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
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
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. Architectural 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 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

- **IARRegion.Listener**
  It is used for receiving notifications when the region has been changed
Classes of IndoorAtlas SDK

- **IAExtraInfo**
  It's an immutable class that contains extraneous SDK information which is returned by IALocationManager.getExtraInfo().

- **IAGeofence.Builder**
  It's a builder to configure IAGeofence.

- **IAGeofenceEvent**
  A class that describes a geofence event.

- **IAGeofenceRequest**
  It provides the list of geofences to monitor the SDK

- **IALocation**
  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.

• IARRegion

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.

```xml
<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]

```java
private void setupGroundOverlay(IFloorPlan floorPlan, Bitmap bitmap) {

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

    if (mMap != null) {
        BitmapDescriptor bitmapDescriptor = BitmapDescriptorFactory.fromBitmap(bitmap);
        ILatLng ilatLng = floorPlan.getCenter();
        LatLng center = new LatLng(ilatLng.latitude, ilatLng.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, IALocationManager 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]

```java
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
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.

```gradle
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.

```xml
<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:

```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*.

```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.

```groovy
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.

```java
```

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

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
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.

![IndoorAtlas Login Credentials](image)

**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’.

![Create Location](image1)

Figure 15. Create Location

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

![Create Location Page](image2)

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.

- 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.
In this project, we have uploaded the floorplans of four floors.
Each floor has its unique floorId which is useful in our android code to distinguish between other Floor plans.
• 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 Waypoints

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 Waypoints

• 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](image)

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](image)
Data Flow Diagram of User

Figure 25. Data Flow Diagram of User
• Selecting a Book

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

• Check Location on Floor plan

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.

Figure 26. Checking Location on 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](image)

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](image)
• 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 with 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](image-url)
• 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.

![Found my book button](image)

*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.

![Thank you message](image)

*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.

![Please select your Book!](image)

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 diagram 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.
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.

```java
dependencies {
    compile 'com.indooratlas.android:indooratlas-android-sdk:{{site.version.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.
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.

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

```java
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.

```xml
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="AIzaSyAXAv16xcPlyq4CyExbQEvyp1DN6F5i7s" />
```

**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:

```java
private final int CODE_PERMISSIONS = 1...

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    String[] neededPermissions = {
        Manifest.permission.ACCESS_FINE_LOCATION,
        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:

```java
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_FINE_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_FINE_LOCATION);
        }
    }
}
```

@override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
    switch (requestCode) {
    case REQUEST_CODE_ACCESS_FINE_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.

```java
@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 ILocationManager you can now get periodic updates of the user’s location. The periodic updates of the user’s location is generated by the ILocationListener interface which has a onLocationChanged() callback method. onLocationChanged method updates the user's location as he moves
private ILocationListener mLocationListener = new ILocationListener()

@Override
public void onLocationChanged(ILocation location)

Log.i(TAG, "latitude: " + location.getLatitude());
Log.i(TAG, "longitude: " + location.getLongitude());
Log.i(TAG, "floor number: " + location.getFloorLevel());

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

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

Figure 50. Fetching Location Updates

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]
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 52. Fetch floor plan code

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.

```java
private IARegionListener mRegionListener = new IARegionListener() {
    // 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 entering a new location
            Log.d(TAG, "Location Changed to " + region.getName());
        }
    };

    @Override
    public void onExitRegion(IARegion region) {
        if (region.getType() == IARegion.TYPE_FLOOR_PLAN) {
            mCurrentFloorPlan = null;
            // notice that a change of floor plan (e.g., floor change)
            // is signalled 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

```java
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.

```java
private IARegion.Listener mRegionListener = new IARegion.Listener()
{

    @Override
    public void onEnterRegion(IARegion region) {

        if (region.getType() == IARegion.TYPE_FLOOR_PLAN) {
            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(mOverlay.FloorPlan)*/) {
                mCameraPositionNeedsUpdating = true; // entering new fp, need to move camera
                if (mGroundOverlay != null) {
                    mGroundOverlay.remove();
                    mGroundOverlay = null;
                }
                mOverlay.FloorPlan = region; // overlay will be this (unless error in loading)
                fetchFloorPlan(newId);
            } else {
                mGroundOverlay.setTransparency(0.0f);
            }
        }
    }

    Log.d(TAG, "Enter ": region.getType() = IARegion.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());
}

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]

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

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

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

// our camera position needs updating if location has significantly changed
if (mCameraPositionNeedsUpdating) {
    mMap=nullCamera(CameraUpdateFactory.newLatLngZoom(location, 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]

**Waypoints**
eeb9b503-b153-49fa-9e43-7549018ec82f

![Add waypoints by clicking the floor plan.](image)

- **Jupiter**
  - 34.18248386, -117.32389525

- **Venus**
  - 34.18272183, -117.32370548

Figure 59. Book Waypoints in IndoorAtlas web app
In the Android code, each book has been assigned its own Activity, for example: MercuryActivity.java.

![Image of file structure]

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.
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.

```javascript
Latlng mercury_book = new Latlng(34.18249717, -117.32396673);

Figure 61. Waypoint Book Location
```

```javascript
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]
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
First, Select the book from the Drop down menu.

On the Second Page, Read the message and start moving towards the mentioned Floor.

Hit on 'Mark my Book' button after reaching the right floor. Swipe Refresh to read message again.

Move towards your book and Hit 'Found my Book'

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
Figure 74. Screen of Third Page- After hitting Found my book button
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.
1. MainActivity.java

```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.ILocation;
import com.indooratlas.android.sdk.ILocationListener;
import com.indooratlas.android.sdk.ILocationManager;
import com.indooratlas.android.sdk.ILocationRequest;

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

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

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

    mLocationManager.requestLocationUpdates([ILocationRequest,create()], this);
}

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

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

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

@Override
public void onStatusChanged(String provider, int status, Bundle extras) {
    if (status != LocationManager.STATUS_SHOULD_INFORM_USER) {
        Toast.makeText(this, status + "", Toast.LENGTH_SHORT).show();
    }
}
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_FINE_LOCATION);
                        }
                    })
                .setNegativeButton(R.string.permission_buttondeny, 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_FINE_LOCATION);
}

@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
    switch (requestCode) {
    case REQUEST_CODE_ACCESS_FINE_LOCATION:
        if (grantResults.length > 0)
            if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                Toast.makeText(this, "Permission granted thank you!",
                        Toast.LENGTH_LONG).show();
            };
    break;
    }

    @Override
    public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l) {
        final Intent intent;
        switch(i){
            case R:
                break;
            case 1:
                intent = new Intent(MainActivity.this, CommonActivity.class);
                intent.putExtra("someVariable", "1");
                startActivity(intent);
1. **MainActivity.java**

```java
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

```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/apk/res-auto"
    tools:context=".MainActivity"...>
3. CommonActivity.java

```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.ILocationListener;
import com.indooratlas.android.sdk.ILocationManager;
import com.indooratlas.android.sdk.ILocationRequest;
import com.indooratlas.android.sdk.IARRegion;

import com.indooratlas.android.sdk.resources.IAFloorPlan;
import com.indooratlas.android.sdk.resources.ILatLng;
import com.indooratlas.android.sdk.resources.ILocationListenerSupport;
import com.indooratlas.android.sdk.resources.IResourceManager;
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 IARRegion mOverlayFloorPlan = null;
    private GroundOverlay mGroundOverlay = null;
    private IALocationManager mIALocationManager;
    private IAResourceManager mAResourceManager;
    private IATask<IFloorPlan> mFetchFloorPlanTask;
    private Target mLoadTarget;
    private boolean mCameraPositionNeedsUpdating = true; // update on first location

    /* Listener that handles location change events. */
    /* Location changed, move marker and camera position. */
    @Override
    public void onLocationChanged(final IALocation 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;
        }
    }
}


```java
};

LatLng latLng = new LatLng(location.getLatitude(),
location.getLongitude()); // Location of user from TA 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, 6));
    mCameraPositionNeedsUpdating = false;
}

};

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

@Override
public void onEnterRegion(IARRegion region) {
    if (region.getType() == IARRegion.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(255);
        }
    }

    Log.d(TAG, "Enter " + region.getType() + (region instanceof IARRegion.VENUE ? " VENUE " : " FLOOR_PLAN " + region.getId()));
}

@Override
public void onExitRegion(IARRegion 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);
    }

    Log.d(TAG, "Exit " + region.getType() + (region instanceof IARRegion.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(true);

// instantiate ILocationManager and IResourceManager
mLocationManager = ILocationManager.create(this);
mResourceManager = IResourceManager.create(this);
final Button button = (Button) findViewById(R.id.buttoncommon);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        AlertDialog.Builder alertdialog = new
        AlertDialog.Builder(CommonActivity.this);
        alertdialog.setTitle("Location");

        switch (someVariable) {

            case "0":
                break;

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

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

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

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

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

                switch (someVariable) {

                    case "0":
                        break;
```java
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;
}

```
```
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 = alertDialog.create();
alert.show();
@Override
protected void onDestroy() {
    super.onDestroy();
    // remember to clean up after ourselves
    mLocationManager.destroy();
}

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

    // start receiving location updates & monitor region changes
    mLocationManager.requestLocationUpdates(new LocationRequest(),
            mListener);
    mLocationManager.registerRegionListener(mRegionListener);
}

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

//
// Sets bitmap of floor plan as ground overlay on Google Maps
//
private void setupGroundOverlay(AFloorPlan floorPlan, Bitmap bitmap) {
    if (mGroundOverlay != null) {
        mGroundOverlay.remove();
    }
}
if (mMap != null) {
    BitmapDescriptor bitmapDescriptor =
        BitmapDescriptorFactory.fromBitmap(bitmap);
    LatLng 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 IFloorPlan 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) {
                if (null)
            }

            @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();

    IATask<IFloorPlan> task = mResourceManager.fetchFloorPlanWithId(id);
    task.setCallback(new IAsyncResultCallback<IFloorPlan>() {

        @Override
        public void onResult(IAsyncResult<IFloorPlan> 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;
    }
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_width="match_parent"
                    android:layout_height="match_parent">
        <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android" 
                        android:id="@+id/layout">
            <Fragment xmlns:android="http://schemas.android.com/apk/res/android"
                      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.mehapantam.runtime.MapActivity"
                      android:layout_alignParentStart="true" />

            <Button
                android:id="@+id/buttoncommon"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignParentEnd="true"
                android:layout_alignParentStart="true" />

        </RelativeLayout>
    </RelativeLayout>
</android.support.v4.widget.SwipeRefreshLayout>
5. MercuryActivity.java

    package com.example.snehaspantam.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.IALocationListenerManager;
    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;
```java
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 mercury_book = new LatLng(44.18299777, -117.92380873);

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

    private Marker mMarker;
    private IRegion mOverlayFloorPlan = null;
    private IGroundOverlay mGroundOverlay = null;
    private ILocationManager mLocationManager;
    private IResourceManager mResourceManager;
    private IAISimpleTask<IAFloorPlan> mFetchFloorPlanTask;
    private ITarget 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(IALocation location) {

        if(points.size() == 0)
        {
            points.clear();
        }
```
polylines.get("polylineanenha").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.book_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, 20));
    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();
private IRegion.Listener mRegionListener = new IRegion.Listener() {

@Override
public void onEnterRegion(IRegion region) {
    if (region.getType() == IRegion.TYPE_FLOOR_PLAN) {
        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.setTransform(330);
        }
    }
}

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

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

private void polylineMethod() {

PolylineOptions polylineOptions = new PolylineOptions().width(6).color(RED).geodesic(true);

polylines.put("polylineaneha", mMap.addPolyline(polylineOptions.add(points.get(0), mercury_book)));
}

@Override
protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);
setContentView(R.layout.activity_mercury);
SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager()
.findViewById(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(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 alertDialog = new
        AlertDialog.Builder(MercuryActivity.this);
        alertDialog.setTitle("Thank you for using our App!");
        alertDialog.setPositiveButton("OK", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                Intent intent = new Intent(MercuryActivity.this, MainActivity.class);
                startActivity(intent);
            }
        });
        AlertDialog alert = alertDialog.create();
        alert.show();
    }
});

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

@Override
protected void onResume() {

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

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

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

//}

/**
 * Sets bitmap of floor plan as ground overlay on Google Maps.
 */
private void setupGroundOverlay(ALfloorPlan floorPlan, Bitmap bitmap) {
    if (mGroundOverlay != null) {
        mGroundOverlay.remove();
    }
    if (mMap != null) {
        BitmapDescriptor bitmapDescriptor =
            BitmapDescriptorFactory.fromBitmap(bitmap);
        LatLng center = floorPlan.getCenter();
        GroundOverlayOptions fpOverlay = new GroundOverlayOptions()
            .image(bitmapDescriptor)
            .position(center, floorPlan.getWidthMeters(), floorPlan.getHeightMeters())
            .bearing(floorPlan.getBearing());
        mGroundOverlay = mMap.addGroundOverlay(fpOverlay);
    }
}
private void fetchFloorPlanBitmap(fina1 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;
            }
        };
    }
    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);
}
```java
/**
 * 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

```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:layout_alignParentTop="true"
    android:layout_alignParentStart="true">

    <fragment
        android:id="@+id/mapFragment"
        android:name="com.google.android.gms.maps.SupportMapFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentTop="true"
        android:layout_alignParentStart="true"
        tools: psychiatry="true"/>

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentEnd="true"
        android:layout_alignParentTop="true"
        android:text="Select Location"
        android:layout_gravity="start"
        android:layout_weight="1"
        android:background="@android:color/white"
        android:textColor="@android:color/colorAccent"
        android:textColorDrawable="@android:color/white" />

</RelativeLayout>

</android.support.v4.widget.SwipeRefreshLayout>
7. AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.androidpantn.runtime">
    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
    <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:required="true"
        android:name="com.google.android.PermissionAndAppPermission" />

    <application android:allowBackup="true" android:icon="@mipmap/ic_launcher_round" android:roundIcon="true" android:label="@style/AppTheme">
        <activity android:name="com.example.androidpantn.RuntimeActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```
The API key for Google Maps-based APIs is defined as a string resource. Give the title "com.example.google.maps.api.key".

Note that the API key is linked to the encryption key used to sign the APK.

We need a different API key for each encryption key, including the default key that is used to sign the APK for publishing.

You can include the keys for the default and chosen values in your/app/src/main/res/values/.

<meta-data>
  <name>com.example.google.maps.api.key</name>
  <value>AIzaSyA_yoFyFyDhp1D6659z7194</value>
</meta-data>

The API key for Google Maps-based APIs is defined as a string resource.

Give the title "com.example.google.maps.api.key".

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 default key that is used to sign the APK for publishing.

You can define the keys for the default and chosen values in your/app/src/main/res/values/.

<meta-data>
  <name>com.example.google.maps.api.key</name>
  <value>AIzaSyA_yoFyFyDhp1D6659z7194</value>
</meta-data>
8. Strings.xml

```xml
<resources>
    <string name="app_name">Book-Run</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_goolglemap_basic_title">03. Google Maps - Basic</string>
    <string name="example_goolglemap_basic_description">Demonstrates showing location on GoogleMaps</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_img_overlay_title">05. Overlay with Open Street Map</string>
    <string name="example_img_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="dialog_set_location_title">Set location</string>
    <string name="error_could_not_set_location">Error setting location</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">Hi: %s</string>
    <string name="menu_set_channel">Set channel</string>
    <string name="channel_dialog_title">Channel name: %s</string>
    <string name="error_setting_channel">Setting channel failed: %s</string>
</resources>
```
```xml
<setting name="example_sharelocation_description" Demonstration sharing locations via 3rd party cloud service. /></string>
<setting name="example_sharelocation_title" 06. Location sharing? /></string>
<setting name="example_change_color" Change color? /></string>
<setting name="example_change_color" Change color? /></string>
<setting name="example_credentials_title" 06. Set credentials? /></string>
<setting name="example_credentials_description" Demonstrates setting credentials from code? /></string>
<setting name="dialog_request_options_title" Location request options? /></string>
<setting name="dialog_request_options_label" Fastest interval (milliseconds) /></string>
<setting name="shortest_displacement_label" Shortest displacement (meters) /></string>
<setting name="default_label" was default? /></string>
<setting name="example_region_description" Demonstrates automatic region transitions and vertical position information /></string>
<setting name="example_regions_title" 07. Regions? /></string>
<setting name="region_information" Region information? /></string>
<setting name="venue_outside" Outside mapped area? /></string>
<setting name="venue_inside" In venue? /></string>
<setting name="floor_plan_inside" No floor plan? /></string>
<setting name="floor_level" Floor level? /></string>
<setting name="floor_certainty_percentage" Certainty: % /></string>
<setting name="IndoorAtlas_floor_plan_id" /></string>

```
<items>
  <item>Venus</item>
  <item>Earth</item>
  <item>Mars</item>
  <item>Jupiter</item>
</items>
REFERENCES

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


