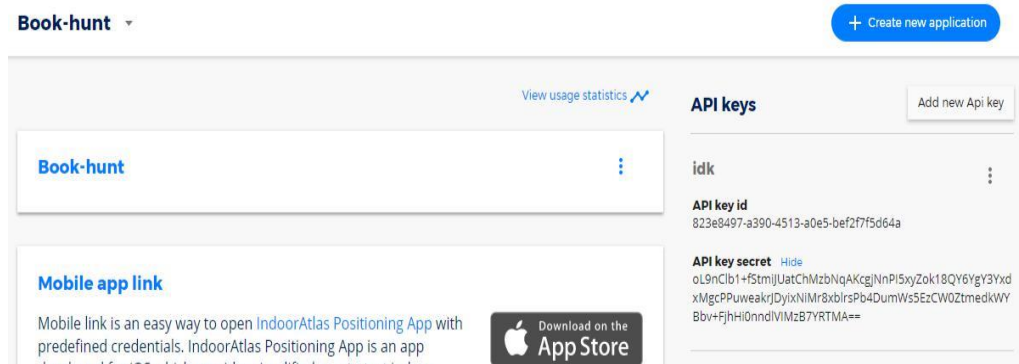


Figure 23. Applications in IndoorAtlas

You can Log out once you obtain the API keys. API key used for Book-Hunt is shown below.

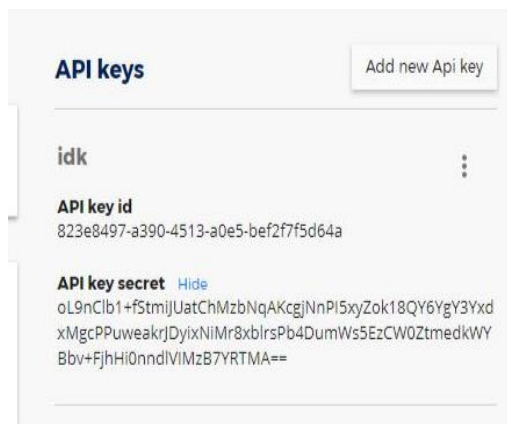


Figure 24. API key used for Book Hunt!

Data Flow Diagram of User

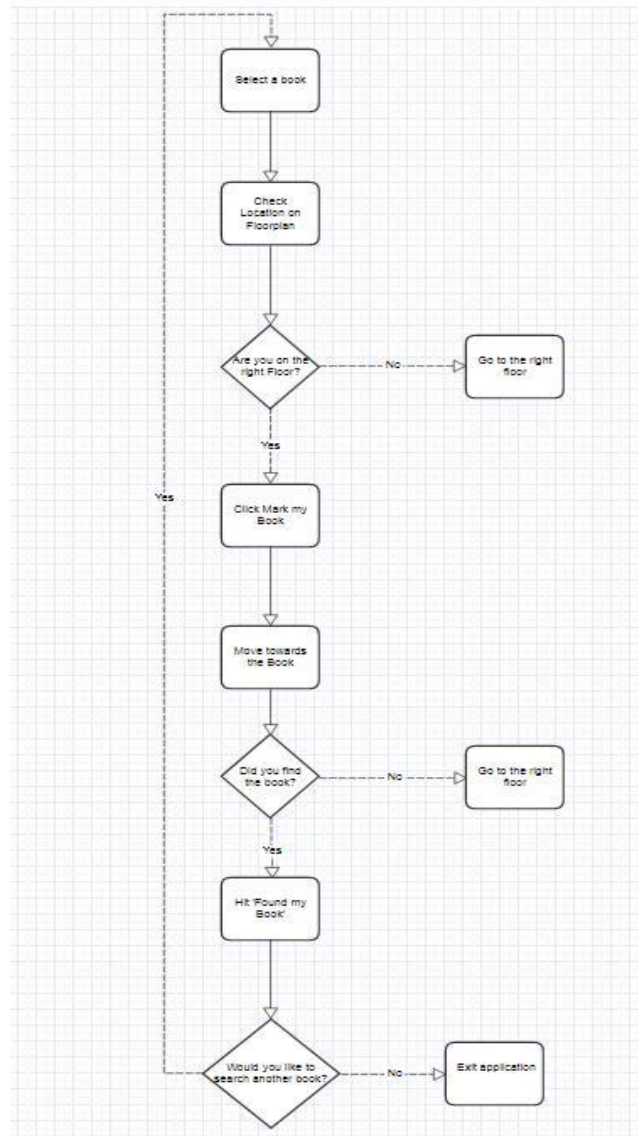
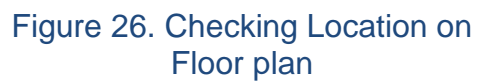


Figure 25. Data Flow Diagram of User

- First the Mobile app asks the user to select a book. There are five books in the List: Mercury, Venus, Earth, Mars and Jupiter.

- After the book is selected, you are redirected to the next page which displays the User's Location on the Floor Plan. The Floor Plan updates as you move from one floor to another. The user needs to check if his location is accurate on the Floor plan.



- Are you on the Right floor?

On the Second page of the Mobile application, a short message appears, which directs the user to the Floor, the book is located in. The user needs to go to the respective floor. If the user is on the right floor, he needs to click the button: Mark my book. An alert dialog box, confirms if you are on the right floor? If you are on the right floor, Click Yes or Click No.



Figure 27. Toast to direct student towards the book

If the user is on the right floor, he needs to click the button: Mark my book.

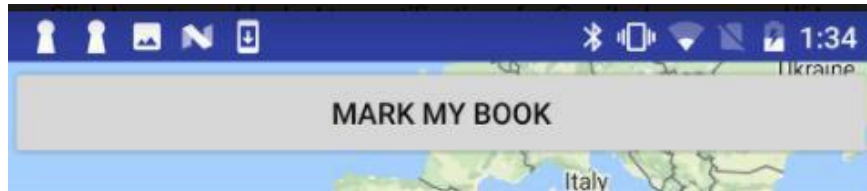


Figure 28. Mark my book button

An alert dialog box, confirms if you are on the right floor? If you are on the right floor, Click Yes or Click No.

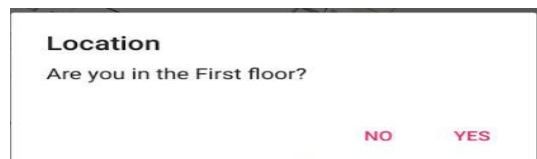


Figure 29. Alert box confirming floor

- Move towards the book

After Clicking the Yes button on the Alert dialog, you are navigated to another page in the app where a marker is placed on the Book's location. Note that, there are two markers on the floor plan, one which represents the user location and another which represents the book location. The marker of user's location is black and round in shape, whereas the marker of the book is red and in the shape of a book. A polyline between the two markers makes it easier for the user to move towards the book.

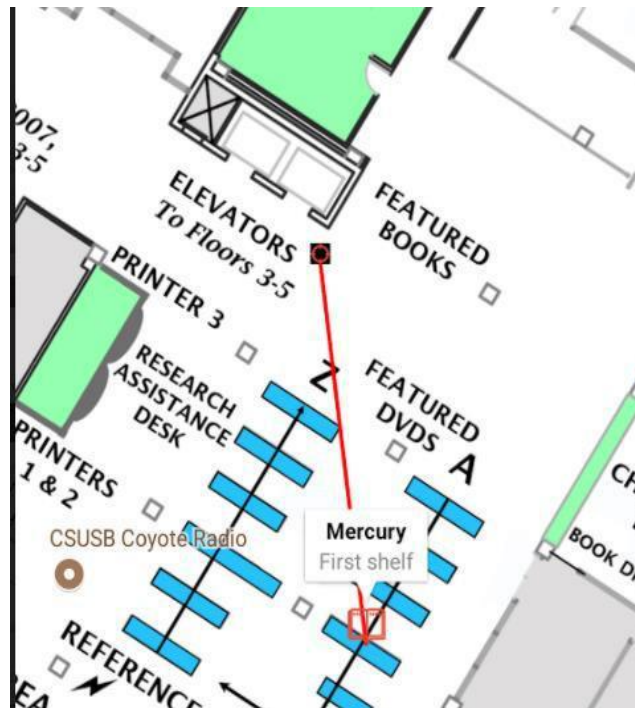


Figure 30. Polyline showing navigation

- Did you find the book?

If the user finds his desired book, he needs to click on the Button that says: Found my book. If he does not find his book, he needs to refresh app or click the Back button to check if he is on the right floor.



Figure 31. Found my book button

On clicking the Found my book button a dialog box appears, that thanks the users for using the Application.

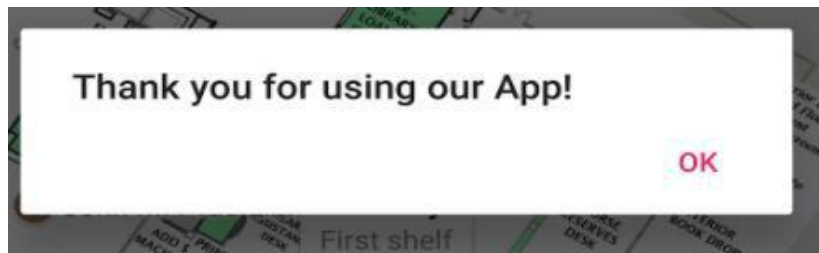


Figure 32. Thank you message

- Would you like to search for another book?

On Clicking the OK button, the users are redirected to the first page of the app that displays the drop down box of the books. If the user wishes to search for another book, he can go through the whole process again or he can Terminate the application.

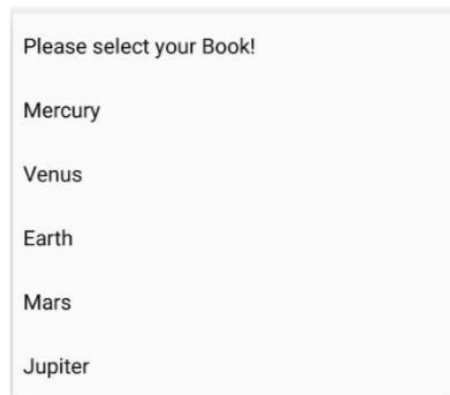
A screenshot of a mobile application interface. At the top, it says "Please select your Book!". Below this, there is a list of five items: Mercury, Venus, Earth, Mars, and Jupiter. Each item is preceded by a small, faint circular icon. The list is contained within a light gray rectangular box with a thin border.

Figure 33. To Search for another Book

CHAPTER SIX

UML DIAGRAMS

Functions of Admin- Use Case Diagram



Figure 34. Use case Diagram for Admin

This use- case diagrams describes the functions the Administrator while building the application. First the administrator needs to upload floor plans on the IndoorAtlas web application. Each floor plan has its own id. Once the floor plans have been uploaded they need to be mapped. The second function of the administrator is to map the location. Mapping is done using the MapCreator mobile application. The administrator needs to walk the whole area, until he has covered the whole area of the location. Once the mapping has been done, the administrator needs to integrate the SDK into the Android code. As mentioned earlier the IndoorAtlas SDK has a lot of features. It has a set of classes and interfaces that help us do a lot of functions like fetching locations, floor plans, geofences etc.

Functions of IndoorAtlas Web Application - Use Case Diagram



Figure 35. Use case diagram for IndoorAtlas Web application

IndoorAtlas Web Application plays a major role in this project as it allows us to add locations and floor plans and also manage data. Usage of IndoorAtlas web app in this project is a step by step process. First the location needs to be added. Next the floor plans need to be uploaded. The floor plans can be a .jpeg or .png files. All details of floor plan needs to be added like the floor number, altitude etc. The third step would be to align the floor plan, the floor plans can be aligned using coordinates or manually. Once the floor plans are aligned you need to add way points. This makes it easier to map the area. Finally, you need to generate your API key and secret, and use it in your android code. This API key and secret distinguishes one user from another, one location from another and one mobile application from another.

Functions of Google Map API- Use Case Diagram

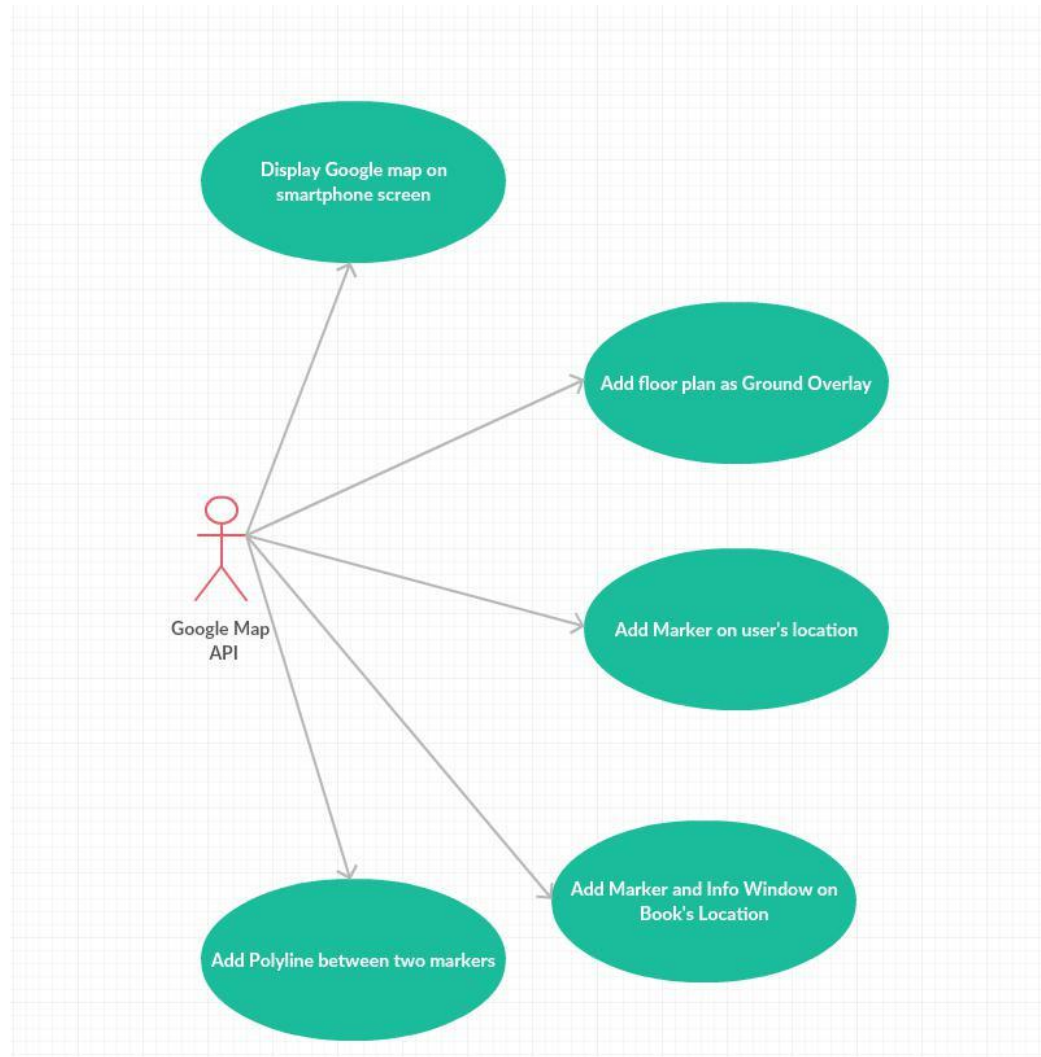


Figure 36. Use case Diagram for Google Maps API

Google Maps play a crucial role in this project. After integrating the IndoorAtlas SDK into the android code we need to integrate the google map SDK. The functions of Google Map API in this project are listed in the above figure. The API has its own inbuilt functions. *onMapReady()* function helps us display map on screen. *addGroundOverlay()* function helps us add floor plans as overlays on maps. *addMarker()* function helps us add markers on Book and User Location. *addPolyline()* function helps us add polyline between the two markers. *showInfoWindow()* lets us show the information window on the markers.

Functions of User- Use Case Diagram

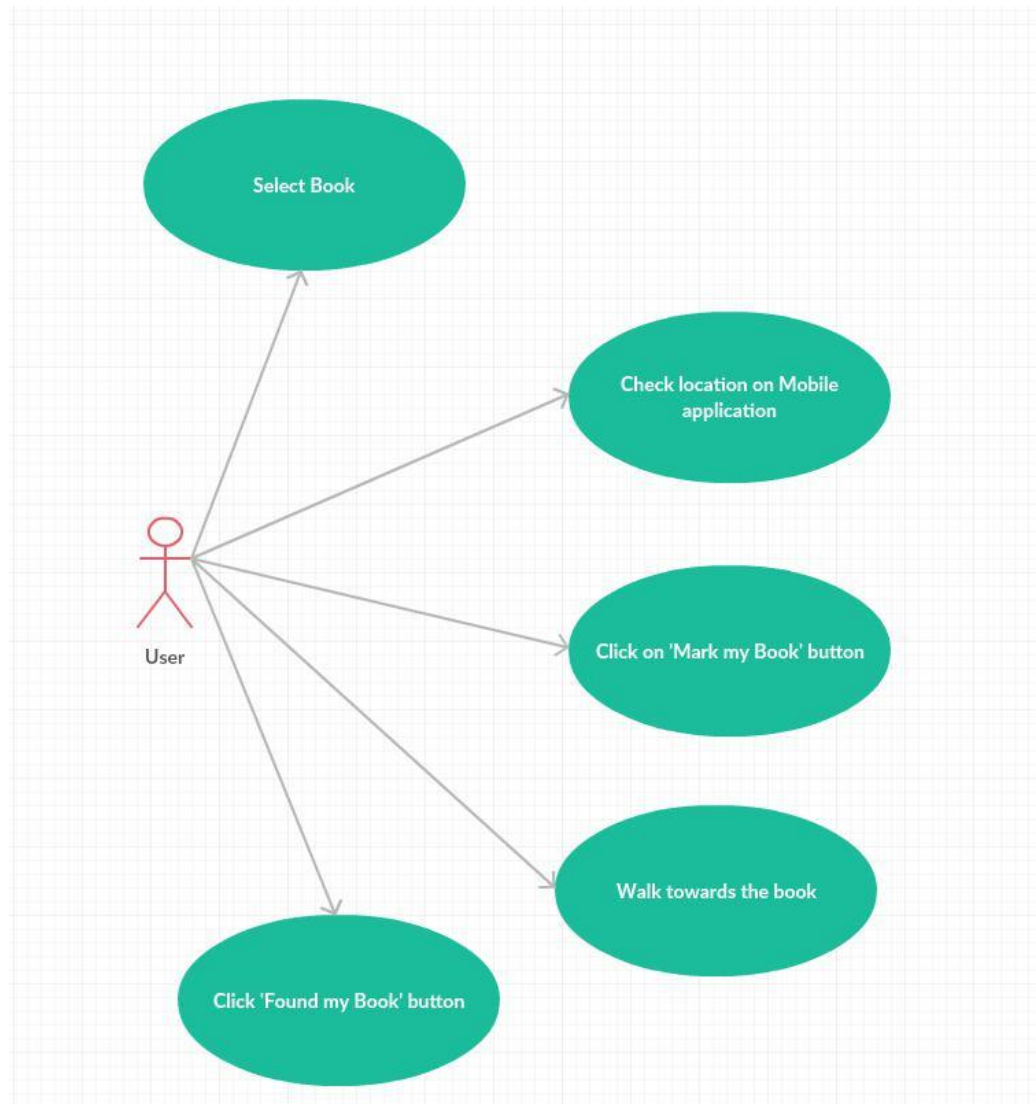


Figure 37. Use case diagram for User

This use-case diagram clearly shows how the user must use the app. Firstly the user needs to select a book on the Home Page. He is redirected to the second page where he needs to check his location, he needs to move to the appropriate floor as prompted. After he reaches the right floor he needs to click on Mark my Book button. On the third page a marker is added on the location of the book, with a polyline showing the route towards the book from the user's current location. Once the user finds his book he needs to click on the button 'Found my Book'. He is now again redirected to the home page in case he needs to find another book.

Indooratlas SDK Package-Class Diagram

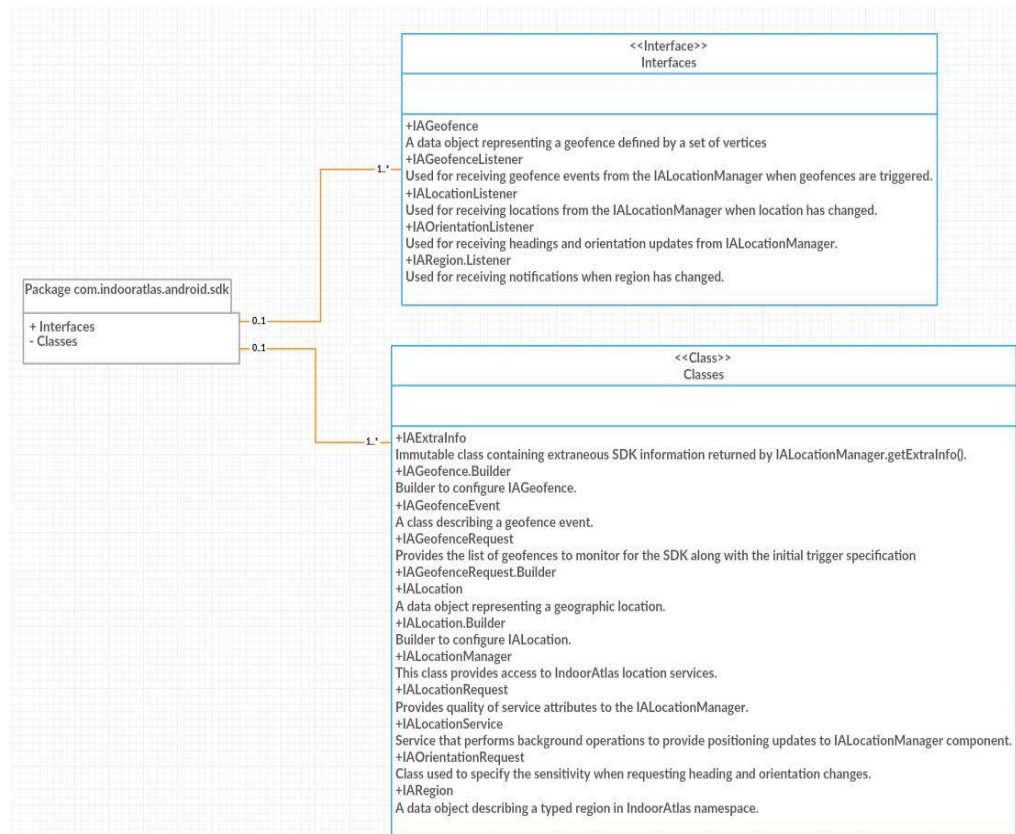


Figure 38. Class Diagram for IndoorAtlas Sdk Package

The above class diagram clearly shows all the interfaces and classes available in the IndoorAtlas package. Each interface and class is associated with a function. These classes help us write the android code using the SDK. For example: IALocationRequest. This class requests the IALocationManager for location updates from the cloud.

Interaction Between Mobile App And Server-Sequence Diagram

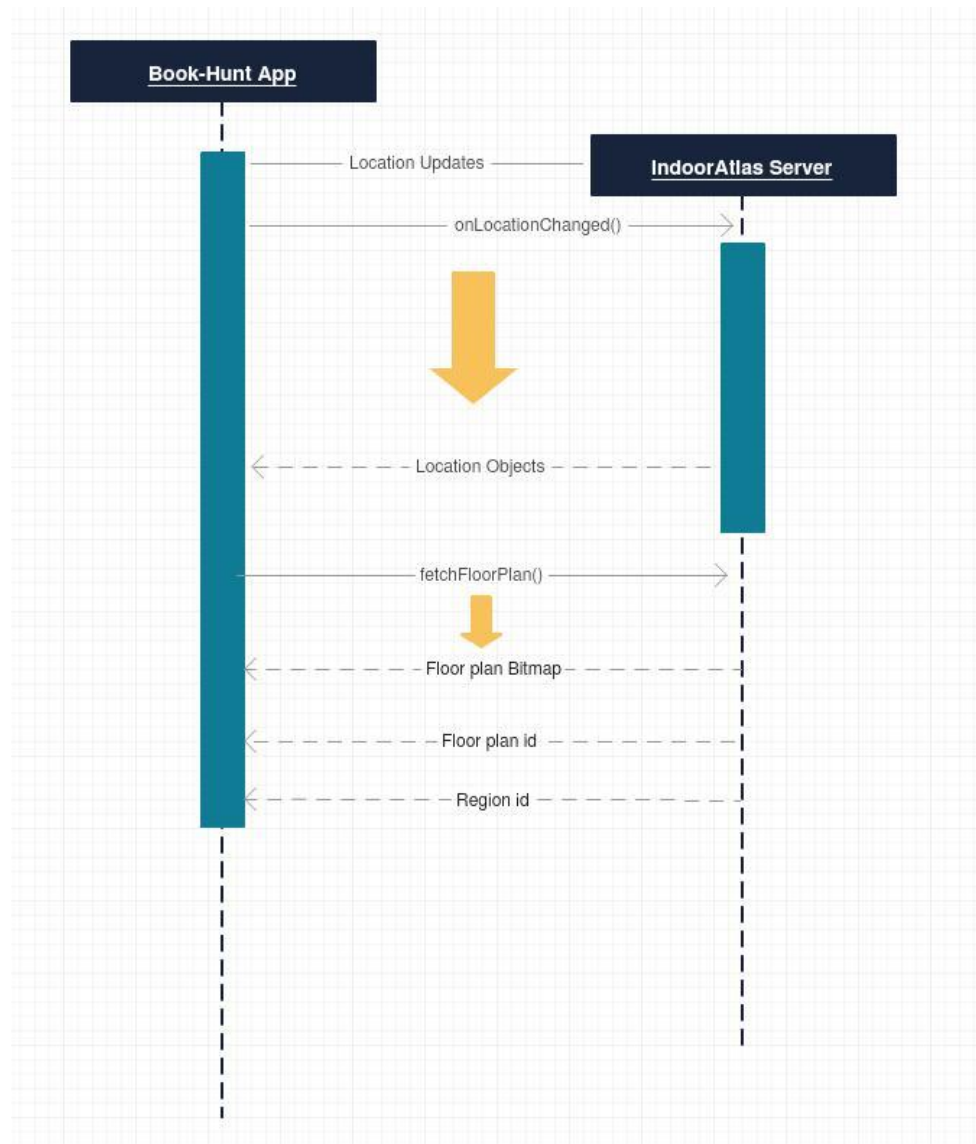


Figure 39. Sequence Diagram to show Interaction between Mobile Application and Server

This sequence diagram shows the relation of the Mobile application to the IndoorAtlas server. The application mainly accesses the cloud to get the location updates of the user. As the user is continuously moving the current location updates rapidly. Henceforth a lot of data is passed from Cloud to the mobile application. The SDK also fetches other data like floor id and floor plans.

CHAPTER SEVEN

PROJECT DEVELOPMENT

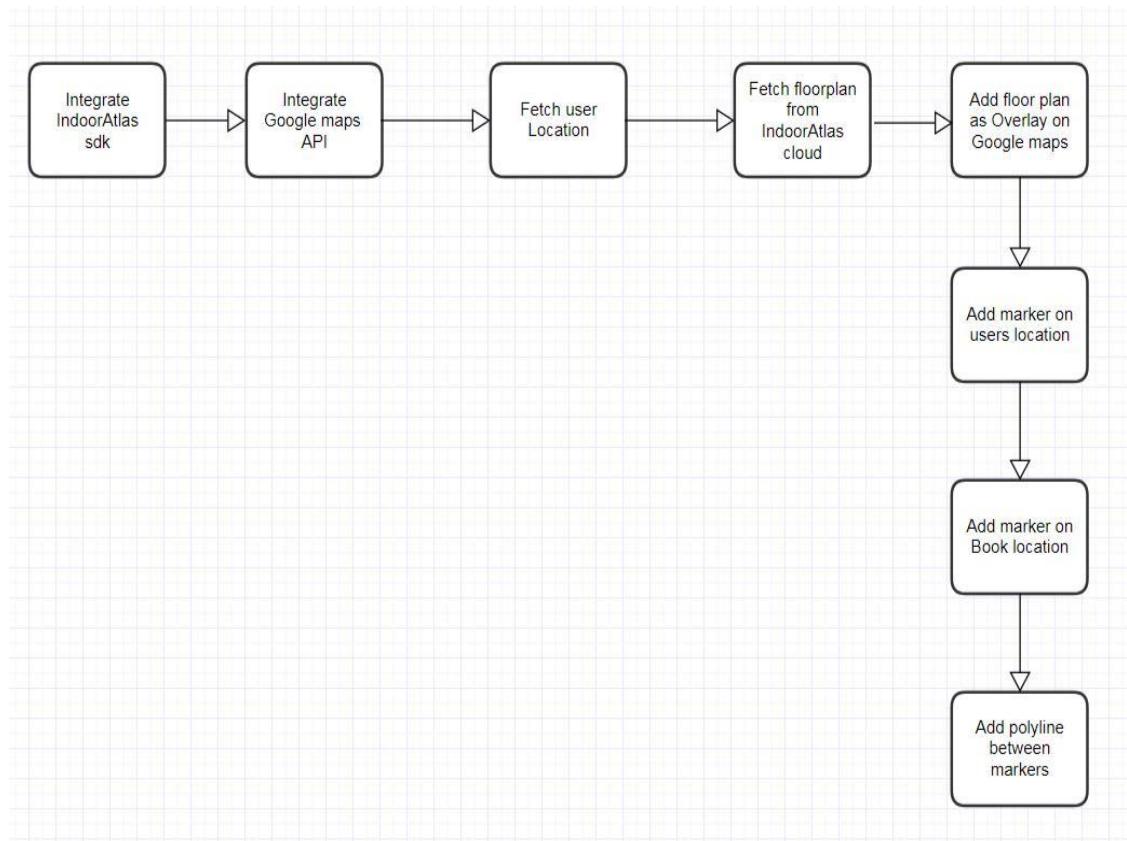


Figure 40. Data Flow Diagram for Project Development

Description

Integrate IndoorAtlas SDK

The first step in the Development of App is to integrate the IndoorAtlas SDK. IndoorAtlas SDK is the core element of this Project. The IndoorAtlas SDK is added in the build.gradle file of the Android application. It is through this SDK that we can fetch user's location inside a building. [5]

Add this to your build.gradle file.

```
dependencies {  
    compile 'com.indooratlas.android:indooratlas-android-sdk:  
{{site.versions.androidSdkVersion}}@aar'  
}  
repositories{  
    maven {  
        url "http://indooratlas-ltd.bintray.com/mvn-public"  
    }  
}
```

Figure 41. Adding Dependencies in Android Project

To use the IndoorAtlas SDK we need to add Sensors and user-permissions. The sensors and user permissions are added ion the Android manifest file

```

<uses-feature android:name="android.hardware.sensor.accelerometer"
    android:required="true" />
<uses-feature android:name="android.hardware.sensor.compass"
    android:required="true" />
<uses-feature android:name="android.hardware.sensor.gyroscope"
    android:required="true" />
<uses-feature android:name="android.hardware.wifi"
    android:required="true" />

<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

```

Figure 42. Adding sensors and Permissions to Android Manifest file

IndoorAtlas SDK has its own API key and secret. This unique API key and secret distinguishes applications inside the IndoorAtlas cloud. This API key and secret are also added to the Android Manifest file.

```

<application>
    <meta-data
        android:name="com.indooratlas.android.sdk.API_KEY"
        android:value="api-key-here"/>
    <meta-data
        android:name="com.indooratlas.android.sdk.API_SECRET"
        android:value="api-secret-here"/>
</application>

```

Figure 43. Syntax to add API key in Android Manifest file

```
<meta-data
    android:name="com.indooratlas.android.sdk.API_KEY"
    android:value="823e8497-a390-4513-a0e5-bef2f7f5d64a" />
<meta-data
    android:name="com.indooratlas.android.sdk.API_SECRET"
    android:value="oL9nC1b1+f5tmiJUatChMzbNqAKcgjNnPI5xyZok18QY6YgY3YxdxMgcPPuweakrJDyixNlMr8xb1rsPb4Duml5SEzCh0ZtmedklnYBbv+F
```

Figure 44. Adding API key and secret to Android Manifest file

Integrate Google Map API

The second step in the Development of app is to integrate Google Map API. This API enables us to display Google Maps on the Mobile Screen. Google Maps API also has its own API key which distinguishes users. This API helps us use the Google Maps features like Marker, Overlay etc. [5]

First we need to add the dependencies in the build.gradle file of the app

```
dependencies {  
    compile 'com.google.android.gms:play-services-maps:+'  
    compile 'com.android.support:appcompat-v7:25.+'  
}
```

Figure 45. Adding Google Map dependencies

Next, we need to add the API key in the Android Manifest file. API key is very essential for this Mobile Application.

```
<meta-data  
    android:name="com.google.android.geo.API_KEY"  
    android:value="AIzaSyAXAv1_6xcPLyq4CyExbQEfvplDNGP5i7s" />
```

Figure 46. Adding API key to Manifest file

Fetch User's Location

The third step in the Development of app is to Fetch User's Location. To get Location Updates on Android 6+ phones we need to add Runtime Permissions. Runtime permissions specially asks the user to turn on Location, Bluetooth and Wi-Fi. [5]

Syntax:

```
private final int CODE_PERMISSIONS = //...

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    String[] neededPermissions = {
        Manifest.permission.CHANGE_WIFI_STATE,
        Manifest.permission.ACCESS_WIFI_STATE,
        Manifest.permission.ACCESS_COARSE_LOCATION
    };
    ActivityCompat.requestPermissions( this, neededPermissions, CODE_PERMISSIONS );
}

//...

@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    //Handle if any of the permissions are denied, in grantResults
}
```

Figure 47. Runtime Permissions

Initialization:

```
private void ensurePermissions() {

    if (ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION)
        != PackageManager.PERMISSION_GRANTED)
    {

        // we don't have access to coarse locations, hence we have not access to wifi either
        // check if this requires explanation to user
        if (ActivityCompat.shouldShowRequestPermissionRationale(this,
            Manifest.permission.ACCESS_FINE_LOCATION) ){

            new AlertDialog.Builder(this)
                .setTitle(R.string.location_permission_request_title)
                .setMessage(R.string.location_permission_request_rationale)
                .setPositiveButton(R.string.permission_button_accept, new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        Log.d(TAG, "request permissions");
                        ActivityCompat.requestPermissions(MainActivity.this,
                            new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                            REQUEST_CODE_ACCESS_COARSE_LOCATION);
                    }
                })
                .setNegativeButton(R.string.permission_button_deny, new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        Toast.makeText(MainActivity.this,
                            R.string.location_permission_denied_message,
                            Toast.LENGTH_LONG).show();
                    }
                })
                .show();

        } else {
            Log.d(TAG, "asking permissions to user");

            // ask user for permission
            ActivityCompat.requestPermissions(this,
                new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                REQUEST_CODE_ACCESS_COARSE_LOCATION);
        }
    }
}

@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {

    switch (requestCode) {
        case REQUEST_CODE_ACCESS_COARSE_LOCATION:

            if (grantResults.length > 0
                || grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                Toast.makeText(this, "Permission granted thank you",
                    Toast.LENGTH_LONG).show();
            }

            break;
    }
}
```

Figure 48. Code for Runtime Permissions

IndoorAtlas SDK has its class known as '*IALocationManager*' which provides access to the services of IndoorAtlas. In an Android coding the method `onCreate ()` acts as the main method. An instance of *IALocationManager* is created in the `on Create ()` method of the Activity.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mIALocationManager = IALocationManager.create(this);
}
```

Figure 49. Calling *IALocationManager* in `onCreate()`

After creating an instance of *IALocationManager* you can now get periodic updates of the user's location. The periodic updates of the user's location is generated by the *IALocationListener* interface which has a `onLocationChanged()` callback method. `onLocationChanged` method updates the user's location as he moves

```

private ILocationListener mILocationListener = new ILocationListener() {

    // Called when the location has changed.
    @Override
    public void onLocationChanged(ILocation location) {

        Log.d(TAG, "Latitude: " + location.getLatitude());
        Log.d(TAG, "Longitude: " + location.getLongitude());
        Log.d(TAG, "Floor number: " + location.getFloorLevel());
    }
};

```

Figure 50. Fetching Location Updates

To stop receiving updates we write *removeLocationUpdates()* method

```

@Override
protected void onPause() {
    super.onPause();
    mILocationManager.removeLocationUpdates(mILocationListener);
}

```

Figure 51. To stop Receiving Location Updates

Fetch Floor Plan from IndoorAtlas Cloud

Now that we get Location updates, we need to fetch floor plan from the IndoorAtlas Cloud. The IndoorAtlas SDK provides an API for fetching the floor plan images that is stored in the cloud while mapping phase.

- An *ImageView* instance is created in the application layout resource file. It will hold the bitmap image.
- The *IAFloorPlan* class represents floor plan data. It has a URL field pointing to the floor plan image.
- The *IAResourceManager* class provides an interface to fetch *IAFloorPlan* objects from IndoorAtlas services. [2]

```

private void fetchFloorPlan(String id) {
    // Cancel pending operation, if any
    if (mPendingAsyncResult != null && !mPendingAsyncResult.isCancelled()) {
        mPendingAsyncResult.cancel();
    }

    mPendingAsyncResult = mResourceManager.fetchFloorPlanWithId(id);
    if (mPendingAsyncResult != null) {
        mPendingAsyncResult.setCallback(new IAsyncResultCallback<IAFloorPlan>() {
            @Override
            public void onResult(IAsyncResult<IAFloorPlan> result) {
                Logger.d(TAG, "onResult: %s", result);

                if (result.isSuccess()) {
                    handleFloorPlanChange(result.getResult());
                } else {
                    // do something with error
                    Toast.makeText(FloorPlanManagerActivity.this,
                        "loading floor plan failed: " + result.getError(),
                        Toast.LENGTH_LONG)
                        .show();
                }
            }
        }, Looper.getMainLooper()); // deliver callbacks in main thread
    }
}

```

Figure 52. Fetch floor plan code

We use third party library Picasso to easily download the image from cloud.

```

import com.squareup.picasso.Picasso;
// ...
private void handleFloorPlanChange(IAFloorPlan newFloorPlan) {
    Picasso.with(this)
        .load(newFloorPlan.url)
        .into(mFloorPlanImage);
}

```

Figure 53. Picasso download floor plan image code

Floor Detection

The IndoorAtlas SDK automatically recognizes floor plans, locations (a.k.a. venues) and indoor-outdoor-transitions with *IARegion* events.

```
private IARegion.Listener mRegionListener = new IARegion.Listener() {  
    // when null, we are not on any mapped area  
    // this information can be used for indoor-outdoor detection  
    IARegion mCurrentFloorPlan = null;  
  
    @Override  
    public void onEnterRegion(IARegion region) {  
        if (region.getType() == IARegion.TYPE_FLOOR_PLAN) {  
            // triggered when entering the mapped area of the given floor plan  
            Log.d(TAG, "Entered " + region.getName());  
            Log.d(TAG, "floor plan ID: " + region.getId());  
            mCurrentFloorPlan = region;  
        }  
        else if (region.getType() == IARegion.TYPE_VENUE) {  
            // triggered when near a new location  
            Log.d(TAG, "Location changed to " + region.getId());  
        }  
    }  
  
    @Override  
    public void onExitRegion(IARegion region) {  
        // leaving a previously entered region  
        if (region.getType() == IARegion.TYPE_FLOOR_PLAN) {  
            mCurrentFloorPlan = null;  
            // notice that a change of floor plan (e.g., floor change)  
            // is signaled by an exit-enter pair so ending up here  
            // does not yet mean that the device is outside any mapped area  
        }  
    }  
};
```

Region listeners are also registered to the `IALocationManager`

```
mIALocationManager.registerRegionListener(mRegionListener);
```

Figure 54. Region detection code

Add Floor Plan as an Overlay on Google Maps

Ground overlays are image overlays that are tied to latitude/longitude coordinates, so they move when you drag or zoom the map. We use the floor plan bitmap as a ground overlay on the map. This lets the user check location on the Floor plan.

```
private IRegion.Listener mRegionListener = new IRegion.Listener() {

    @Override
    public void onEnterRegion(IRegion region) {
        if (region.getType() == IRegion.TYPE_FLOOR_PLAN) {
            final String newId = region.getId();
            // Are we entering a new floor plan or coming back the floor plan we just left?
            if (mGroundOverlay == null || !region.equals(mOverlayFloorPlan)) {
                mCameraPositionNeedsUpdating = true; // entering new fp, need to move camera
                if (mGroundOverlay != null) {
                    mGroundOverlay.remove();
                    mGroundOverlay = null;
                }
                mOverlayFloorPlan = region; // overlay will be this (unless error in loading)
                fetchFloorPlan(newId);
            }
        } else {
            mGroundOverlay.setTransparency(0.0f);
        }
    }
}

Log.d(TAG, "Enter " + (region.getType() == IRegion.TYPE_VENUE
    ? "VENUE "
    : "FLOOR_PLAN ") + region.getId());
}
```

Figure 55. Adding floor plan as Overlay

```

@Override
public void onExitRegion(IRegion region) {
    if (mGroundOverlay != null) {
        // Indicate we left this floor plan but leave it there for reference
        // If we enter another floor plan, this one will be removed and another one loaded
        mGroundOverlay.setTransparency(0.5f);
    }
    Log.d(TAG, "Enter " + (region.getType() == IRegion.TYPE_VENUE
        ? "VENUE "
        : "FLOOR_PLAN ") + region.getId());
}

};

```

Figure 56. Exit region

Add Marker on User's Location

The next step is to store the Latitude and Longitude of the user's location in a variable example: `latLng` and add a marker on that particular location.

Marker's on Google Maps indicate single locations. As the Location updates, the value in the location variable (`latLng`) also updates which in turn updates the location of Marker. Google Maps API has a functionality which updates the Camera position, as the marker location of user gets updated. [3]

```
if (mMap == null) {
    // location received before map is initialized, ignoring update here
    return;
}

LatLng latLng = new LatLng(location.getLatitude(), location.getLongitude()); /*Location of user from IA SDK*/

if (mMarker == null) {
    // first location, add marker
    mMarker = mMap.addMarker(new MarkerOptions().position(latLng)
        .icon(BitmapDescriptorFactory.fromResource(R.mipmap.icon2)));
} else {
    // move existing markers position to received location
    mMarker.setPosition(latLng);
}

// our camera position needs updating if location has significantly changed
if (mCameraPositionNeedsUpdating) {
    mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(latLng, 19));
    mCameraPositionNeedsUpdating = false;
}
```

Figure 57. Adding Marker

After fetching Locations, floor plans and adding marker on User's Location the output of the Mobile Application looks like this below.



Figure 58. Marker in Mobile application

Adding a Marker on the Book's Location

After displaying the location of user, we now display the location of the Book. Each book has been assigned a waypoint in the IndoorAtlas Cloud. The waypoints are renamed as the names of books. [3]

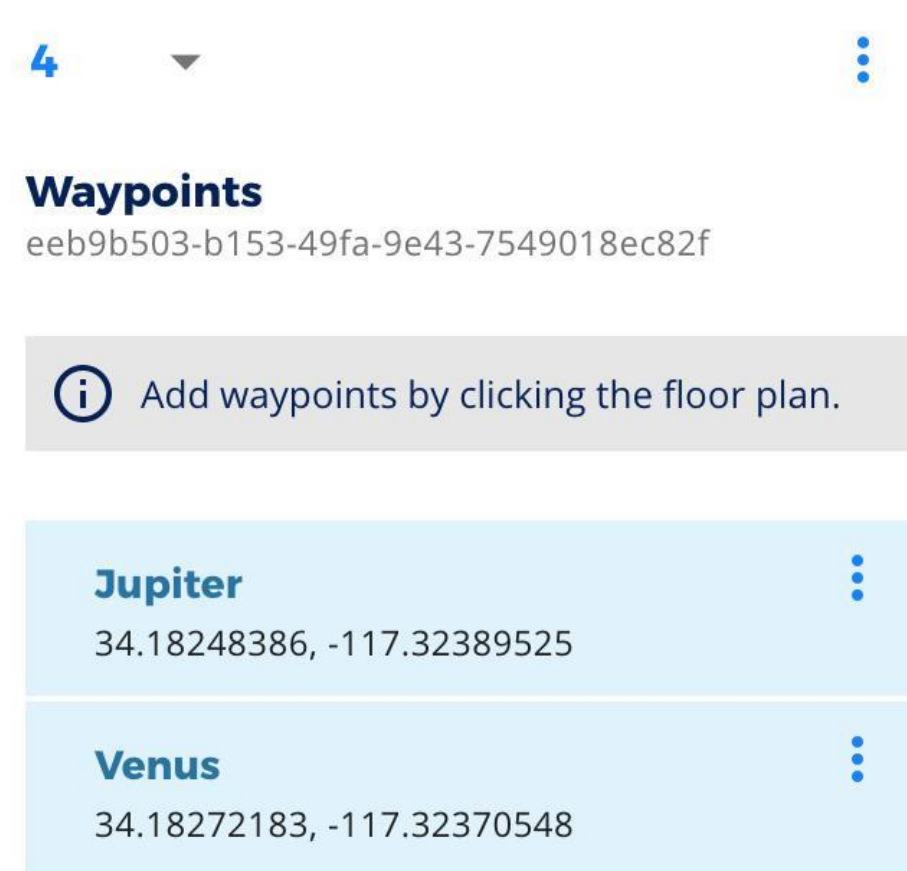


Figure 59. Book Waypoints in IndoorAtlas web app

In the Android code, each book has been assigned its own Activity, for example: MercuryActivity.java.

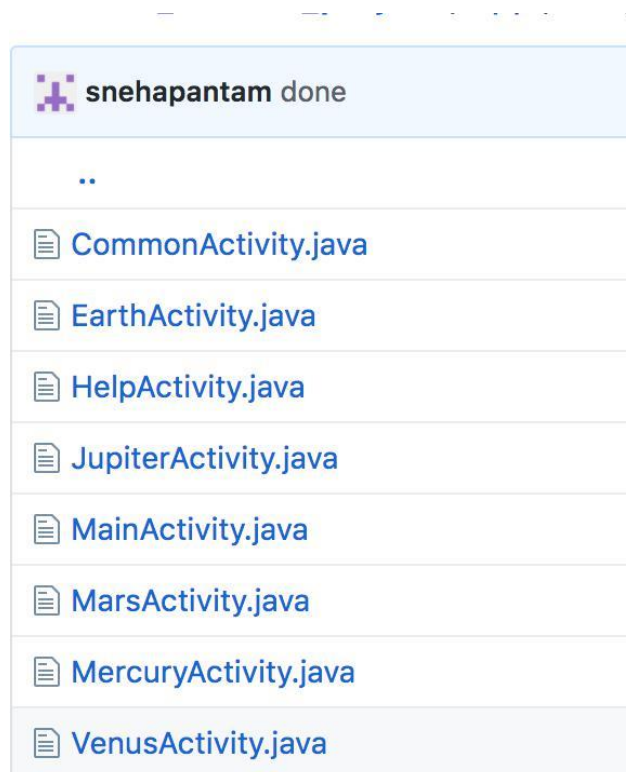


Figure 60. Individual Activities for books

Each Activity has a variable which stores its own Book Location. For Example, MercuryActivity.java stores variable mercury which contains the coordinates of book Mercury.

```
LatLng mercury_book= new LatLng(34.18249717, -117.32380673);
```

Figure 61. Waypoint Book Location

For Each book location a marker is set which points the Book Location. The marker of the book is red in color and it has a unique symbol of a 'book'. This marker which represents the location of the book has an *Info Window* which shows information about the position of book in the shelf, example: Third shelf. In Google Maps, an Info Window displays text or images in a popup window above the map. They are always attached to a Marker.

```
Marker mMarker1 = mMap.addMarker(new MarkerOptions().position(mercury_book).title("Mercury").snippet("First shelf")
    .icon(BitmapDescriptorFactory.fromResource(R.mipmap.book_icon)));

mMarker1.showInfoWindow();
polylinemethod();
```

Figure 62. Adding Marker on Book's Location

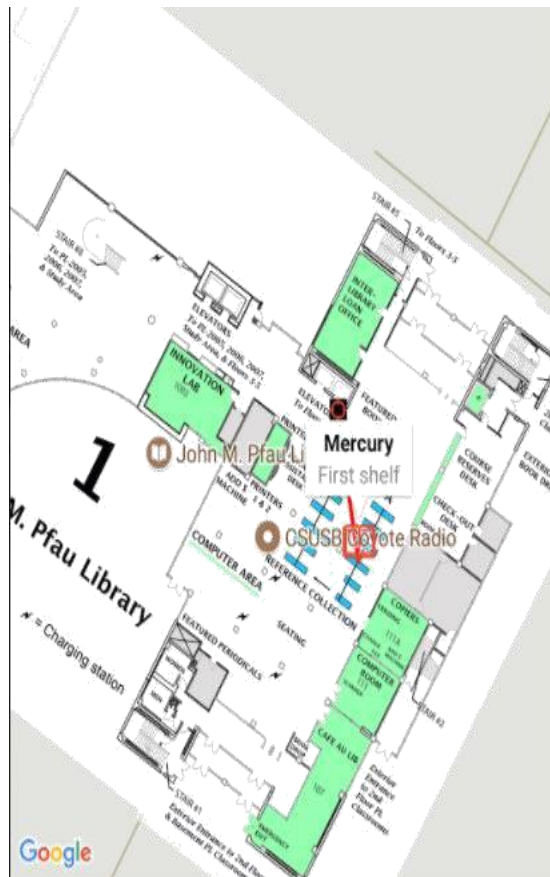


Figure 63. Book Marker on Mobile application

Adding Polyline between Markers.

Google Map API provides us with an exclusive polyline feature which helps us to see the straight line route from User's location to Book Location. The Polyline used in this Mobile application is a Red line between both the Markers.

[3]

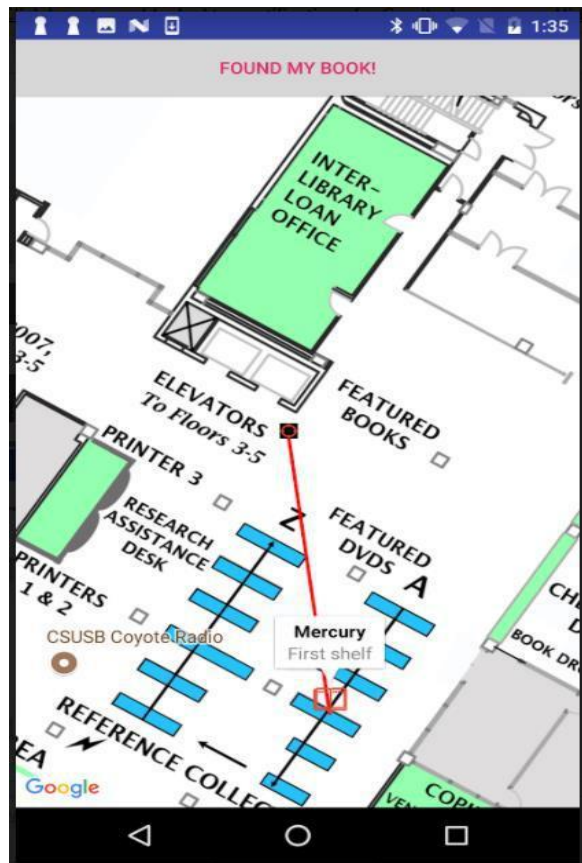


Figure 64. Polyline in Mobile application

CHAPTER EIGHT

LIMITATIONS OF BOOK-HUNT!

Some Limitations of this Mobile application Book-Hunt are

1. Inaccurate:

As the area is mapped using the Wi-Fi and Bluetooth signals manually by a person, the user's location cannot be as accurate as possible. Accuracy can be attained if a team of people map the whole area having 100% Wi-Fi signal. [2]

2. Specific Location:

This mobile app works only in John M. Pfau Library CSUSB. The location is added and all the floor plans have been aligned only to this one location. IndoorAtlas allows us to add only one location per mobile app. [1]

3. Elevator problems

When a student moves in an elevator, due to lack of signal the Wi-Fi and Bluetooth turn off. Wi-Fi and Bluetooth are very necessary for the application to work. Henceforth a refresh button is setup to refresh the layout each time the student moves out of an Elevator. [2]

CHAPTER NINE

PROJECT SCREENS

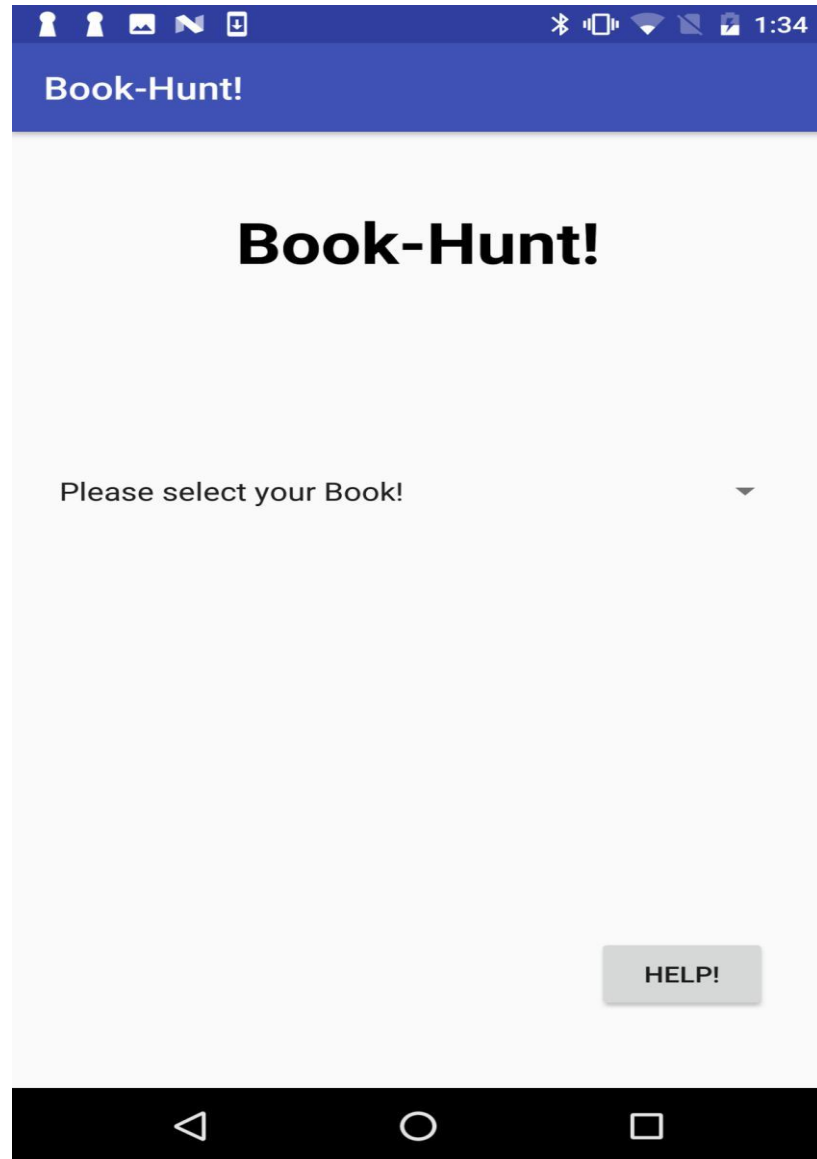


Figure 65. Screen of First Page

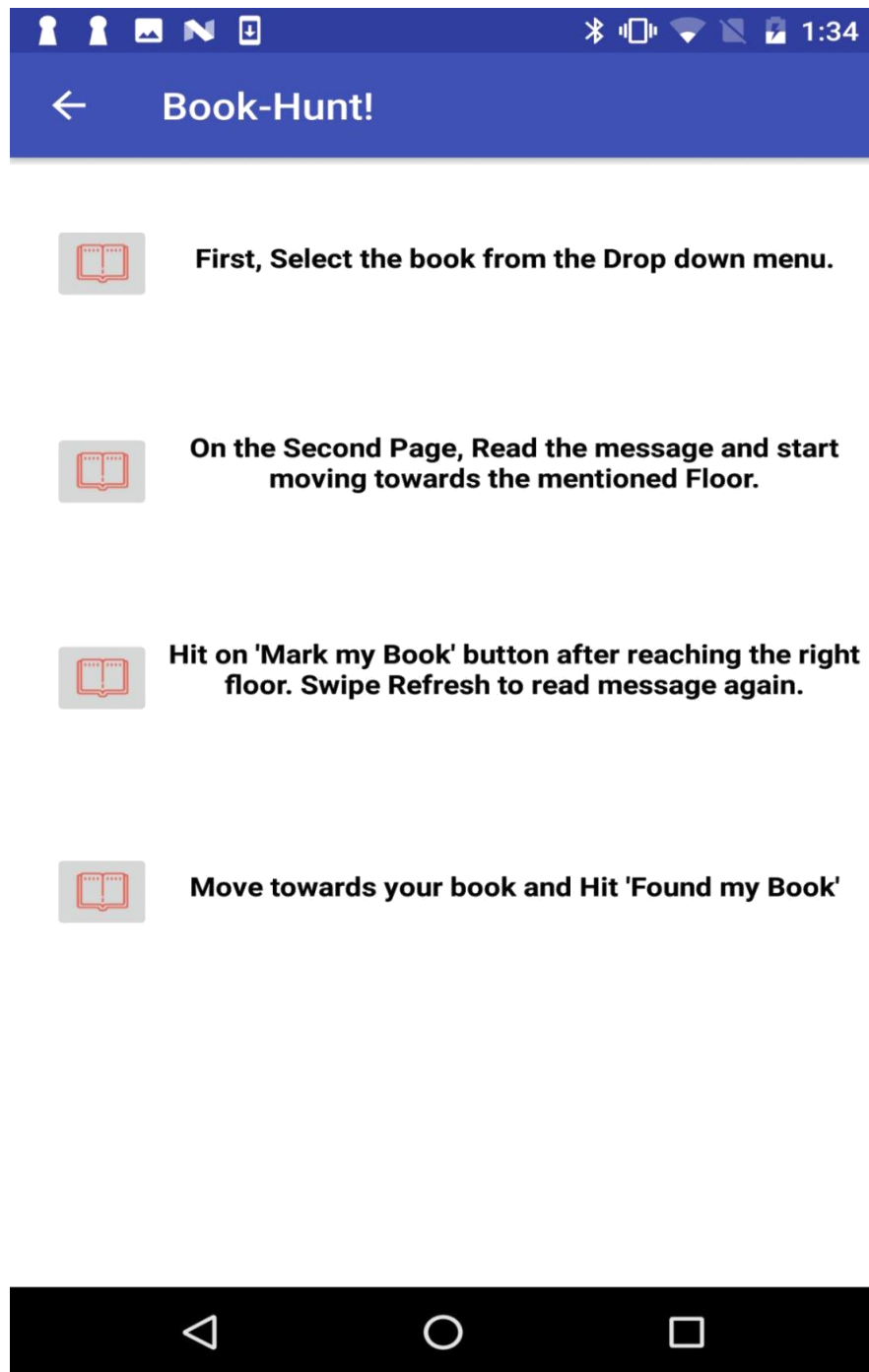


Figure 66. Screen of Help Page

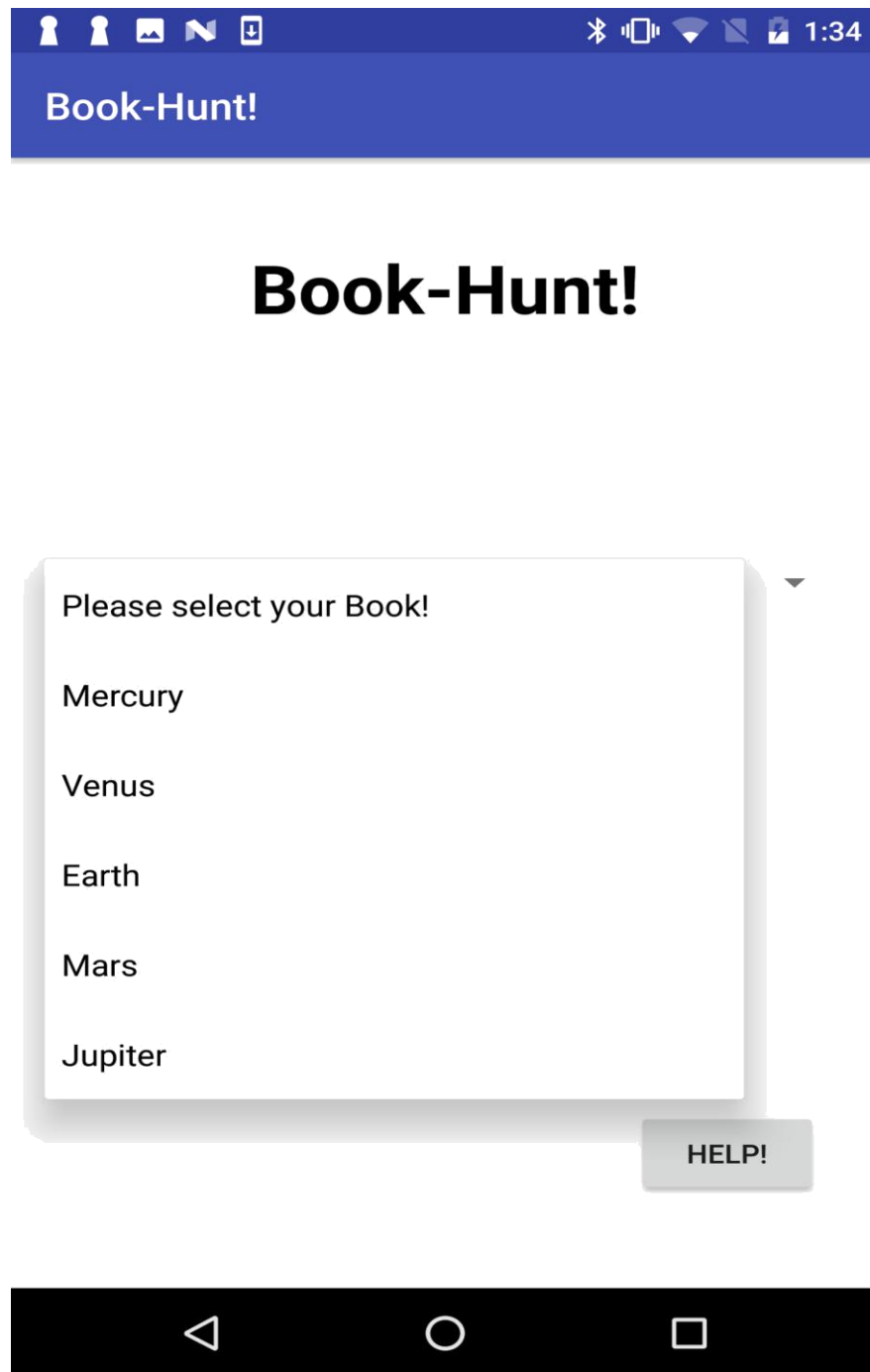


Figure 67. Screen of First Page- Select a Book

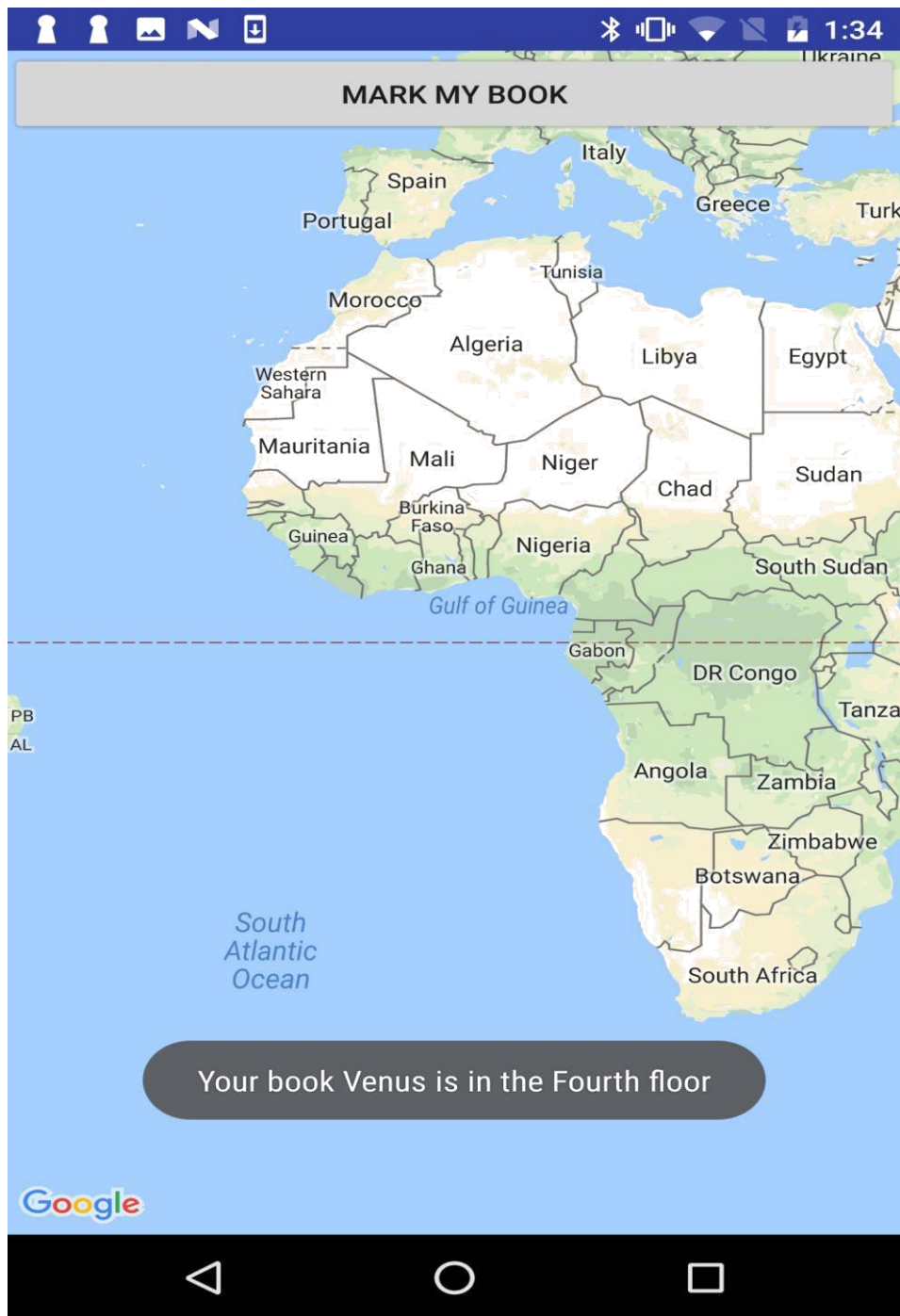


Figure 68. Screen of Second Page- Toast to display floor number

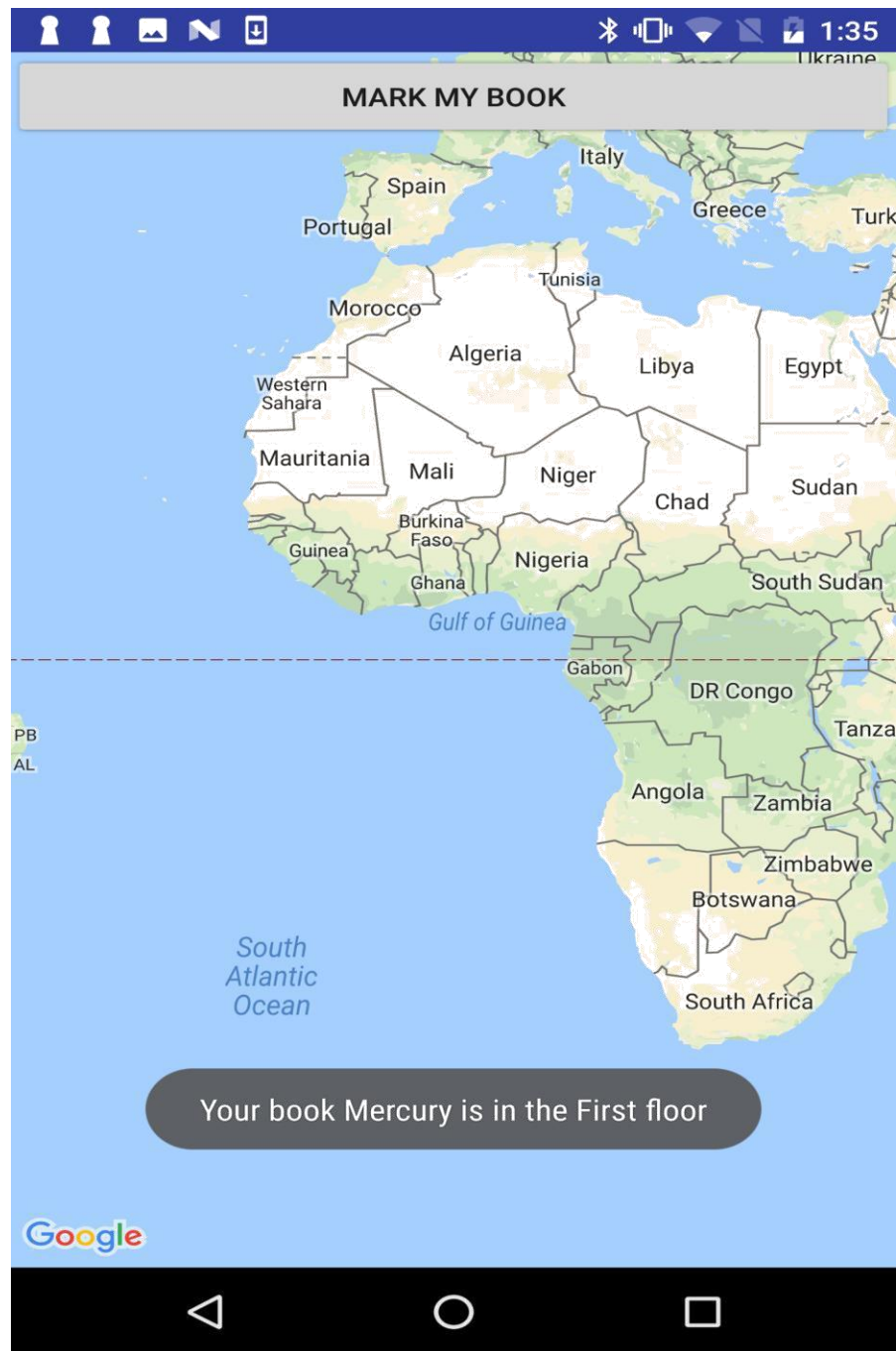


Figure 69. Screen of Second Page

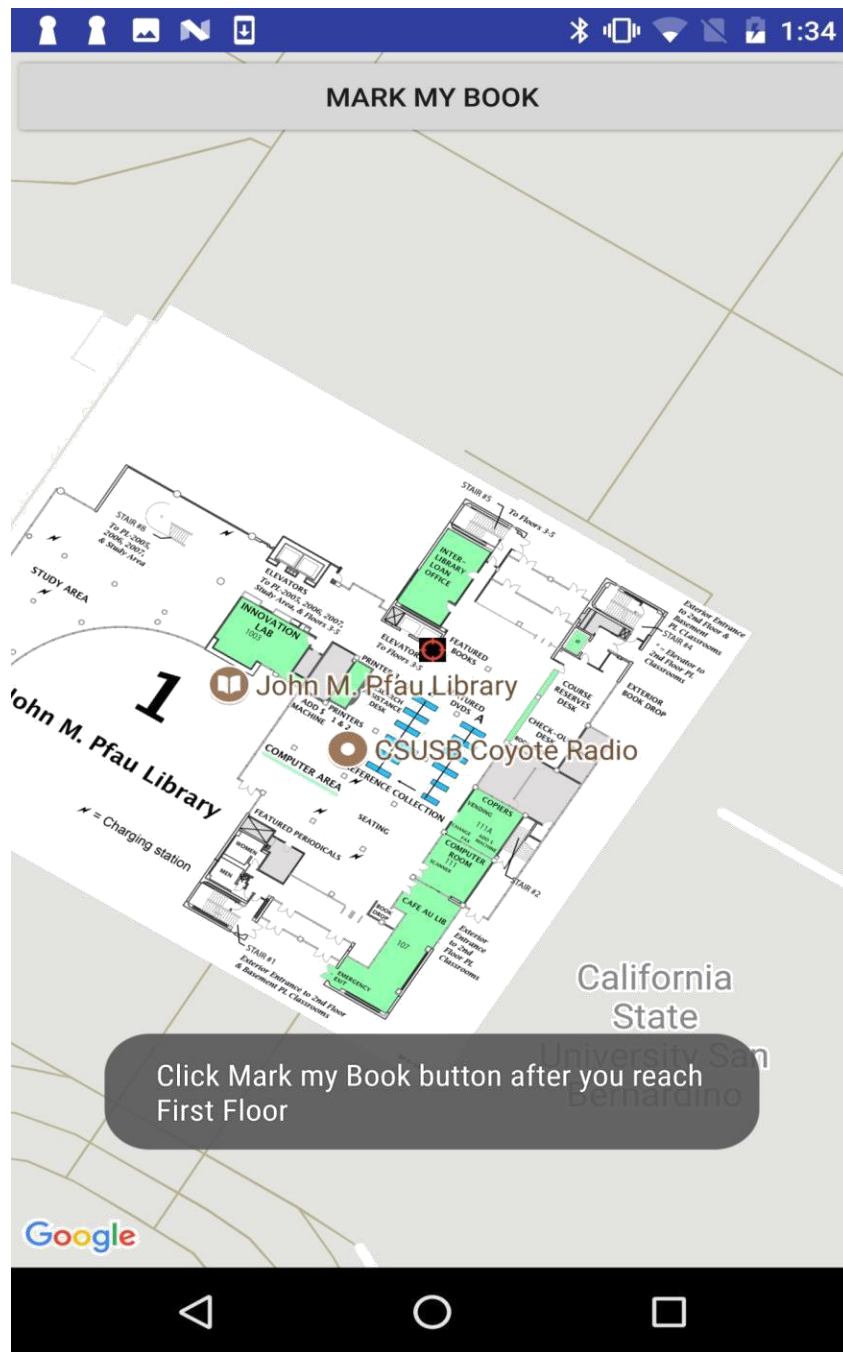


Figure 70. Screen of Second Page- Marker on User's Location

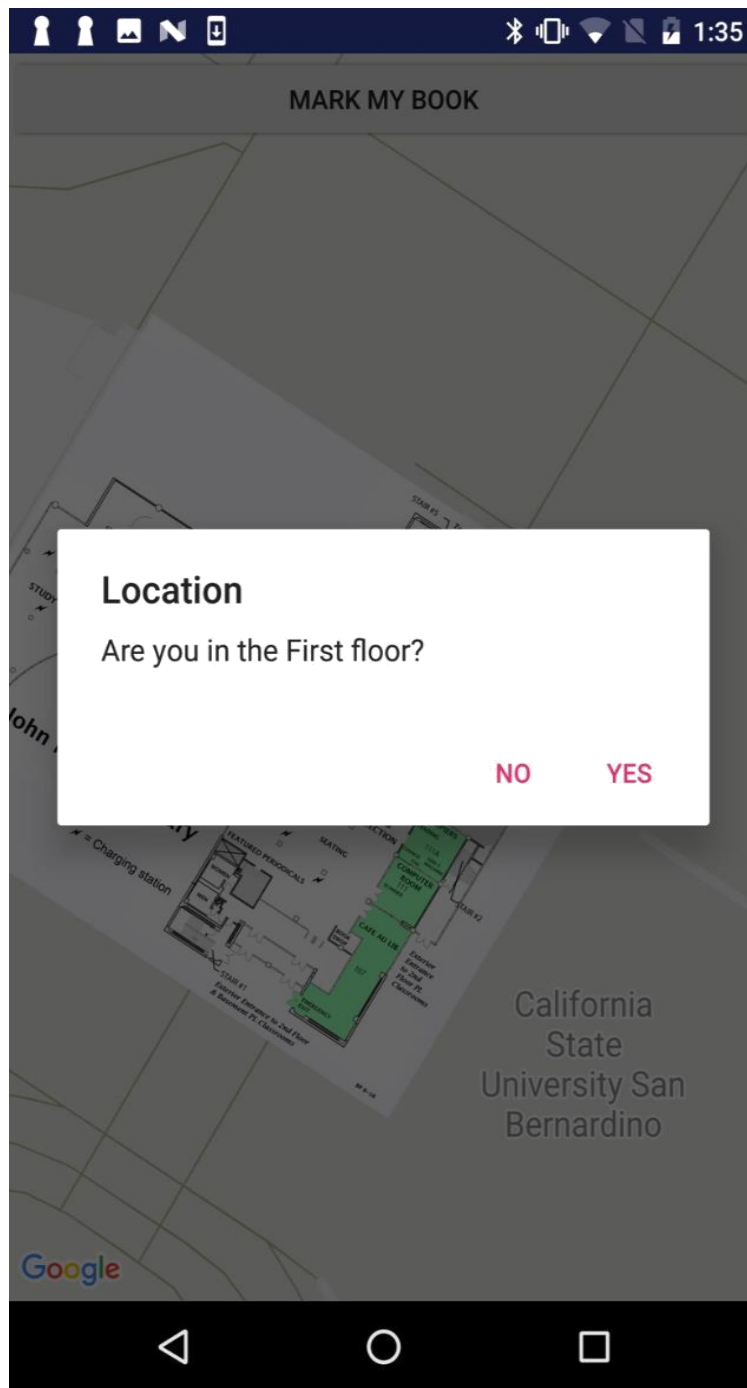


Figure 71. Screen of Second Page- Hitting Mark my book Button

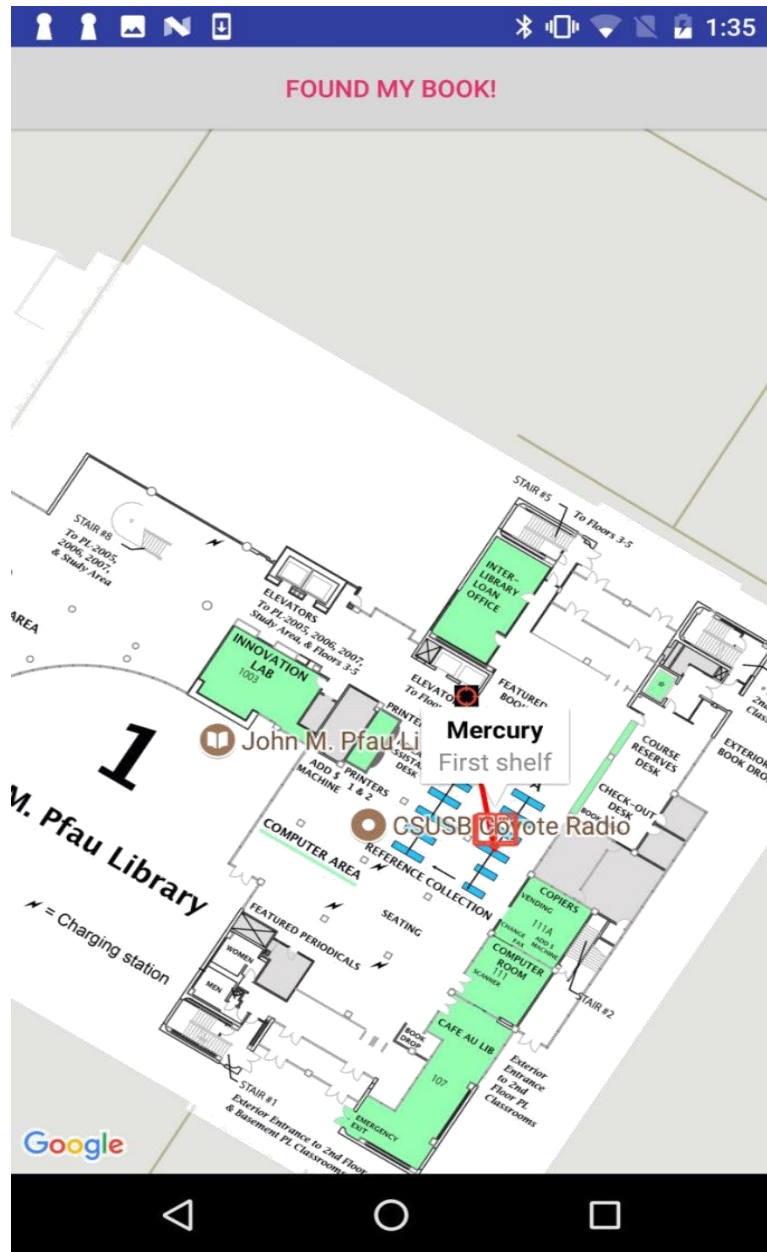


Figure 72. Screen of Third Page- Marker on book's location

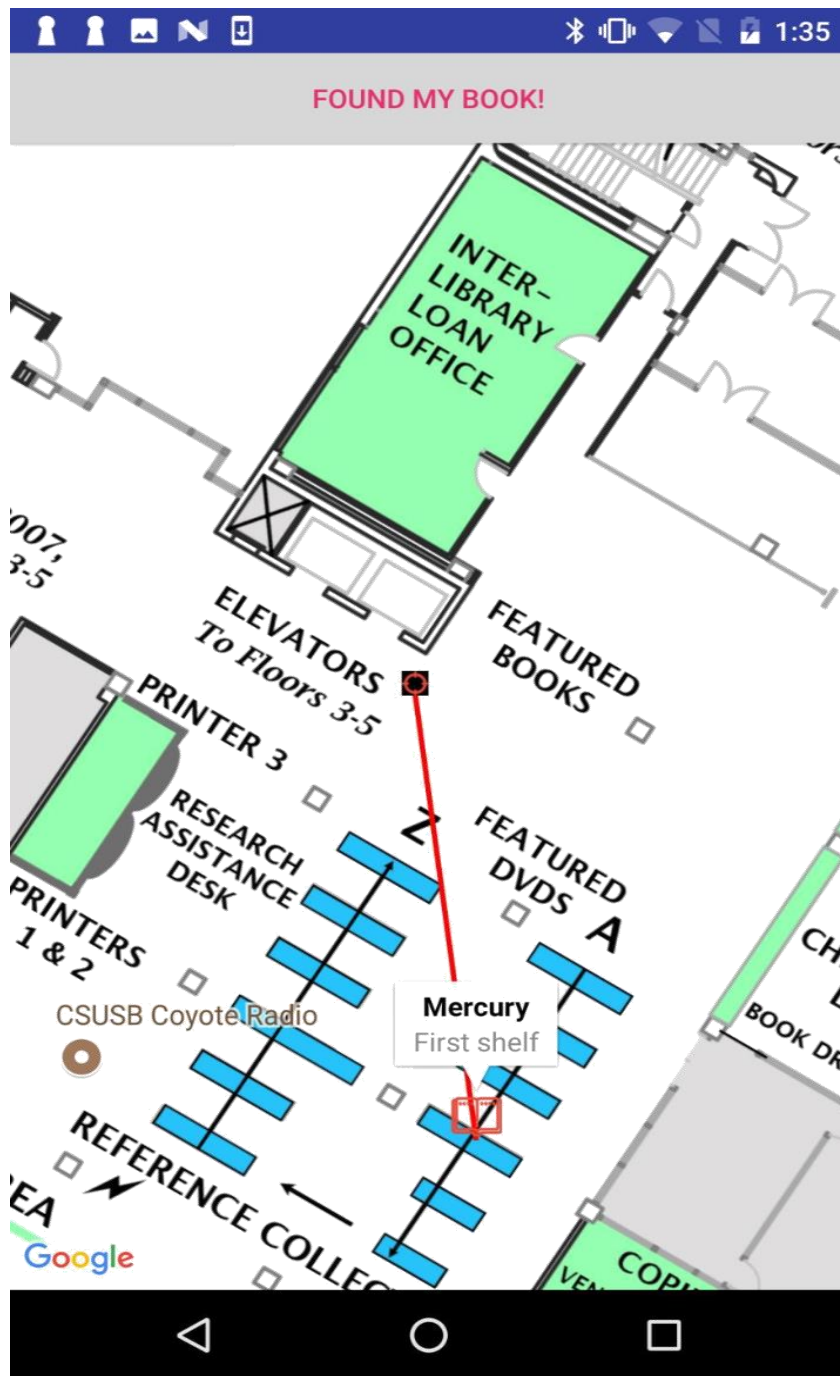


Figure 73. Screen of Third Page- Polyline between two markers

CHAPTER TEN

CONCLUSION

In this modern world, every student has a mobile device. These devices have technologies like Wi-Fi, Bluetooth and GPS. The main aim is to use these technologies to track a student inside a building.

Indoor positioning is one of the most challenging things to achieve these days. Using indoor atlas SDK I was able to map inside of a building and get the position of the student using the Wi-Fi, Bluetooth and GPS. The location of the student is however inaccurate, but perfect mapping would help us achieve accurate results.

Indoor positioning systems (IPS) locate people or objects inside a building using radio signals, geomagnetic fields, inertial sensor data, barometric pressure, camera data or other sensory information collected by a smartphone device or tablet. There are many different types of indoor positioning systems available on the market today. IndoorAtlas provides a unique Platform-as-a-Service (PaaS) solution that runs a disruptive geomagnetic positioning in its full-stack hybrid technology core to accurately pinpoint a location inside a building.

APPENDIX A
ANDROID CODE

1. MainActivity.java

```
package com.example.snehapantam.runtime;

import android.Manifest;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.IntentSender;
import android.content.pm.PackageManager;
import android.graphics.Color;
import android.location.Location;
import android.service.carrier.CarrierMessagingService;
import android.support.annotation.NonNull;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.text.TextUtils;
import android.util.Log;
import android.view.View;
import android.widget.*;

import com.google.android.gms.maps.GoogleMap;

import com.google.android.gms.maps.model.Marker;
import com.indooratlas.android.sdk.IALocation;
import com.indooratlas.android.sdk.IALocationListener;
import com.indooratlas.android.sdk.IALocationManager;
import com.indooratlas.android.sdk.IALocationRequest;

public class MainActivity extends AppCompatActivity implements
    IALocationListener, AdapterView.OnItemClickListener {
    private static final String TAG = "MainActivity";

    private IALocationManager mLocationManager;
```

```

@Override
protected void onDestroy() {
    super.onDestroy();
    mLocationManager.destroy();
}

@Override
protected void onResume() {
    super.onResume();

    mLocationManager.requestLocationUpdates(IALocationRequest.create(), this);
}

@Override
protected void onPause() {
    super.onPause();
    if (mLocationManager != null) {
        mLocationManager.removeLocationUpdates(this);
    }
}

public void onLocationChanged(IALocation location) {
    ensurePermissions();

    Log.d(TAG, "Latitude: " + location.getLatitude());
    Log.d(TAG, "Longitude: " + location.getLongitude());

}

@Override
public void onStatusChanged(String provider, int status, Bundle extras) {
    // N/A
}

```

```

private void ensurePermissions() {

    if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION)
        != PackageManager.PERMISSION_GRANTED)
    {

        // we don't have access to coarse locations, hence we have not access to wifi
either
        // check if this requires explanation to user
        if (ActivityCompat.shouldShowRequestPermissionRationale(this,
            Manifest.permission.ACCESS_FINE_LOCATION) ){

            new AlertDialog.Builder(this)
                .setTitle(R.string.location_permission_request_title)
                .setMessage(R.string.location_permission_request_rationale)
                .setPositiveButton(R.string.permission_button_accept, new
DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        Log.d(TAG, "request permissions");
                        ActivityCompat.requestPermissions(MainActivity.this,
                            new
String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                                REQUEST_CODE_ACCESS_COARSE_LOCATION);
                    }
                })
                .setNegativeButton(R.string.permission_button_deny, new
DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        Toast.makeText(MainActivity.this,
                            R.string.location_permission_denied_message,
                            Toast.LENGTH_LONG).show();
                    }
                })
                .show();

        } else {
            Log.d(TAG, "asking permissions to user");

            // ask user for permission

```



```

        ActivityCompat.requestPermissions(this,
            new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
            REQUEST_CODE_ACCESS_COARSE_LOCATION);

    }

}

@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[]
grantResults) {

    switch (requestCode) {
        case REQUEST_CODE_ACCESS_COARSE_LOCATION:

            if (grantResults.length > 0
                || grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                Toast.makeText(this, "Permission granted thank you",
                    Toast.LENGTH_LONG).show();

            }
            break;
    }
}

@Override
public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {

    final Intent intent;
    switch(i){

        case 0:
            break;

        case 1:
            intent= new Intent(MainActivity.this, CommonActivity.class);

            intent.putExtra("someVariable", "1");
            startActivity(intent);
    }
}

```

```

        break;

    case 2:
        intent= new Intent(MainActivity.this,CommonActivity.class);
        intent.putExtra("someVariable", "2");
        startActivity(intent);
        break;

    case 3:
        intent= new Intent(MainActivity.this,CommonActivity.class);
        intent.putExtra("someVariable", "3");
        startActivity(intent);
        break;

    case 4:
        intent= new Intent(MainActivity.this,CommonActivity.class);
        intent.putExtra("someVariable", "4");
        startActivity(intent);
        break;

    case 5:
        intent= new Intent(MainActivity.this,CommonActivity.class);
        intent.putExtra("someVariable", "5");
        startActivity(intent);
        break;

    }

}

@Override
public void onNothingSelected(AdapterView<?> adapterView) { }
}

```

2. Activity_main.xml

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"

```

```

        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:paddingLeft="@dimen/activity_horizontal_margin"
        android:paddingRight="@dimen/activity_horizontal_margin"
        android:paddingTop="@dimen/activity_vertical_margin"
        android:paddingBottom="@dimen/activity_vertical_margin"
        tools:context=".MainActivity">

        <!-- MapView -->

        <TextView
            android:id="@+id/textView2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true"
            android:layout_centerHorizontal="true"
            android:layout_marginTop="29dp"
            android:text="@string/book_hunt"
            android:textColor="@android:color/black"
            android:textSize="36sp"
            android:textStyle="bold" />

        <Spinner
            android:id="@+id/spinner"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_alignParentStart="true"
            android:layout_below="@+id/textView2"
            android:layout_marginTop="121dp" />

        <Button
            android:id="@+id/help"
            style="@style/Widget.AppCompat.ActionMode"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentBottom="true"
            android:layout_alignParentEnd="true"
            android:layout_marginBottom="32dp"
            android:layout_marginEnd="12dp"
            android:text="@string/help"
            android:textAppearance="@style/TextAppearance.AppCompat.Button" />

    </RelativeLayout>

```

3. CommonActivity.java

```
package com.example.snehapantam.runtime;

import android.content.DialogInterface;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.drawable.Drawable;
import android.os.Bundle;
import android.os.Looper;
import android.support.v4.app.FragmentActivity;
import android.support.v4.widget.SwipeRefreshLayout;
import android.support.v7.app.AlertDialog;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.BitmapDescriptor;
import com.google.android.gms.maps.model.BitmapDescriptorFactory;
import com.google.android.gms.maps.model.GroundOverlay;
import com.google.android.gms.maps.model.GroundOverlayOptions;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;

import com.indooratlas.android.sdk.IALocation;
import com.indooratlas.android.sdk.IALocationListener;
import com.indooratlas.android.sdk.IALocationManager;
import com.indooratlas.android.sdk.IALocationRequest;
import com.indooratlas.android.sdk.IARegion;

import com.indooratlas.android.sdk.resources.IAFloorPlan;
import com.indooratlas.android.sdk.resources.IALatLng;
import com.indooratlas.android.sdk.resources.IALocationListenerSupport;
import com.indooratlas.android.sdk.resources.IAResourceManager;
import com.indooratlas.android.sdk.resources.IAResult;
import com.indooratlas.android.sdk.resources.IAResultCallback;
import com.indooratlas.android.sdk.resources.IATask;
import com.squareup.picasso.Picasso;
import com.squareup.picasso.RequestCreator;
import com.squareup.picasso.Target;
```

```

public class CommonActivity extends FragmentActivity implements
OnMapReadyCallback{

    private static final String TAG = "CommonActivity";

    /* used to decide when bitmap should be downscaled */
    private static final int MAX_DIMENSION = 2048;

    private GoogleMap mMap; // Might be null if Google Play services APK is not
available.
    private Marker mMarker;

    private IRegion mOverlayFloorPlan = null;
    private GroundOverlay mGroundOverlay = null;
    private ILocationManager mILocationManager;
    private IResourceManager mResourceManager;
    private ITask<IAFloorPlan> mFetchFloorPlanTask;
    private Target mLoadTarget;
    private boolean mCameraPositionNeedsUpdating = true; // update on first location

    /**
     * Listener that handles location change events.
     */
    public ILocationListener mListener = new ILocationListenerSupport() {

        /**
         * Location changed, move marker and camera position.
         */
        @Override
        public void onLocationChanged(final ILocation location) {

            Log.d(TAG, "new location received with coordinates: " + location.getLatitude()
                + ", " + location.getLongitude());

            if (mMap == null) {
                // location received before map is initialized, ignoring update here
                return;
            }
        }
    };

```

```

    }

    LatLng latLng = new LatLng(location.getLatitude(),
location.getLongitude());/*Location of user from IA SDK*/

    if (mMarker == null) {
        // first location, add marker
        mMarker = mMap.addMarker(new MarkerOptions().position(latLng)
            .icon(BitmapDescriptorFactory.fromResource(R.mipmap.icon2)));
    } else {
        // move existing markers position to received location
        mMarker.setPosition(latLng);
    }

    // our camera position needs updating if location has significantly changed
    if (mCameraPositionNeedsUpdating) {
        mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(latLng, 19));
        mCameraPositionNeedsUpdating = false;
    }

}

};

/**
 * Listener that changes overlay if needed
 */

```

```

private IRegion.Listener mRegionListener = new IRegion.Listener() {

    @Override
    public void onEnterRegion(IRegion region) {
        if (region.getType() == IRegion.TYPE_FLOOR_PLAN) {
            final String newId = region.getId();
            // Are we entering a new floor plan or coming back the floor plan we just left?
            if (mGroundOverlay == null || !region.equals(mOverlayFloorPlan)) {
                mCameraPositionNeedsUpdating = true; // entering new fp, need to move
camera
                if (mGroundOverlay != null) {
                    mGroundOverlay.remove();
                    mGroundOverlay = null;
                }
                mOverlayFloorPlan = region; // overlay will be this (unless error in loading)
                fetchFloorPlan(newId);

            } else {
                mGroundOverlay.setTransparency(0.0f);
            }
        }
        Log.d(TAG, "Enter " + (region.getType() == IRegion.TYPE_VENUE
            ? "VENUE "
            : "FLOOR_PLAN ") + region.getId());

    }

    @Override
    public void onExitRegion(IRegion region) {
        if (mGroundOverlay != null) {
            // Indicate we left this floor plan but leave it there for reference
            // If we enter another floor plan, this one will be removed and another one
loaded
            mGroundOverlay.setTransparency(0.5f);
        }
        Log.d(TAG, "Enter " + (region.getType() == IRegion.TYPE_VENUE
            ? "VENUE "
            : "FLOOR_PLAN ") + region.getId());
    }

};

```

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_common);
    SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager()
    .findFragmentById(R.id.map);
    mapFragment.getMapAsync(this);

    final SwipeRefreshLayout swipeRefreshLayout = (SwipeRefreshLayout)
findViewById(R.id.swipe);

    swipeRefreshLayout.setOnRefreshListener(new
SwipeRefreshLayout.OnRefreshListener() {
        @Override
        public void onRefresh() {
            finish();
            startActivity(getIntent());
        }
    });

    Bundle extras = getIntent().getExtras();
    if (extras != null) {
        final String someVariable = extras.getString("someVariable");

        Log.d(TAG, "value" + someVariable);

        switch (someVariable) {

            case "0":
                break;

            case "1":
                Toast.makeText(CommonActivity.this, "Your book Mercury is in the First
floor", Toast.LENGTH_LONG).show();
                Toast.makeText(CommonActivity.this, "Click Mark my Book button after
you reach First Floor", Toast.LENGTH_LONG).show();

                break;

```



```

        case "2":
            Toast.makeText(CommonActivity.this, "Your book Venus is in the Fourth
            floor", Toast.LENGTH_LONG).show();
            Toast.makeText(CommonActivity.this, "Click Mark my Book button after
            you reach Fourth Floor", Toast.LENGTH_LONG).show();

            break;

        case "3":
            Toast.makeText(CommonActivity.this, "Your book EarthActivity is in the
            Third floor", Toast.LENGTH_LONG).show();
            Toast.makeText(CommonActivity.this, "Click Mark my Book button after
            you reach Third Floor", Toast.LENGTH_LONG).show();

            break;

        case "4":
            Toast.makeText(CommonActivity.this, "Your book Mars is in the Third
            floor", Toast.LENGTH_LONG).show();
            Toast.makeText(CommonActivity.this, "Click Mark my Book button after
            you reach Third Floor", Toast.LENGTH_LONG).show();

            break;

        case "5":
            Toast.makeText(CommonActivity.this, "Your book Jupiter is in the Fourth
            floor", Toast.LENGTH_LONG).show();
            Toast.makeText(CommonActivity.this, "Click Mark my Book button after
            you reach Fourth Floor", Toast.LENGTH_LONG).show();

            break;

    }

    // prevent the screen going to sleep while app is on foreground
    findViewById(android.R.id.content).setKeepScreenOn(false);

    // instantiate IALocationManager and IAResourceManager
    mIALocationManager = IALocationManager.create(this);
    mResourceManager = IAResourceManager.create(this);

```

```

        final Button button = (Button) findViewById(R.id.buttoncommon);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                AlertDialog.Builder alerdialog = new
                AlertDialog.Builder(CommonActivity.this);
                alerdialog.setTitle("Location");

                switch (someVariable) {

                    case "0":
                        break;

                    case "1":
                        alerdialog.setMessage("Are you in the First floor?");
                        break;

                    case "2":
                        alerdialog.setMessage("Are you in the Fourth floor?");
                        break;

                    case "3":
                        alerdialog.setMessage("Are you in the Third floor?");
                        break;

                    case "4":
                        alerdialog.setMessage("Are you in the Third floor?");
                        break;
                    case "5":
                        alerdialog.setMessage("Are you in the Fourth floor?");
                        break;

                }

                alerdialog.setPositiveButton("Yes", new DialogInterface.OnClickListener()
{
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                final Intent intent;

                switch (someVariable) {

                    case "0":
                        break;

```

```

        case "1":
            intent = new Intent(CommonActivity.this, MercuryActivity.class);

            startActivity(intent);
            break;

        case "2":
            intent = new Intent(CommonActivity.this, VenusActivity.class);
            startActivity(intent);
            break;

        case "3":
            intent = new Intent(CommonActivity.this, EarthActivity.class);
            startActivity(intent);
            break;

        case "4":
            intent = new Intent(CommonActivity.this, MarsActivity.class);
            startActivity(intent);
            break;
        case "5":
            intent = new Intent(CommonActivity.this, JupiterActivity.class);
            startActivity(intent);
            break;
    }

    }
});
AlertDialog.setNegativeButton("No", new DialogInterface.OnClickListener()
{
    @Override
    public void onClick(DialogInterface dialogInterface, int i) {
        switch (someVariable) {

            case "0":
                break;

            case "1":
                Toast.makeText(CommonActivity.this, "Please go to the First
floor", Toast.LENGTH_LONG).show();

```

```

        break;

        case "2":
            Toast.makeText(CommonActivity.this, "Please go to the Fourth
floor", Toast.LENGTH_LONG).show();

            break;

        case "3":
            Toast.makeText(CommonActivity.this, "Please go to the Third
floor", Toast.LENGTH_LONG).show();

            break;

        case "4":
            Toast.makeText(CommonActivity.this, "Please go to the Third
floor", Toast.LENGTH_LONG).show();

            break;
        case "5":
            Toast.makeText(CommonActivity.this, "Please go to the Fourth
floor", Toast.LENGTH_LONG).show();

            break;

    }

}

});
AlertDialog alert = alerdialog.create();
alert.show();

}
});

```

```

    }
}

@Override
protected void onDestroy() {
    super.onDestroy();
    // remember to clean up after ourselves
    mIALocationManager.destroy();
}

@Override
protected void onResume() {
    super.onResume();
    if (mMap == null) {
        // Try to obtain the map from the SupportMapFragment.
        ((SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map))
        .getMapAsync(this);
    }

    // start receiving location updates & monitor region changes
    mIALocationManager.requestLocationUpdates(IALocationRequest.create(),
mListener);
    mIALocationManager.registerRegionListener(mRegionListener);
}

@Override
protected void onPause() {
    super.onPause();
    // unregister location & region changes
    mIALocationManager.removeLocationUpdates(mListener);
    mIALocationManager.unregisterRegionListener(mRegionListener);
}

/**
 * Sets bitmap of floor plan as ground overlay on Google Maps
 */
private void setupGroundOverlay(IAFloorPlan floorPlan, Bitmap bitmap) {
    if (mGroundOverlay != null) {
        mGroundOverlay.remove();
    }
}

```

```

        if (mMap != null) {
            BitmapDescriptor bitmapDescriptor =
                BitmapDescriptorFactory.fromBitmap(bitmap);
            IALatLng iaLatLng = floorPlan.getCenter();
            LatLng center = new LatLng(iaLatLng.latitude, iaLatLng.longitude);
            GroundOverlayOptions fpOverlay = new GroundOverlayOptions()
                .image(bitmapDescriptor)
                .position(center, floorPlan.getWidthMeters(), floorPlan.getHeightMeters())
                .bearing(floorPlan.getBearing());

            mGroundOverlay = mMap.addGroundOverlay(fpOverlay);
        }

        /**
         * Download floor plan using Picasso library.
         */
        private void fetchFloorPlanBitmap(final IAFloorPlan floorPlan) {

            final String url = floorPlan.getUrl();

            if (mLoadTarget == null) {
                mLoadTarget = new Target() {

                    @Override
                    public void onBitmapLoaded(Bitmap bitmap, Picasso.LoadedFrom from) {
                        Log.d(TAG, "onBitmap loaded with dimensions: " + bitmap.getWidth() + "x"
                            + bitmap.getHeight());
                        setupGroundOverlay(floorPlan, bitmap);
                    }

                    @Override
                    public void onPrepareLoad(Drawable placeHolderDrawable) {
                        // N/A
                    }

                    @Override
                    public void onBitmapFailed(Drawable placeHolderDrawable) {
                        Log.d(TAG, "Failed to load bitmap");
                        mOverlayFloorPlan = null;
                    }
                };
            }

            RequestCreator request = Picasso.with(this).load(url);

```

```

final int bitmapWidth = floorPlan.getBitmapWidth();
final int bitmapHeight = floorPlan.getBitmapHeight();

if (bitmapHeight > MAX_DIMENSION) {
    request.resize(0, MAX_DIMENSION);
} else if (bitmapWidth > MAX_DIMENSION) {
    request.resize(MAX_DIMENSION, 0);
}

request.into(mLoadTarget);
}

/**
 * Fetches floor plan data from IndoorAtlas server.
 */
private void fetchFloorPlan(String id) {

    // if there is already running task, cancel it
    cancelPendingNetworkCalls();

    final IATask<IAFloorPlan> task = mResourceManager.fetchFloorPlanWithId(id);

    task.setCallback(new IAResultCallback<IAFloorPlan>() {

        @Override
        public void onResult(IAResult<IAFloorPlan> result) {

            if (result.isSuccess() && result.getResult() != null) {
                // retrieve bitmap for this floor plan metadata
                fetchFloorPlanBitmap(result.getResult());
            } else {
                // ignore errors if this task was already canceled
                if (!task.isCancelled()) {
                    // do something with error
                    Log.d(TAG, "Loading floor plan failed: " + result.getError());
                    mOverlayFloorPlan = null;
                }
            }
        }
    }, Looper.getMainLooper()); // deliver callbacks using main looper

    // keep reference to task so that it can be canceled if needed
    mFetchFloorPlanTask = task;

```

```

    }

    /**
     * Helper method to cancel current task if any.
     */
    private void cancelPendingNetworkCalls() {
        if (mFetchFloorPlanTask != null && !mFetchFloorPlanTask.isCancelled()) {
            mFetchFloorPlanTask.cancel();
        }
    }

    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;
    }
}

```


4. activity_common.xml

```
<android.support.v4.widget.SwipeRefreshLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/swipe"
android:layout_width="match_parent"
android:layout_height="match_parent">

<RelativeLayout android:layout_height="match_parent"
android:layout_width="match_parent"
xmlns:android="http://schemas.android.com/apk/res/android">

    <fragment xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:map="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:id="@+id/map"
        android:name="com.google.android.gms.maps.SupportMapFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context="com.example.snehapantam.runtime.MapsActivity"
        android:layout_alignParentBottom="true"
        android:layout_alignParentStart="true" />

    <Button
        android:id="@+id/buttoncommon"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/mark_my_book"
        android:layout_alignParentTop="true"
        android:layout_alignParentStart="true" />

</RelativeLayout>
</android.support.v4.widget.SwipeRefreshLayout>
```

5. MercuryActivity.java

```
package com.example.snehapantam.runtime;

import android.content.DialogInterface;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.drawable.Drawable;
import android.os.Bundle;
import android.os.Looper;
import android.os.Process;
import android.support.v4.app.FragmentActivity;
import android.support.v4.widget.SwipeRefreshLayout;
import android.support.v7.app.AlertDialog;
import android.util.Log;
import android.view.View;
import android.widget.Button;

import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.BitmapDescriptor;
import com.google.android.gms.maps.model.BitmapDescriptorFactory;
import com.google.android.gms.maps.model.GroundOverlay;
import com.google.android.gms.maps.model.GroundOverlayOptions;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;
import com.google.android.gms.maps.model.Polyline;
import com.google.android.gms.maps.model.PolylineOptions;

import com.indooratlas.android.sdk.IALocation;
import com.indooratlas.android.sdk.IALocationListener;
import com.indooratlas.android.sdk.IALocationManager;
import com.indooratlas.android.sdk.IALocationRequest;
import com.indooratlas.android.sdk.IARegion;

import com.indooratlas.android.sdk.resources.IAFloorPlan;
import com.indooratlas.android.sdk.resources.IALatLng;
import com.indooratlas.android.sdk.resources.IALocationListenerSupport;
import com.indooratlas.android.sdk.resources.IAResourceManager;
import com.indooratlas.android.sdk.resources.IAResult;
import com.indooratlas.android.sdk.resources.IAResultCallback;
import com.indooratlas.android.sdk.resources.IATask;
import com.squareup.picasso.Picasso;
import com.squareup.picasso.RequestCreator;
```

```

import com.squareup.picasso.Target;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import static android.graphics.Color.RED;

public class MercuryActivity extends FragmentActivity implements
    OnMapReadyCallback {

    private static final String TAG = "MercuryActivity";

    /* used to decide when bitmap should be downscaled */
    private static final int MAX_DIMENSION = 2048;
    final Map<String, Polyline> polylines = new HashMap<>();
    List<LatLng> points= new ArrayList<LatLng>();
    LatLng mercury_book= new LatLng(34.18249717, -117.32380673);
    private GoogleMap mMap; // Might be null if Google Play services APK is not
    available.
    private Marker mMarker;
    private IRegion mOverlayFloorPlan = null;
    private GroundOverlay mGroundOverlay = null;
    private ILocationManager mILocationManager;
    private IResourceManager mResourceManager;
    private ITask<IAFloorPlan> mFetchFloorPlanTask;
    private Target mLoadTarget;
    private boolean mCameraPositionNeedsUpdating = true;
    /**
     * Listener that handles location change events.
     */
    public ILocationListener mListener = new ILocationListenerSupport() {

        /**
         * Location changed, move marker and camera position.
         */
        @Override
        public void onLocationChanged(final ILocation location) {

            if(points.size() == 1)
            {
                points.clear();
            }

```

```

polylines.get("polylinesneha").remove();
}

Log.d(TAG, "new location received with coordinates: " + location.getLatitude()
    + "," + location.getLongitude());

if (mMap == null) {
    // location received before map is initialized, ignoring update here
    return;
}

LatLng latLng = new LatLng(location.getLatitude(), location.getLongitude());

if (mMarker == null) {
    // first location, add marker
    mMarker = mMap.addMarker(new MarkerOptions().position(latLng)
        .icon(BitmapDescriptorFactory.fromResource(R.mipmap.icon2)));
} else {
    // move existing markers position to received location
    mMarker.setPosition(latLng);
}

// our camera position needs updating if location has significantly changed
if (mCameraPositionNeedsUpdating) {
    mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(latLng, 21));
    mCameraPositionNeedsUpdating = false;
}

points.add(latLng);

Marker mMarker1 = mMap.addMarker(new
MarkerOptions().position(mercury_book).title("Mercury").snippet("First shelf")
    .icon(BitmapDescriptorFactory.fromResource(R.mipmap.book_icon)));

mMarker1.showInfoWindow();
polylinemethod();

```

```

    }
};
/**
 * Listener that changes overlay if needed
 */
private IRegion.Listener mRegionListener = new IRegion.Listener() {

    @Override
    public void onEnterRegion(IRegion region) {
        if (region.getType() == IRegion.TYPE_FLOOR_PLAN) {
            final String newId = region.getId();
            // Are we entering a new floor plan or coming back the floor plan we just left?
            if (mGroundOverlay == null || !region.equals(mOverlayFloorPlan)) {
                mCameraPositionNeedsUpdating = true; // entering new fp, need to move
camera
                if (mGroundOverlay != null) {
                    mGroundOverlay.remove();
                    mGroundOverlay = null;
                }
                mOverlayFloorPlan = region; // overlay will be this (unless error in loading)
                fetchFloorPlan(newId);

            } else {
                mGroundOverlay.setTransparency(0.0f);
            }
        }
        Log.d(TAG, "Enter " + (region.getType() == IRegion.TYPE_VENUE
            ? "VENUE "
            : "FLOOR_PLAN ") + region.getId());

    }

    @Override
    public void onExitRegion(IRegion region) {
        if (mGroundOverlay != null) {
            // Indicate we left this floor plan but leave it there for reference
            // If we enter another floor plan, this one will be removed and another one
loaded
            mGroundOverlay.setTransparency(0.5f);
        }
    }
}

```

```

        Log.d(TAG, "Enter " + (region.getType() == IRegion.TYPE_VENUE
            ? "VENUE "
            : "FLOOR_PLAN ") + region.getId());
    }

};

private void polylinemethod() {

    PolylineOptions polylineOptions= new
PolylineOptions().width(10).color(RED).geodesic(true);

polylines.put("polylinesneha", mMap.addPolyline(polylineOptions.add(points.get(0),merc
ury_book)));

}

@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_mercury);
    SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager()
        .findFragmentById(R.id.map);
    mapFragment.getMapAsync(this);
    // points = new ArrayList<LatLng>(); // to do to delete

    // prevent the screen going to sleep while app is on foreground
    findViewById(android.R.id.content).setKeepScreenOn(true);

    // instantiate IALocationManager and IAResourceManager
    mIALocationManager = IALocationManager.create(this);
    mResourceManager = IAResourceManager.create(this);

    final SwipeRefreshLayout swipeRefreshLayout = (SwipeRefreshLayout)
findViewById(R.id.swipe);

    swipeRefreshLayout.setOnRefreshListener(new
SwipeRefreshLayout.OnRefreshListener() {

```

```

@Override
public void onRefresh() {
    finish();
    startActivity(getIntent());
}
});

final Button finish = (Button) findViewById(R.id.finish);

finish.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View view) {

        AlertDialog.Builder alerdialog = new
AlertDialog.Builder(MercuryActivity.this);
        alerdialog.setTitle("Thank you for using our App!");

        alerdialog.setPositiveButton("OK", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                final Intent intent = new Intent(MercuryActivity.this, MainActivity.class);
                startActivity(intent);
            }
        });
        AlertDialog alert = alerdialog.create();
        alert.show();
    }
});

@Override
protected void onDestroy() {
    super.onDestroy();
    // remember to clean up
    //
    // after ourselves

    mIALocationManager.destroy();
}

@Override
protected void onResume() {

```



```

        super.onResume();
        if (mMap == null) {
            // Try to obtain the map from the SupportMapFragment.
            ((SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map))
                .getMapAsync(this);

        }

        // start receiving location updates & monitor region changes
        mILocationManager.requestLocationUpdates(ILocationRequest.create(),
mListener);
        mILocationManager.registerRegionListener(mRegionListener);
    }

    @Override
    protected void onPause() {
        super.onPause();
        // unregister location & region changes
        mILocationManager.removeLocationUpdates(mListener);
        mILocationManager.unregisterRegionListener(mRegionListener);
    }

    /**
     * Sets bitmap of floor plan as ground overlay on Google Maps
     */
    private void setupGroundOverlay(IAFloorPlan floorPlan, Bitmap bitmap) {

        if (mGroundOverlay != null) {
            mGroundOverlay.remove();
        }

        if (mMap != null) {
            BitmapDescriptor bitmapDescriptor =
BitmapDescriptorFactory.fromBitmap(bitmap);
            IALatLng iaLatLng = floorPlan.getCenter();
            LatLng center = new LatLng(iaLatLng.latitude, iaLatLng.longitude);
            GroundOverlayOptions fpOverlay = new GroundOverlayOptions()
                .image(bitmapDescriptor)
                .position(center, floorPlan.getWidthMeters(), floorPlan.getHeightMeters())
                .bearing(floorPlan.getBearing());

            mGroundOverlay = mMap.addGroundOverlay(fpOverlay);
        }
    }

```



```

}

/**
 * Download floor plan using Picasso library.
 */
private void fetchFloorPlanBitmap(final IAFloorPlan floorPlan) {

    final String url = floorPlan.getUrl();

    if (mLoadTarget == null) {
        mLoadTarget = new Target() {

            @Override
            public void onBitmapLoaded(Bitmap bitmap, Picasso.LoadedFrom from) {
                Log.d(TAG, "onBitmap loaded with dimensions: " + bitmap.getWidth() + "x"
                    + bitmap.getHeight());
                setupGroundOverlay(floorPlan, bitmap);
            }

            @Override
            public void onPrepareLoad(Drawable placeHolderDrawable) {
                // N/A
            }

            @Override
            public void onBitmapFailed(Drawable placeHolderDrawable) {
                Log.d(TAG, "Failed to load bitmap");
                mOverlayFloorPlan = null;
            }
        };
    }

    RequestCreator request = Picasso.with(this).load(url);

    final int bitmapWidth = floorPlan.getBitmapWidth();
    final int bitmapHeight = floorPlan.getBitmapHeight();

    if (bitmapHeight > MAX_DIMENSION) {
        request.resize(0, MAX_DIMENSION);
    } else if (bitmapWidth > MAX_DIMENSION) {
        request.resize(MAX_DIMENSION, 0);
    }

    request.into(mLoadTarget);
}

```

```

/**
 * Fetches floor plan data from IndoorAtlas server.
 */
private void fetchFloorPlan(String id) {

    // if there is already running task, cancel it
    cancelPendingNetworkCalls();

    final IATask<IAFloorPlan> task = mResourceManager.fetchFloorPlanWithId(id);

    task.setCallback(new IAResultCallback<IAFloorPlan>() {

        @Override
        public void onResult(IAResult<IAFloorPlan> result) {

            if (result.isSuccess() && result.getResult() != null) {
                // retrieve bitmap for this floor plan metadata
                fetchFloorPlanBitmap(result.getResult());
            } else {
                // ignore errors if this task was already canceled
                if (!task.isCancelled()) {
                    // do something with error
                    Log.d(TAG, "Loading floor plan failed: " + result.getError());
                    mOverlayFloorPlan = null;
                }
            }
        }
    }, Looper.getMainLooper()); // deliver callbacks using main looper

    // keep reference to task so that it can be canceled if needed
    mFetchFloorPlanTask = task;

}

/**
 * Helper method to cancel current task if any.
 */
private void cancelPendingNetworkCalls() {
    if (mFetchFloorPlanTask != null && !mFetchFloorPlanTask.isCancelled()) {
        mFetchFloorPlanTask.cancel();
    }
}

@Override

```

```
public void onMapReady(GoogleMap googleMap) {  
    mMap = googleMap;  
  
}  
  
}
```

6. activity_mercury.xml

```
<android.support.v4.widget.SwipeRefreshLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/swipe"
  android:layout_width="match_parent"
  android:layout_height="match_parent">

  <RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_alignParentStart="true"
    android:layout_alignParentTop="true">

    <fragment xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:map="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/tools"
      android:id="@+id/map"
      android:name="com.google.android.gms.maps.SupportMapFragment"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:layout_alignParentBottom="true"
      android:layout_alignParentStart="true"
      tools:context="com.example.snehapantam.runtime.MapsActivity" />

    <Button
      android:id="@+id/finish"
      style="@style/Widget.AppCompat.Button"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_alignParentStart="true"
      android:layout_alignParentTop="true"
      android:background="@android:color/background_light"
      android:backgroundTint="?android:attr/colorButtonNormal"
      android:text="@string/found_my_book"
      android:textColor="@color/colorAccent" />

  </RelativeLayout>

</android.support.v4.widget.SwipeRefreshLayout>
```

7. AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.snehapantam.runtime">

    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"
/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
/>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
/>

    <uses-feature
        android:glEsVersion="0x00020000"
        android:required="true" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <meta-data
            android:name="com.indooratlas.android.sdk.API_KEY"
            android:value="823e8497-a390-4513-a0e5-bef2f7f5d64a" />
        <meta-data
            android:name="com.indooratlas.android.sdk.API_SECRET"
            android:value="oL9nC1b1+fStmiJUatChMzbNqAKcgjNnPI5xyZok18QY6YgY3YxdxMgcPPuwea
krJDyixNiMr8xblrsPb4DumWs5EzCW0ZtmedkWYBbv+FjhHi0nnd1VIMzB7YRTMA==" />
        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="AIzaSyAXAv1_6xcPLyq4CyExbQEfvplDNGP5i7s" />

        <activity android:name=".MercuryActivity"
            android:parentActivityName=".CommonActivity"/>
    </application>
</manifest>
```

```

        <activity android:name=".VenusActivity"
            android:parentActivityName=".CommonActivity" />
        <activity
            android:name=".MarsActivity"
            android:label="@string/title_activity_mars2"
            android:parentActivityName=".CommonActivity"
            android:theme="@style/AppTheme.NoActionBar">
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.example.snehapantam.runtime.CommonActivity" />
            </activity>
        <activity
            android:name=".EarthActivity"
            android:label="@string/title_activity_earth"
            android:parentActivityName=".CommonActivity"
            android:theme="@style/AppTheme.NoActionBar" />

        <!--The API key for Google Maps-based APIs is defined as a string
        resource. (See the file "res/values/google maps api.xml").
        Note that the API key is linked to the encryption key used
        to sign the APK.
        You need a different API key for each encryption key,
        including the release key that is used to
        sign the APK for publishing.
        You can define the keys for the debug and release targets in
        src/debug/ and src/release/.
        -->
        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="AIzaSyAXAv1_6xcPLYq4CyExbQEfvplDNGP5i7s" />

        <!--
        The API key for Google Maps-based APIs is defined as a string
        resource.
        (See the file "res/values/google maps api.xml").
        Note that the API key is linked to the encryption key used to
        sign the APK.
        You need a different API key for each encryption key, including
        the release key that is used to
        sign the APK for publishing.
        You can define the keys for the debug and release targets in
        src/debug/ and src/release/.-->

        <activity
            android:name=".CommonActivity"
            android:label="@string/title_activity_common">
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.example.snehapantam.runtime.MainActivity"
            />
        </activity>
        <activity android:name=".HelpActivity"
            android:parentActivityName=".MainActivity"></activity>
    </application>

</manifest>

```


8. Strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Book-Hunt!</string>
    <string name="action_clear">Clear</string>
    <string name="example_simple_title">01. Simple</string>
    <string name="example_simple_description">Demonstrates basic
functionality, shows events as text entries.</string>
    <string name="example_imageview_title">02. ImageView</string>
    <string name="example_imageview_description">Demonstrates showing
locations on a floor plan displayed with ImageView.</string>
    <string name="example_googlemaps_basic_title">03. Google Maps -
Basic</string>
    <string name="example_googlemaps_basic_description">Demonstrates showing
location on GoogleMap</string>
    <string name="example_googlemaps_overlay_title">04. Google Maps -
Overlay</string>
    <string name="example_googlemaps_overlay_description">Demonstrates
showing location over a floor plan which is loaded as map overlay. Shows both
IndoorAtlas location and platform location (GPS).</string>
    <string name="example_osm_overlay_title">08. Overlay with Open Street
Map</string>
    <string name="example_osm_overlay_description">Maps overlay with Open
Street Map</string>
    <string name="location_permission_request_title">Permission
required</string>
    <string name="location_permission_request_rationale">Access to coarse
location is required for faster first fix.</string>
    <string name="location_permission_denied_message">No access to locations.
Time to first fix may be severely delayed!</string>
    <string name="storage_permission_denied_message">No access to external
storage. Permission needed for displaying floor plans</string>
    <string name="button_ok">OK</string>
    <string name="button_cancel">Cancel</string>
    <string name="button_close">Close</string>
    <string name="permission_button_accept">Sure</string>
    <string name="permission_button_deny">Deny</string>
    <string name="configuration_incomplete_title">Configuration
incomplete</string>
    <string name="configuration_incomplete_message">API credentials needed,
please see gradle.properties in project root level.</string>
    <string name="dialog_set_location_title">Set location</string>
    <string name="error_could_not_set_location">Error setting location:
%s</string>
    <string name="action_share">Share...</string>
    <string name="share_dialog_title">Your shared name?</string>
    <string name="current_channel">Current channel: %s</string>
    <string name="error_loading_floor_plan">Error while loading floor
plan</string>
    <string name="title_my_name">Me: %s</string>
    <string name="menu_set_channel">Set channel</string>
    <string name="channel_dialog_title">Channel name</string>
    <string name="error_setting_channel">Setting channel failed: %s</string>

```

```

    <string name="example_sharelocation_description">Demonstrates sharing
ones location via 3rd party cloud service.</string>
    <string name="example_sharelocation_title">05. Location sharing</string>
    <string name="menu_change_color">Change color</string>
    <string name="example_credentials_title">06. Set credentials</string>
    <string name="example_credentials_description">Demonstrates setting
credentials from code</string>
    <string name="dialog_request_options_title">Location request
options</string>
    <string name="fastest_interval_label">Fastest interval
(millisecons)</string>
    <string name="shortest_displacement_label">Shortest displacement
(meters)</string>
    <string name="default_label">use default</string>
    <string name="example_regions_description">Demonstrates automatic region
transitions and vertical position information</string>
    <string name="example_regions_title">07. Regions</string>
    <string name="region_information">Region information</string>
    <string name="venue_outside">Outside mapped area</string>
    <string name="venue_inside">In venue</string>
    <string name="floor_plan_outside">No floor plan</string>
    <string name="floor_level">Floor level</string>
    <string name="floor_certainty_percentage">Certainty: %1f %</string>
    <string name="indooratlas_floor_plan_id"></string>

    <!-- Background example -->
    <string name="example_background_description">Demonstrates running
IndoorAtlas location service in the background</string>
    <string name="example_background_title">08. Background mode</string>
    <string name="button_start_background">Request location updates</string>
    <string name="button_stop_background">Remove location updates</string>
    <string name="button_share_background">Share events</string>
    <string name="text_background_received">Last received location:
%</string>
    <string name="button_reset_background">Clear stored events</string>
    <string name="text_background_example">Request location updates with a
PendingIntent. Updates are received by an IntentService that stores them to a
file. Note that you can leave the application and still receive updates.
</string>

    <!-- Orientation example -->
    <string name="example_orientation_description">Demonstrates displaying
the 3D orientation of the device. Panorama image credit: ESO (Very Large
Telescope ready for action)</string>
    <string name="example_orientation_title">09. Orientation</string>
    <string name="text_bearing_initial">Bearing: no updates</string>
    <string name="text_bearing">Bearing: %1$5.1f\u00b0</string>
    <string name="text_heading_initial">Heading: no updates</string>
    <string name="text_heading">Heading: %1$5.1f\u00b0</string>
    <string name="text_orientation_initial">Orient.: no updates</string>
    <string name="text_orientation">Orient.:
[%1$5.2f,%2$5.2f,%3$5.2f,%4$5.2f]</string>
    <string name="title_activity_maps">Map</string>

    <string-array name="planets_array">
        <item>Please select your Book!</item>
        <item>Mercury</item>

```



```

        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
        <item>Jupiter</item>

    </string-array>
    <string name="title_activity_mars2">MarsActivity2</string>
    <string name="title_activity_earth">Earth</string>
    <string name="title_activity_imagview">Imagview</string>
    <string name="title_activity_newone">Map</string>
    <string name="title_activity_common">Map</string>
    <string name="book_hunt">Book-Hunt!</string>
    <string name="help">Help!</string>
    <string name="found_my_book">Found my Book!</string>
    <string name="mark_my_book">Mark my book</string>

</resources>

```

REFERENCES

References used in the document are listed below. These Books have helped in writing this documentation clearly.

1. Sand, S., Dammann, A., & Mensing, C. (2014). *Positioning in Wireless Communications Systems*. Hoboken: Wiley
2. Karimi, H. (2015). *Indoor Wayfinding and Navigation*. Hoboken: CRC Press.
3. W., R. (2015). *Learning Android Google Maps*. Birmingham: Packt Publishing.
4. Horton, J., Vasconcelos, H., & Portales, R. (2016). *Android: Programming for Developers*. Packt Publishing.
5. Ruiz, A. (2015). *Mastering Android Application Development*. Birmingham: Packt Publishing