**Software Design**

**Document**

**for**

**Augmented Reality for Hydrology**

**Version 3.0 approved**

**Prepared by Christopher Hung Nguyen, Ernesto Padilla, Cuong Pham, Wilbert Veit, Kaicheng Zhou**

**Jet Propulsion Laboratory (JPL)**

**September 5, 2017**

Table of Contents................................................................................................................... 1

Revision History..................................................................................................................... 2

1. Introduction................................................................................................................ 3
   1. Purpose........................................................................................................... 3
   2. Document Conventions……………………................................................. 3
   3. Intended Audience and Reading Suggestions................................................ 4
   4. System Overview........................................................................................... 4
2. Design Considerations............................................................................................... 5
   1. Assumptions and dependencies..................................................................... 5
   2. General Constraints........................................................................................ 6
   3. Goals and Guidelines..................................................................................... 7
   4. Development Methods................................................................................... 7
3. Architectural Strategies.............................................................................................. 8
4. System Architecture................................................................................................... 9
   1. Architectural Overview.................................................................................. 9
   2. System Overview........................................................................................... 10
5. Policies and Tactics.................................................................................................... 12
   1. Specific Products Used.................................................................................. 12
   2. Requirements traceability............................................................................... 12
   3. Testing the software....................................................................................... 12
   4. Engineering Trade-Offs.................................................................................. 13
   5. Guidelines and conventions........................................................................... 14
   6. Maintenance................................................................................................... 14
   7. Future Extension/Enhancements.................................................................... 14
6. Detailed System Design............................................................................................. 15
7. Detailed Lower level Component Design.................................................................. 21
8. Database …………………………………………………………………………. 23
9. Overview of User Interface………………………………………………………….. 25
10. Requirements Validation and Verification................................................................. 28
11. Glossary..................................................................................................................... 35
12. References................................................................................................................. 36

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Date | Reason For Changes | Version |
| First Draft | 11/27/2017 | First initial draft of the document | 1.0 |
| Second Draft | 04/19/2018 | Second draft of the document | 2.0 |
| Final Draft | 05/05/2018 | Final draft of the document | 3.0 |
|  |  |  |  |

**1. Introduction**

Jet Propulsion Laboratory (JPL) is world-renowned for their contributions on the advancement of robotics and spacecrafts, but they also do a lot of research on geographical computing as well. Watertrek is a database of Hydrology data that JPL has constructed and maintained, and it contains a lot of data related to the Earth’s different water-containment facilities such as wells, reservoirs, rivers, and lakes. JPL has worked tirelessly on new methods to provide and visualize this data to scientifically-minded audience who may not have a data science background. One of the visualization techniques that JPL has decided upon is Augmented Reality, and it is with this goal that JPL has partnered up with California State University Los Angeles to create an Augmented Reality framework that they can use with Watertrek, as well as other scientific data that contains geolocation information.

A general-purpose Augmented Reality framework that can visualize data with geolocation information shall be developed and delivered for this project. This framework aims to simplify the development process of Augmented Reality applications, eliminate the necessity of the user/developer having prior knowledge of the various sensors and computer graphics features available on the target development platform, and provide the foundation for more customization and potential expansion of the framework to many other platforms with similar capabilities.

An app shall also be developed by implementing the framework mentioned above in conjunction with a concurrently developed data library that retrieves Hydrology data from Watertrek. This app shall be a showcase of one specific use case of the framework, and it shall allow for Augmented Reality visualization of all Hydrology data that is currently available from Watertrek.

**1.1 Purpose**

The purpose of this document is four-fold:

a) Define a full set of requirements for the Augmented Reality Framework for Hydrology.

b) Define the design for the Augmented Reality Framework for Hydrology.

c) Define the Test Plan for the Augmented Reality Framework for Hydrology..

d) Define implement feasible modules for the Augmented Reality Framework for Hydrology.

The complete definition of all Augmented Reality Framework for Hydrology requirements provides the source requirement inputs for the development of the subsequent supporting software subsystems documents.

**1.2 Document Conventions**

This document follows MLA format. Bold-faced text is meant to emphasize section headings, and italicized text is to denote important technical terms that you can find also in the Glossary. Every requirement statement in this document has equal priority.

**1.3 Intended Audience and Reading Suggestions**

This document is intended for software developers, testers, and project managers. This document contains a full description of the design and implementation of both the framework and the app as mentioned above. Section 1, which is this section, contains an overview. Section 2 contains all of the considerations the team has made for the project. The sections from 3 and onward, up to 9 describe fully the design and implementation of the system. Section 10 is a table showing how specific requirements are addressed by the implementation. Section 11 is a Glossary of terms. For developers, sections from 3 to 9 may be of interest. For testers, section 10 may be of interest. For project managers, section 1, 2 and 11 may be of interest.

**1.4 System Overview**

This project contains two different systems: a framework and an app. The framework is intended for software developers, and it contains high-level interfaces that allow developers to implement an augmented reality app of their own design, and these interfaces have been designed with the goal of simplifying the system such that the developers will not need to have prior knowledge of location/motion sensors, or computer graphics features. The app is intended for scientifically-minded individuals to visualize the Hydrology data that is provided by Watertrek, and it contains a simple and intuitive user interface that allows them to toggle what kind of data they would like to visualize. In this document, components that are specific to the app (not a part of the framework) shall not be denoted as such. Components that are specific to the framework and are implemented/utilized by the app shall be denoted in their headings as being a part of the framework.

**2. Design Considerations**

This section contains and describes many of the issues that we faced while coming up with a design for both the framework and the app.

2.0.1 Use of this software around areas of magnetic interference will result in information of high inaccuracy and may display incorrect information.

2.0.2 A preload or caching of hydrology data may be performed before venturing to areas without a stable internet connection.

2.0.3 Large queries in data may see an increased wait time in its response.

2.0.4 Varying load time means the system needs to be able to handle loading/unloading data and visualization simultaneously and arbitrarily.

**2.1 Assumptions and Dependencies**

Assumptions and dependencies for the App:

2.1.1. The device must be running the Android mobile operating system with at least version 5.0 (or SDK level 21).

2.1.2. The device must be capable of functioning with OpenGL ES version 2.0.

2.1.3. The device is assumed to have a suite of functioning sensors including Accelerometer, Magnetometer, Gyroscope, and GPS.

2.1.4. The device is assumed to have a touch screen.

2.1.5. The device is assumed to have a functional camera.

2.1.6. The device is assumed to have a functional network connection. This can be either a WIFI connection or a cellular network connection.

Assumptions and dependencies for the Framework:

2.1.7. The developer is assumed to have access to Android Software Development Kit.

2.1.8. The developer is assumed to be familiar with the basics of Android app development.

2.1.9. The developer is assumed to be familiar with the basics of geolocation information.

**2.2 General Constraints**

2.2.1. A network connection is required. The preload of data also requires an initial connection to load region specific.

2.2.2. Watertrek hydrology data currently only contains data for the western United States. Therefore, only several states may have a functioning data.

2.2.3. Mobile devices may not have enough processing power for more advanced computational tasks so surface tracking is very constrained, as evident in Google’s attempt with their ARCore framework.

2.2.4. Even though most modern phone devices should support OpenGL ES 2.0, their implementation is different and may require different approaches to presenting 3D graphics altogether. Graphical bugs are almost unavoidable.

2.2.5. Different phones use different combinations of accelerometer/magnetometer/gyroscope with varying levels of accuracy so movement tracking may not be very accurate.

2.2.6. GPS sensor has big error range (up to 30 ft) so the user’s location may not be accurate or even stable with some devices.

2.2.7. Android Studio continually comes out with new versions and updates that may sometimes change the development environment in a very major way, which means rapid maintenance of the code base is necessary for the framework to be able to function with newer versions of the Android SDK.

2.2.8. A new Android version may deprecate something that the framework uses so at some point, a separate fork may be necessary for the framework to maintain support with the majority of the Android user base.

2.2.9. Watertrek is currently not accessible to the general public, and it requires secure authentication in order for the app to retrieve data. As a result, the app should only be for internal use between Cal State LA and JPL until security improves or changes.

2.2.10. Watertrek is still currently under development so at any point, certain functionalities in the app may stop working and will require coordination with JPL in order to restore.

**2.3 Goals and Guidelines**

Goals and guidelines for the App:

2.3.1. The Augmented Reality application for visualizing JPL’s Watertrek Hydrology data shall be delivered by May 2018.

2.3.2. The app shall work on current Android devices with camera, GPS and motion sensors.

2.3.3. The app shall be able to visualize Hydrology data that is currently available via Watertrek.

2.3.4. The app shall have a user interface that supports viewing different data types.

Goals and guidelines for the Framework:

2.3.5. The general-purpose Augmented Reality framework shall be delivered by May 2018.

2.3.6. The framework shall support Android version 5.0 and above.

2.3.7. The framework shall allow for visualization of a large amount of data all at once

2.3.8. The framework shall perform at reasonable performance on devices that can support Android version 5.0 and above.

2.3.9. The framework shall support the base functionality without any major crash (bugs may still exist).

**2.4 Development Methods**

In this project, an Agile Development Scheme will be used. The characteristics of our workflow are as follows:

* List desired features upfront: Together with our Customer (JPL) we will make a list of features that should be a part of the app or the framework. The customer will tell us which features have highest priority to them, while we determine how much research/work will be required to implement each feature.
* Implement features:
  + Customer determines next feature that is important to them.
  + We add feature to either app or framework.
  + If it was a framework component, update the app to use this feature.
  + Test the new feature in isolation.
  + Test the feature as part of the app.
  + Repeat

**3. Architectural Strategies**

This project contains two separate components: a framework, and an app that utilizes that framework. Each of these components follows a different strategic path:

Strategies for the Framework:

* Certain platform-specific components are unavoidable as they are necessary for proper integration, as well as for maintaining performance and compliance with system services. To that end, efforts shall be made to detach these components from the core visualization components as much as possible.
* The core visualization components that are responsible for Augmented Reality are detached from the visualization components. These are typically mathematical operations, computer graphics primitives, and other modules that support them. The goal here is to keep them well abstracted from the platform such that they can be reusable with a different platform in the future.
* If there is any module that is common between the current target platform and another platform (i.e.: both platforms provide a similar interface), then the commonality can be considered and such a module does not necessarily need to be abstracted. An example of this would be the OpenGL ES rendering directives. OpenGL ES has become a de facto standard in computer graphics, so it is available on all mobile platforms, and the functions provided behave the same both on mobile devices and on desktop.
* Callbacks and inheritable interfaces are provided where applicable and as much as possible to allow for future expansion and customization. The goal here is two folds: to provide an interface for the developer who uses the framework to be able to implement his own design without having to tear down the framework and rebuilding it from scratch; and to provide the baseline for future extensions/enhancements without requiring the core components to be significantly altered or modified. This hopefully will make future versions of the framework still compatible with implementations that were built using an older version, while still providing more features.

Strategies for the App:

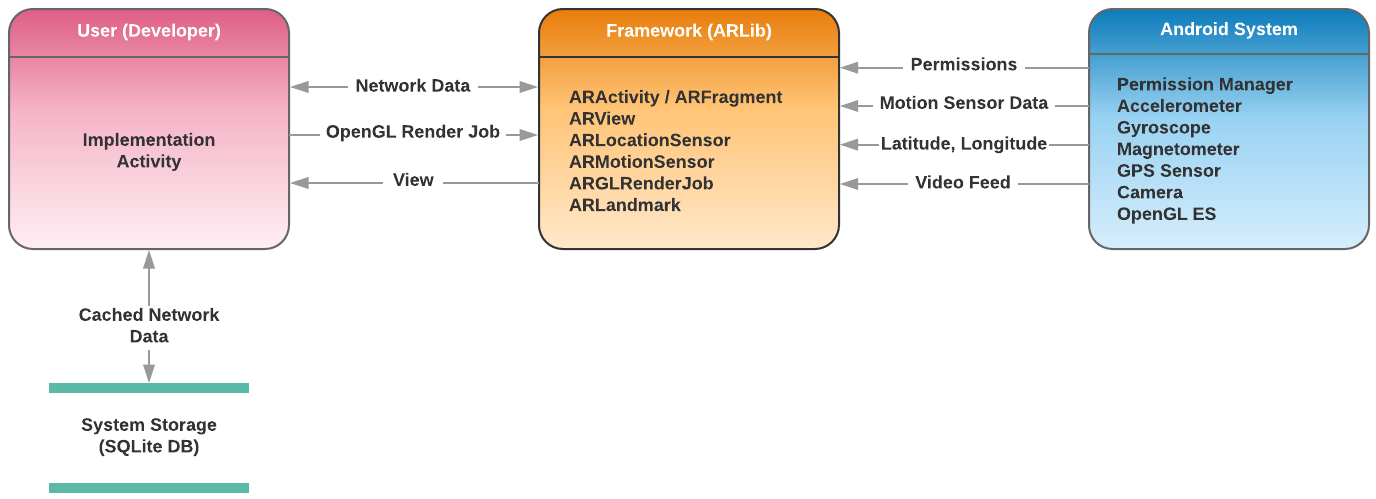
* Follows the principle of KISS (Keep It Simple Stupid).
* Aside from the data framework, no extra work shall be necessary to implement the basic functionalities of Augmented Reality visualization.
* The data framework shall minimize network utilization as much as possible.

**4. System Architecture**

**4.1 Architectural Overview**

The architecture for both the Framework and the App is the same and can be broken down into three main factors: the User (software developer), the Framework (ARLib), and the Android system.

Here is a diagram (DFD level 0) that shows how this architecture works at a high level:



* **The User (Developer):** this is a factor in the architecture because considerations need to be made on how the Framework will be used. To that end, one of the team members, Christopher Hung Nguyen, acts as the user for the project, and his work on the App embodies this factor. Whatever the user requires in order to implement his App is exactly the base functionalities that we have to provide in the Framework.
* **The Framework (ARLib):** this is the main goal of the project. The Framework provides functionalities that the User can utilize to implement their App. Also, the Framework acts as an intermediary between the User and the Android System to abstract and take care of many of the setup procedures that the User would otherwise have to do by themselves. For instance, the Framework automatically takes care of setting up permissions, motion sensor data, camera, and OpenGL ES rendering, so the user does not have to go through the trouble of doing that.
* **The Android System:** through the use of Android SDK, many interfaces are available that allows for more convenient access to hardware features that are available on the device. For instance, Android provides an Orientation sensor, a mixture of the accelerometer, magnetometer, and gyroscope depending on how the system is setup (this is potentially different depending on the hardware of the device), so this sensor can be used directly without us having to implement our own code to do about the same thing.

**4.2 System Overview**

Here is an overview of the Framework as a system, and how it connects to the App (Implementation Activity), incidentally, this is also our DFD level 1:



There are six major modules in this system. They are described in more details in section 6. Here is a brief overview of them:

**4.1. ARFragment / ARActivity:** these are separate classes but they act as one module, as the functionalities they provide are almost the same, and it comes down to whichever style the developer prefers. These components are specific to the Android interface or subsystem, and as such, they are provided only as a convenience for the user. Both of them automatically take care of setting up permissions, sensors, camera, and OpenGL rendering so the user just has to implement what they would like.

**4.2. ARView / ARCameraView:** both of these are, again, the same and they act as one module. These components are implemented by ARFragment or ARActivity, respectively, to provide a camera view and an interface for the motion sensors. The user may choose to directly implement either of these views to create their own activity or fragment with a custom behavior that is more tailored toward their needs.

**4.3. ARLocationSensor:** this is an interface that abstracts the GPS sensor’s functionalities. It is further abstracted by either the ARFragment or ARActivity to provide data to the user without making them go through the trouble of setting up permissions for this.

**4.4. ARMotionSensor:** this is yet another interface that abstracts the motion sensors’ functionalities. It is further abstracted by either the ARView or ARCameraView to provide data to the user without making them go through the trouble of having to set it up.

**4.5. NetworkUtils:** this is an interface that is specific to the app, and is not part of the Framework. This interface allows the user to make a connection to the internet without having to go through the procedures of having to set it up. It can be directly referenced by the user, but in this system, it is utilized by the \*Service classes to provide data for the user.

**4.6. \*Service classes:** these are described in more details in section 6.2.0. Essentially, these are the various service classes that provide an interface to connect to the Watertrek database and retrieve Hydrology data.

**5. Policies and Tactics**

**5.1 Choice of which specific products used**

* With Android development, Android Studio and the SDK were decided upon as they allow for native development on the platform, and therefore can allow for the highest level of performance tweak compared to using other third party solutions such as Unity.
* Java was chosen as the main development language because of its popularity. Cal State LA predominantly features Java as the main development and education language in the Computer Science program, so the choice of going with Java will allow Cal State LA students to familiarize themselves with the framework faster than if any other language was used. This may prove to be crucial if this project shall continue to be a senior design project for the next generation of computer scientists at Cal State LA.
* SQL was chosen as the database language as Android supports SQLite natively as its default storage and database management system, so using SQL allows for native development and achieves the highest level of performance with the system. Also, Cal State LA teaches SQL as one of the most basic database systems so this decision will also carry potential future development implications.
* The Android SDK provides many interfaces that allow for easier integration with certain low-level devices on mobile devices, such as motion sensors, location sensor, and the camera. These interfaces are utilized where they are applicable to reduce the complexity of the framework, and their utilization is separate from the main visualization components of the framework so the framework maintains high-level components that can be carried over to another platform.
* Watertrek RESTful API already has a lot of functionalities provided that takes care of some geolocation math, so we do not have to re-implement the same functionalities in the app. We may still have to implement some functionalities not currently provided by Watertrek such as digital elevation.

**5.2 Plans for ensuring requirements traceability**

* Section 10 has a table which describes all of the requirements and which modules satisfy them.

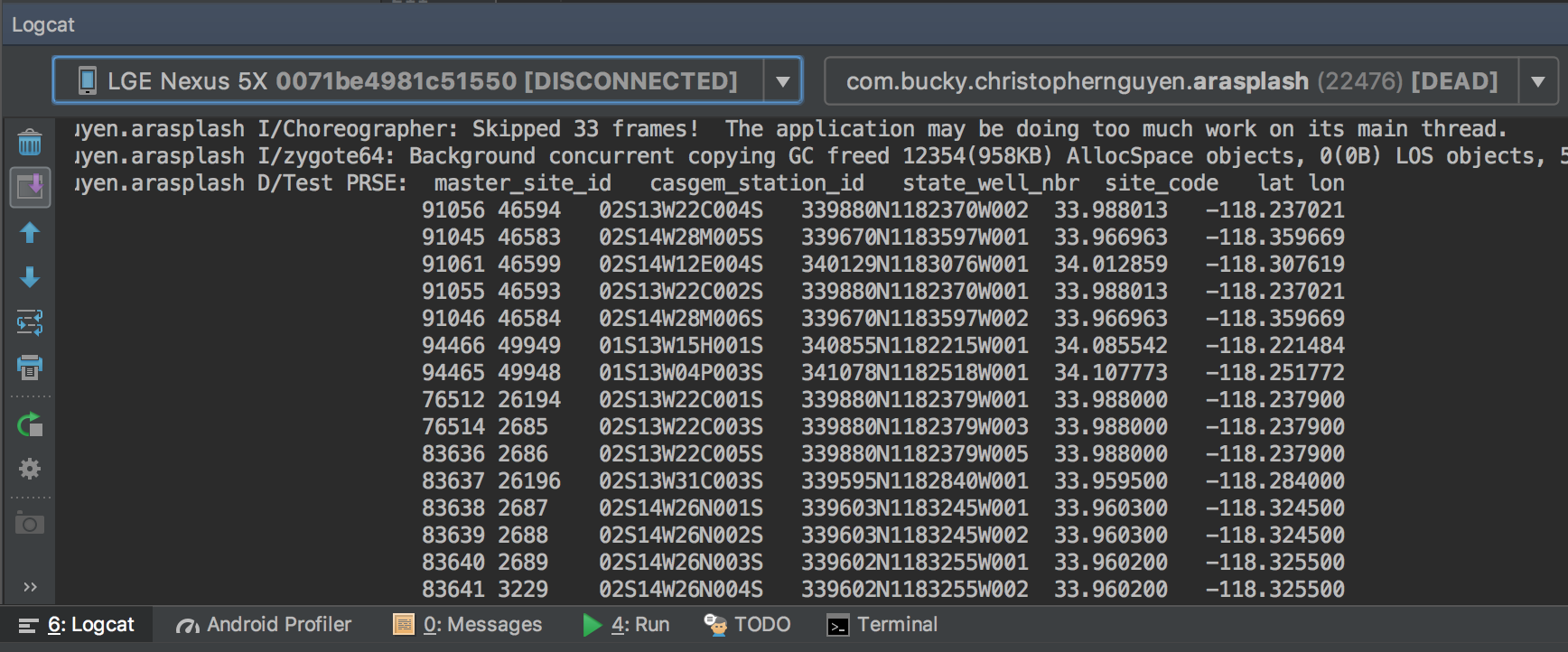
**5.3 Testing the Software**

Android Studio features many tools that aid in the testing process. Of note are the following:

5.3.1 Logcat:

Logcat shows system messages as well as messages specific to the currently running app, so it can be used to print data output in text form from an actual device, and it can also be used to trace back an error or exception to wherever in the code it occurred.

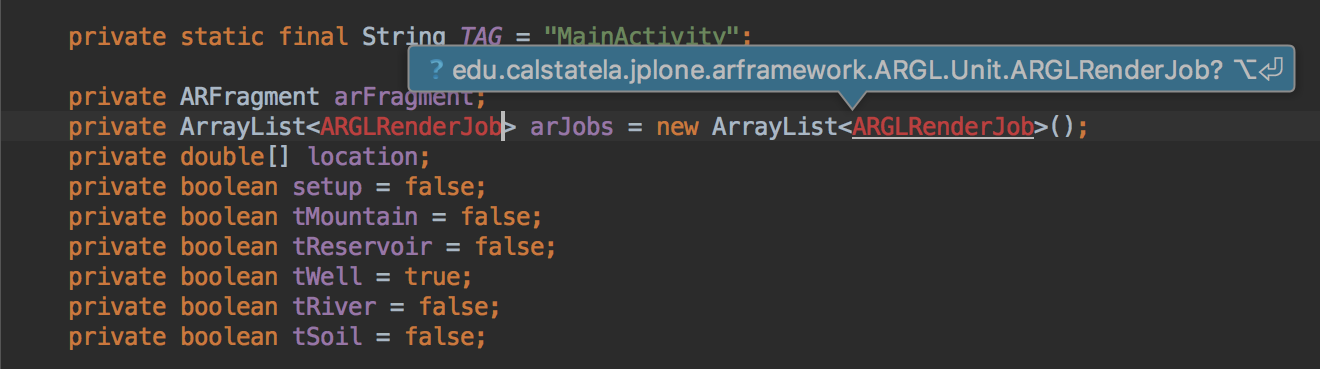
Sample Logcat output showing retrieved data:



5.3.2 Android Studio built-in code editor:

The built-in code editor of Android Studio is quite good at catching errors, and suggesting solution. It is quite possibly the second most used tool for error testing, and it is responsible for reducing a significant portion of errors before the code gets compiled.

Sample error catching and suggestion by the code editor:



**5.4 Engineering Trade-Offs**

As mentioned, some engineering trade-offs were made when taking into consideration the Watertrek REST API resources. When creating network services we decided that something that was easily readable by humans was more important over something that was completely abstract, and therefore more reusable. We made similar concessions when designing our objects and classes later used to bind the hydrology data. We also removed a limiter from our search query to accommodate developers wanting to test large areas.

**5.5 Coding Guidelines and Conventions**

The CamelCase naming convention was observed during the creation of the AR framework and the AR application.

**5.6 Maintenance**

As with any other framework, maintenance is expected. The AR framework and the AR Application components were designed in such a way as to be easy to use and maintain. The data service classes do rely on JPL REST API and as the sponsors of this product, will maintain the connectivity with the data service module.

**5.7 Future Extension/Enhancements**

* Adding more types of Drawables: In the Rendering System, there is a class called Drawable. Drawables are items that have a draw() method. When the draw method is called the appropriate OpenGL calls will be made in order to draw the 3D object. Anything that can be drawn will extend the Drawable class. We will provide some basic types of Drawables such as a wireframe drawable, a shaded drawable, a billboard drawable on which 2D drawings can be placed, etc. However, developers with OpenGL knowledge can add additional types of Drawables and they will just fit in with the rest of the Rendering System, as long as they extend the Drawable class.
* Currently, the framework allows for visualization of 3D models, and the app supports loading a digital elevation model (DEM) locally to augment the provided data with an elevation information for each geolocation. In the future, it would be nice if the DEM can be visualized as a 3D overlay in the app.
* A 3D overlay combined with computer vision techniques using OpenCV may potentially allow for more accurate tracking, and we may be able to recognize mountain shapes, well shapes, etc…
* Machine learning and Kalman filter can be used to improve motion and location sensors readings. This is a good potential area for some research.
* The user interface of the app can be polished to a higher level to allow for a more pleasing experience for the user.
* The framework is currently limited to the Android platform, but it can also be converted to iOS, another popular mobile platform with devices that come equipped with all of the hardware features that are necessary.

**6. Detailed System Design**

|  |  |
| --- | --- |
| **6.1.0 Data Service (Module)** |  |
| 6.1.1 Responsibilities | Data Service module is comprised of several classes that retrieve information for several “Points of Interest.” This includes, Reservoirs, Wells, Soil Moisture, and Snotel. |
| 6.1.2 Constraints | Depending on the version of the module a verification of credentials is done upon using any of the services. This may change from the current basic authorization. Depending on the size of the query, the response may vary in wait time. |
| 6.1.3 Composition | Data Service component provides classes that provide data for each of the P.O.I. |
| 6.1.4 Uses/Interactions | Developers will be able to access data and exposed resources on JPL’ servers. Watertrek Hydrology data will be provided. |
| 6.1.5 Resources |  |
| 6.1.6 Interface/Exports |  |

|  |  |
| --- | --- |
| **6.2.0 Data (Module)** | Data module contains data type objects for “Points of Interest.” This includes classes to bind objects of type: Reservoir, Well, Soil Moisture, and Snotel. |
| 6.2.1 Responsibilities |  |
| 6.2.1 Constraints |  |
| 6.2.2 Composition | Data Service component provides classes that provide data types and objects for each of the P.O.I. These were coded in Java. |
| 6.2.3 Uses/Interactions | Developers will be able to access premade data types that will allow them to quickly bind data to objects. |
| 6.2.4 Resources |  |
| 6.2.5 Interface/Exports |  |

|  |  |
| --- | --- |
| **6.3.0 AR Rendering Surface** |  |
| 6.3.1 Responsibilities | This module consists of a User Interface (UI) component for displaying AR. The application developer can use this component directly in their application, along with the Native Android UI components.  In order to display AR, two layers must be displayed.   1. The first layer is the camera layer. It comes directly from the device camera’s output. 2. The second layer (which is drawn on top of the first layer) is the OpenGL layer. The application programmer can make OpenGL calls and see the results displayed in the OpenGL layer. Alternately, the application programmer can make calls to another OpenGL based rendering API (such as our rendering system), and have the results displayed in the OpenGL layer. |
| 6.3.2 Constraints | When rendering 3D graphics in the OpenGL layer, the application programmer is limited to OpenGL es 2.0 method calls, or method calls from another API that is based on OpenGL es 2.0. |
| 6.3.3 Composition | ARActivity - AR Rendering Surface class that can be used everywhere an Android Activity can be used.  ARFragment - AR Rendering Surface class that can be used everywhere an Android Fragment can be used.  ARView - AR Rendering Surface class that can be used everywhere an Android View can be used. |
| 6.3.4 Uses/Interactions | To use one of the AR Surface components, simply create a new class that inherits from the desired type of surface (ARActivity, ARFragment, ARView), and then place it in the Android App.  To render onto the AR Surface component, override its GLInit(), GLResize(int, int), and GLDraw() methods with drawing code called from these methods. |
| 6.3.5 Resources |  |
| 6.3.6 Interface/Exports | Hardware |

|  |  |
| --- | --- |
| **6.4.0 Sensors** |  |
| 6.4.1 Responsibilities | Provides access to the device’s built in sensors. |
| 6.4.2 Constraints |  |
| 6.4.3 Composition | ARSensor - Class that provides access to device’s Accelerometer, Gyroscope, Rotation Vector, Magnetometer  ARSensor.Listener - Inner Class of ARSensor. Allows retrieving sensor data and specifying actions to be carried out when sensor changes.  ARGps - Class that provides access to device’s GPS.  ARGps.Listener - Inner class of ARGps. Allows retrieving sensor data and specifying actions to be carried out when new GPS coordinates are received. |
| 6.4.4 Uses/Interactions | Both ARSensor and ARGps are used in similar ways:   1. Create a subclass of ARSensor/ARGps 2. Create a subclass of ARSensor.Listener/ARGps.Listener and override the onSensorEvent(...)/handleLocation(...) methods. This is how sensor data is retrieved. The sensor data is passed in to this function as an argument. 3. Call stop() and start() methods to stop/start the sensor as needed. |
| 6.4.5 Resources |  |
| 6.4.6 Interface/Exports |  |

|  |  |
| --- | --- |
| **6.5.0 Rendering System** |  |
| 6.5.1 Responsibilities | Allow for the rendering of 3D objects, and placing them directly on the screen. |
| 6.5.2 Constraints |  |
| 6.5.3 Composition | ARGLDrawable - This class (and its subclasses) represents a 3D object that can be drawn (has a draw(...) method)). The draw function takes an entity/model matrix and a camera/view matrix as an argument, to properly place the Drawable during rendering.  ARGLEntity - Objects of this class keep track of the orientation and location of 3D objects. An ARGLEntity can be moved and rotated. Later, a matrix can be retrieved that represents the transformation of that ARGL Entity. Usually one ARGLEntity is used for each 3D object when drawing.  ARGLCamera - Objects of this class keep track of the position of the viewer of 3D objects. An ARGLCamera can be moved and rotated. Later a matrix can be retrieved that represents the transformation of that ARGLCamera. Usually, only one ARGLCamera is needed when drawing all the 3D objects.  ARGLScene - Class that is a collection of ARGLEntities. Allows simpler management/drawing of groups of ARGLEntities. Subclasses of ARGLScene can maintain special arrangements of ARGLEntities, such as keeping all the entities in a circle. |
| 6.5.4 Uses/Interactions | When setting up for 3D rendering (initially):   1. Create Drawables that need to be drawn. 2. Assign an Entity to each Drawable. Multiple Entities can be assigned to a Drawable, if that Drawable is going to be rendered in more than one location. 3. Place the Entities in a Scene 4. Create a Camera.   When drawing (each time a new frame is to be rendered):   1. Update the position/rotation of each Entity. 2. Update the position/rotation of the camera. 3. Obtain the camera matrix and call the Scene’s draw function with the camera matrix as an argument. This will cause all the Entities to call their Drawable’s draw functions with the camera matrix and the entity matris as arguments. |
| 6.5.5 Resources |  |
| 6.5.6 Interface/Exports |  |

|  |  |
| --- | --- |
| **6.6.0 Database - Local Storage** | Described in detail in section 8 |

|  |  |
| --- | --- |
| **6.7.0 User Interface** | Described in detail in Section 9 |

**7. Detailed Lower level Component Design**

**7.1** ReservoirService.java

7.1.1 Class

7.1.2 Used to gather primary and secondary data from JPL’s Watertrek resources. Only provides data for Reservoirs.

7.1.3 No interface

**7.2** SnotelService.java

7.2.1 Class

7.2.2 Used to gather primary and secondary data from JPL’s Watertrek resources. Only provides data for Snotel.

7.2.3 No interface

**7.3** WellService.java

7.3.1 Class

7.3.2 Used to gather primary and secondary data from JPL’s Watertrek resources. Only provides data for Wells.

7.3.3 No interface

7.3.4 Uses provided latitude and longitude to generate a polyline(circle). Also compensates for the variances in meridian and parallels along the poles and equator.

**7.4** SoilMoistureService.java

7.4.1 Class

7.4.2 Used to gather primary and secondary data from JPL’s Watertrek resources. Only provides data for Soil Moisture.

7.4.3 No interface

**7.5** Reservoir.java

7.5.1 Class

7.5.2 Custom Java object used to store Reservoir type data. Contains fields and attributes specific to Reservoir type.

7.5.3 No interface

**7.6** Snotel.java

7.6.1 Class

7.6.2 Custom Java object used to store Snotel type data. Contains fields and attributes specific to Snotel type.

7.6.3 No interface

**7.7** Well.java

7.7.1 Class

7.7.2 Custom Java object used to store Well type data. Contains fields and attributes specific to Well type

7.7.3 No interface

**7.8** SoilMoisture.java

7.8.1 Class

7.8.2 Custom Java object used to store Soil Moisture type data.

7.8.3 No interface

**7.9** River.java

7.9.1 Class

7.9.2 Custom Java object used to store River type data.

7.9.3 No interface

**7.10** DatabaseHelper.java

7.10.1 Class

7.10.2 Used to operate database, create/delete tables, insert/extract data from database.

7.10.3 No interface

7.10.4 Transfer data from hydrology data classes to local database.

**7.11** Network Task

7.11.1 Class

7.11.2 Custom class used to delegate asynchronous tasks and perform http requests with Watertrek resources.

7.11.3 No interface

**8. Database Design**

This app has one database that includes four tables, but could be extended to more.

Table Reservoir:

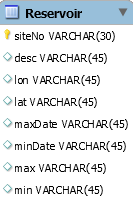
****

Table reservoir has elements that represent a Reservoir object in hydrology.

Table DBGS:

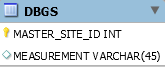
****

Table DBGS is a supplemental table of Table Well that store a list of measurement data of a single well.

Table Well:

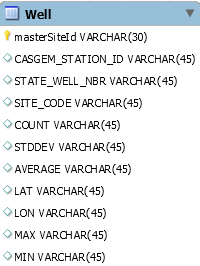
****

Table Well has elements that represent a well object in hydrology.

Table Log:

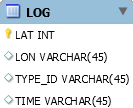
****

Table Log records when and where the application make a rest-api call. User can use the record as reference to decide whether a new api call is going to fire or not.

**9. User Interface**

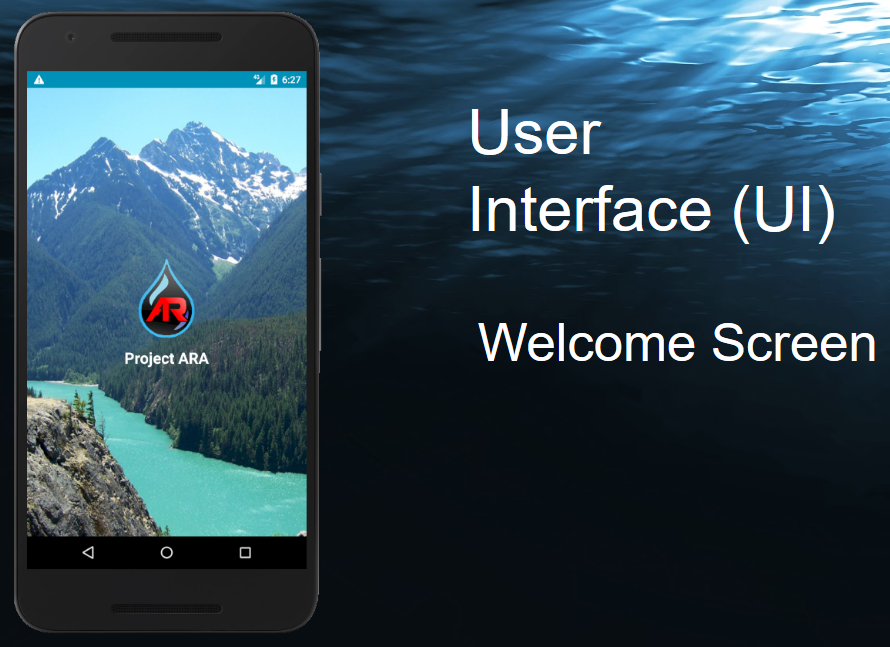
**9.1 Overview of User Interface**

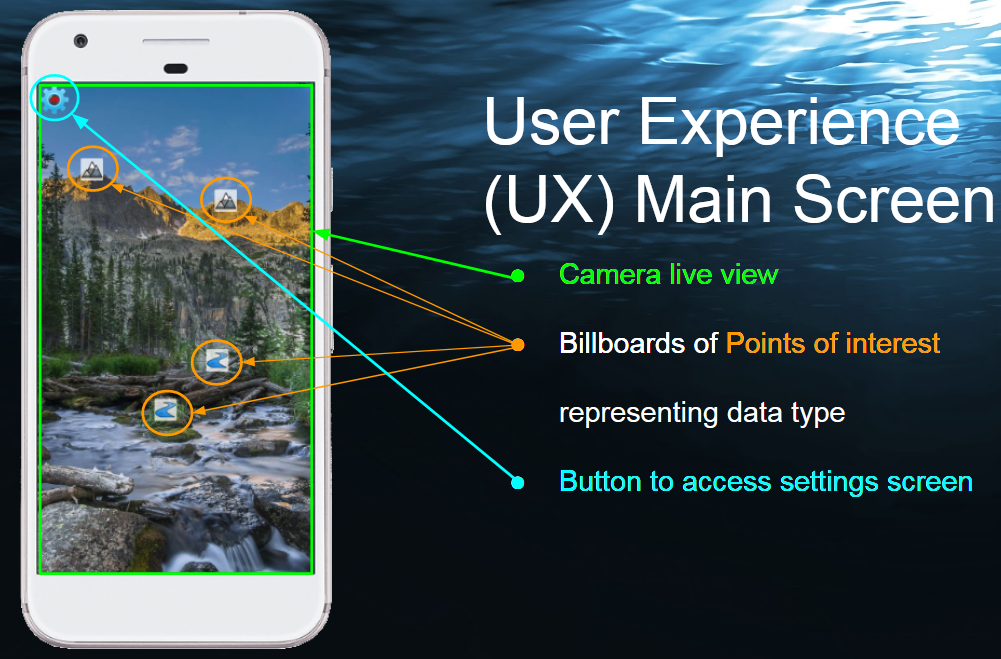
The main objective of the user interface (UI) is to present the user with a sense of realism through augmented reality. This is portrayed by drawing interactive three-dimensional objects onto the device's camera. The objects, called billboards, represent points of interests (PoIs) such as mountains, rivers, reservoirs, wells, and soil moisture. The billboards are placed on the main segment of the PoIs. The user will be able to click on billboards to view descriptions of the PoIs.

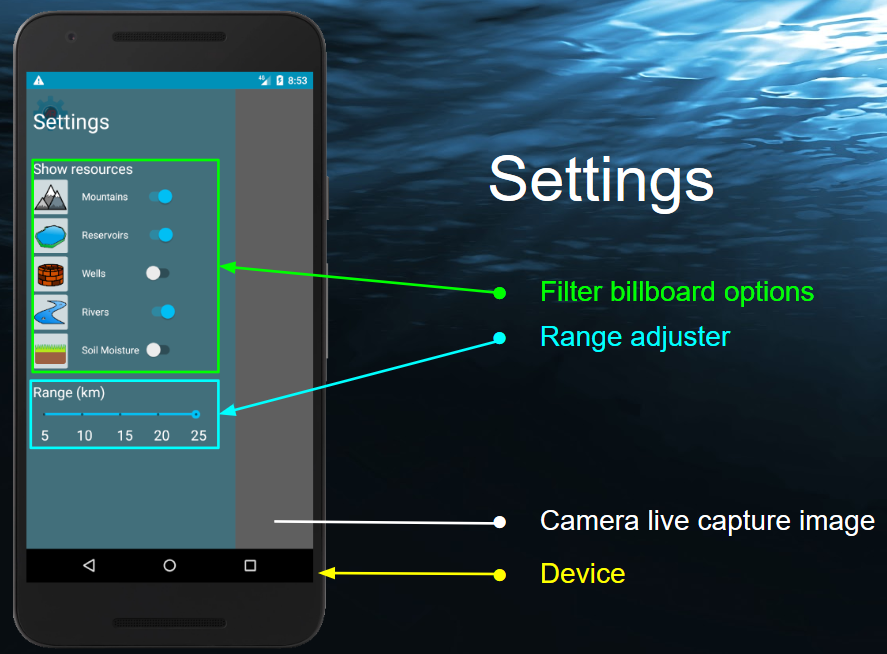
Upon launching the application, the user will be presented with the welcome screen, a scenic view of mountains and river. An icon, the AR Droplet, will fade into the middle of the scene giving the application time to set up its UI.

A gear on the top left corner of the screen will represent the page to the settings. When clicking the gear, the user will be prompted with the settings page. Here the user can toggle which PoIs will be visible and adjust the maximum distance the system will search the PoIs to display the billboards.

**9.2 Screen Frameworks or Images**

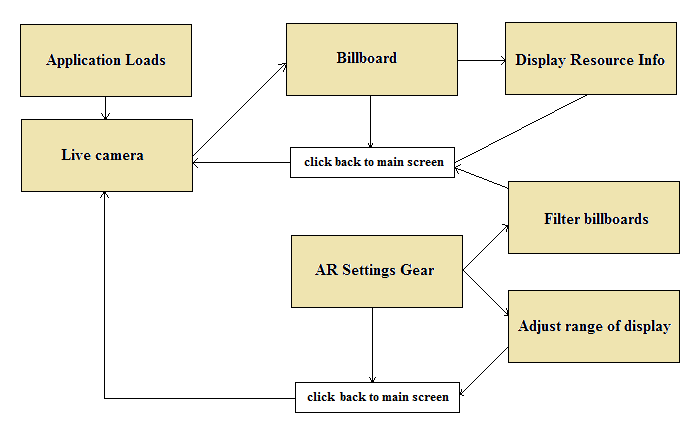






**9.3 User Interface Flow Model**

The application will load upon launch. The camera will display as the main screen, projecting any billboards onto the camera as AR elements. The user may click the billboard to display its resource information or click on the AR settings gear to prompt the settings menu to filter which billboards to display or adjust the range which billboards will appear.



**10. Requirements Validation and Verification**

|  |  |  |
| --- | --- | --- |
| Requirement | Component/Module | Test |
| 4.1.1 Application Requirements | --- |  |
| 4.1.1.1 The application shall maintain a data structure that holds Points of Interest (POIs) as follows: |  |  |
| 4.1.1.1.1 The data structure shall be loaded with POIs from Watertrek upon application startup, and periodically updated as needed to meet the additional POI data structure requirements. | MainActivity class in UI Module | Observation of Code and Manual Testing |
| 4.1.1.1.2 At a minimum, the data structure shall contain all Watertrek POIs that are within the load-radius of the device’s current location. | MainActivity class in UI Module | Observation of Code and Manual Testing |
| 4.1.1.2 The application shall provide a screen, called the “Billboard Screen” that meets the following sub-requirements: | MainActivity class in UI Module (uses MainActivity as class name instead of Billboard Screen | Observation of Code |
| 4.1.1.2.1 The Billboard Screen shall display the device’s front facing camera output as its background. | MainActivity class in UI Module | Manual Testing |
| 4.1.1.2.2 If the device does not have a front facing camera, or if permission for the camera is not granted, the Billboard Screen shall display a white background instead of the camera output. | MainActivity class in UI Module | Observation of Code and Manual Testing |
| 4.1.1.2.3 On top of the camera output (or white background if camera is unavailable), the Billboard Screen shall render billboard objects on top of the camera output to represent currently active POIs that are within the view of the camera. | MainActivity class in UI Module | Observation of Code and Manual Testing |
| 4.1.1.2.4 A POI shall be considered active if it is loaded in the POI data structure, is located within the user specified view-radius of the device’s current location, and user controlled filters do not prevent that type of POI from being displayed. | MainActivity class in UI Module | Observation of Code and Manual Testing |
| 4.1.1.2.5 The POI billboards on the billboard screen shall be rendered in the appropriate position on the screen, such that the pixels of each billboard appear to be located where the physical POI would be located in the camera picture. | MainActivity class in UI Module | Observation of Code and Manual Testing |
| 4.1.1.3 The application shall provide a method by which the user can set the view-radius. | MainActivity class in UI Module | Observation of Code and Manual Testing |
| 4.1.1.4 The application shall provide a filter capability, by which the user can determine which types of POIs will be displayed on the Billboard Screen. | MainActivity class in UI Module | Observation of Code and Manual Testing |
| 4.1.1.5 The application shall provide a method by which the user can select a billboard which is displayed on the Billboard Screen. | MainActivity class in UI Module | Observation of Code and Manual Testing |
| 4.1.1.6 Upon selecting a POI’s billboard on the Billboard Screen, the Application shall display a details screen, which will display the detailed information about the POI downloaded from Watertrek. | MainActivity class in UI Module | Observation of Code and Manual Testing |
| 4.1.2 Framework Requirements | --- |  |
| 4.1.2.1 The Framework shall provide methods for accessing the following location and orientation information for the device: | ARGLSensor and ARGLGps classes in the Sensor Module | Observation of Code and Manual Testing |
| 4.1.2.1.1 Latitude, Longitude, Altitude of the device | ARGLGps class in the Sensor Module | Observation of Code and Manual Testing |
| 4.1.2.1.2 Gravity vector which represents a vector from the device to the center of the earth in device coordinate system. | ARGLSensor class in the Sensor Module | Observation of Code and Manual Testing |
| 4.1.2.1.3 Pitch, Yaw, and Roll angles for the device | VectorMath class | Observation of Code and Manual Testing |
| 4.1.2.1.4 Compass Bearing and Elevation Angle for the device. | ARGLSensor class and VectorMath class | Observation of Code and Manual Testing |
| 4.1.2.1.5 Front vector, Up Vector, Right vector for device in world coordinate system. | VectorMath class | Observation of Code and Manual Testing |
| 4.1.2.2 The framework shall the following 3D Rendering abstractions: |  | Observation of Code and Manual Testing |
| 4.1.2.2.1 The framework shall provide a drawing surface class, on which camera preview, 3D objects and 2D objects can be rendered | ARFragment, ARActivity, ARView | Observation of Code and Manual Testing |
| 4.1.2.2.2 The framework shall provide a camera abstraction. | ARGLCamera | Observation of Code and Manual Testing |
| 4.1.2.2.2.1 The camera shall allow setting the location, angle, and orientation vectors. | ARGLCamera | Observation of Code and Manual Testing |
| 4.1.2.2.2.2 The camera shall provide a view matrix for 3D rendering. | ARGLCamera | Observation of Code and Manual Testing |
| 4.1.2.2.3 The framework shall provide an abstraction for a 3D drawable entity. | ARGLEntity | Observation of Code and Manual Testing |
| 4.1.2.2.3.1 The entity abstraction shall hold information about location, orientation and scale of the entity. | ARGLEntity | Observation of Code and Manual Testing |
| 4.1.2.2.3.2 The entity abstraction shall hold the following information needed for rendering: vertices, normal vectors, shader, texture and texture coordinates. | ARGLDrawable and subclasses | Observation of Code and Manual Testing |
| 4.1.2.2.3.3. The entity abstraction shall provide a drawing function | ARGLDrawable and subclasses | Observation of Code and Manual Testing |
| 4.1.2.2.4 The framework shall provide an abstraction for displaying Billboards, which represent a 3D sign that always faces the camera. | ARGLBillboard and ARGLCircleScene | Observation of Code and Manual Testing |
| 4.1.2.2.4.1 The framework shall provide a method that allows setting texture, position and scale of each Billboard. | ARGLBillboard | Observation of Code and Manual Testing |
| 4.1.2.2.5 The framework shall provide a method for tracing from screen pixel to an entity or Billboard in the 3D world allowing the user to select the object by touch/click | Not implemented.Billboard clicking was implemented in the app, but not provided in a general way in the framework. |  |
| 4.1.2.2.3 Math | --- |  |
| 4.1.2.3.1 The framework shall provide functions for the following vector operations: | --- |  |
| 4.1.2.3.1.1 Add | VectorMath | Observation of Code and Manual Testing |
| 4.1.2.3.1.2 Subtract | VectorMath | Observation of Code and Manual Testing |
| 4.1.2.3.1.3 Cross-Product | VectorMath | Observation of Code and Manual Testing |
| 4.1.2.3.1.4 Dot-Product | VectorMath | Observation of Code and Manual Testing |
| 4.1.2.3.1.5 Calculate Device Pitch/Yaw/Roll | VectorMath | Observation of Code and Manual Testing |
| 4.1.2.3.1.6 Calculate device up vector/front vector/right vector | VectorMath | Observation of Code and Manual Testing |
| 4.1.2.3.1.7 Calculate compass bearing and elevation angle | VectorMath | Observation of Code and Manual Testing |
| 4.1.2.3.1.8 Convert vector from world coordinates to device coordinate (and vice versa) | VectorMath | Observation of Code and Manual Testing |
| 4.1.2.3.2 The framework shall provide functions for the following geographical math operations: | --- |  |
| 4.1.2.3.2.1 Convert between lat/lon/altitude and an absolute x/y/z; | GeoMath | Observation of Code and Manual Testing |
| 4.1.2.3.2.2 Given two coordinates calculate compass bearing and elevation angle from the first coordinate to the second. | VectorMath | Observation of Code and Manual Testing |
| 4.1.2.4 Point of Interest (POI) management | --- |  |
| 4.1.2.4.1 The framework shall provide an abstraction of Points of Interest (POIs) that includes at minimum a type identifier, geographic location (latitude, longitude, altitude), and name. | ARLandmark, Reservoir, Well, Snotel, Soil Moisture, River | Observation of Code and Manual Testing |
| 4.1.2.4.2 The framework shall provide an abstraction of a POI data structure that allows adding, removing, filtering, saving, loading, iterating through POIs efficiently | ARLandmarkTable | Observation of Code and Manual Testing |
| 4.1.2.5 Watertrek Hydrology access | NetworkTask and Service classes | Observation of Code and Manual Testing |
| 4.1.2.5.1 The framework shall provide a method to securely query Water Trek with various parameters for points of interest. | NetworkTask and Service classes | Observation of Code and Manual Testing |
| 4.1.2.5.2 The framework shall provide methods for storing and locally querying data. | Service Classes and local database | Observation of Code and Manual Testing |
| 4.1.2.5.3 Provide method of programmatically detecting or being notified of connection error, data compatibility error, or other type of error associated with Water Trek access. | NetworkTask class | Observation of Code and Manual Testing |

**11. Glossary**

* Android — Mobile Operating System (Google)
* Android Studio —Android Studio is the official Integrated Development Environment (IDE) for Android app development.
* AR — Augmented Reality, is a direct or indirect live view of a physical, real-world environment whose elements are "augmented" by computer-generated perceptual information, ideally across multiple sensory modalities
* ARA — Augmented Reality Application
* IDE — Integrated Development Environment
* Integrated Development Environment —software application that provides tools for software development
* JPL — Jet Propulsion Laboratory
* P.O.I — Point of Interest
* REST — Representational State Transfer

**12. References**

12.0 Android Studio Reference

<https://developer.android.com/reference/java/lang/ref/Reference>

12.1 WaterTrek REST API Reference

<https://watertrek.jpl.nasa.gov/hydrology/restapi.html#/>

12.2 OpenGL

<https://www.khronos.org/registry/OpenGL/specs/es/2.0/es_full_spec_2.0.pdf>

<https://www.khronos.org/registry/OpenGL/specs/es/2.0/GLSL_ES_Specification_1.00.pdf>

12.3 SQLite document list

<https://www.sqlite.org/doclist.html>