**SOFTWARE ENGINEERING**

**CAPSTONE PROJECT**

**PROJECT TITLE: AR FLOOD SIMULATOR**

**Instructor: Dr. Michael Findler**

**Database team:**

**Kavya Kammaripalle**

**Aditya Thummala**

**Nikhil Mayakuntla**

Table of Contents

[1. Project Description 3](#_Toc4532588)

[1.1 Purpose: 3](#_Toc4532589)

[1.2 Scope: 3](#_Toc4532590)

[2. Software Specifications: 3](#_Toc4532591)

[3. Document Standards: 4](#_Toc4532592)

[4. Architecture 5](#_Toc4532593)

[5. Use cases 6](#_Toc4532594)

[5.1 Home-Buyer 7](#_Toc4532595)

[5.2 Builder 9](#_Toc4532596)

[5.3 National Weather Service Officer 11](#_Toc4532597)

[6. Class Diagram 13](#_Toc4532598)

[7. Firebase 14](#_Toc4532599)

[7.1. Introduction: 14](#_Toc4532600)

[7.2. Reasons for choosing firebase: 14](#_Toc4532601)

[8. Getting Started with Firebase: 15](#_Toc4532602)

[8.1. Signing into the Firebase Console: 15](#_Toc4532603)

[8.2 Creating a New Project: 16](#_Toc4532604)

[9. Integrating Firebase Realtime database through Unity 18](#_Toc4532605)

[9.1 Database Structure 18](#_Toc4532606)

[9.2 Saving Data 18](#_Toc4532607)

[10. Writing data into Firebase 19](#_Toc4532608)

[10.1 Building 19](#_Toc4532609)

[10.2 Location 19](#_Toc4532610)

[10.3 Project.cs 22](#_Toc4532611)

[11. Code 24](#_Toc4532612)

[11.1 FirebaseManager.cs 24](#_Toc4532613)

[11.2 Building.cs 26](#_Toc4532614)

[11.3 Location.cs 28](#_Toc4532615)

[11.4 LocationDL.cs 29](#_Toc4532616)

[11.5 Project.cs 32](#_Toc4532617)

[11.6 ProjectDL.cs 33](#_Toc4532618)

[11.7 TestScript.cs 35](#_Toc4532619)

[12. Sprint Schedule 38](#_Toc4532620)

[13. References 38](#_Toc4532621)

# 1. Project Description

Flooding in local areas has become increasingly common raising awareness among society of potential river levels in local areas is important. The primary goal of the proposed work is to uncover issues and risk of nearby flooding by giving an on location Augmented Reality (AR) impression with regards to the seriousness of a given flood warning or a user indicated flood level. As a result, the public will be quicker to respond in times of flooding thus potentially diminishing damage and saving lives.

Augmented reality flood simulator using HoloLens is a windows desktop application in which the user (builder/home-buyer/national weather service officer) will choose a location and predict if it is a flood prone area by choosing user-specified inches of rain per hour and predicting the flood level accordingly in order to prevent any constructions in that particular flood zone area. The user can also use this application in an already flooded area to provide safety measures ahead of time by utilizing the old information to avoid any future calamities reducing damage and saving lives.

## 1.1 Purpose:

AR flood simulator purpose is to stream the affected flood regions using HoloLens and can predict the potential flood zone areas utilizing map box. It also provides awareness among the people about the impact and level of flooding at specific areas.

## 1.2 Scope:

AR Flood simulatorhelps to gain a sense of risk within environment. It helps to provide proper locations for well planning and designing of the city. By using the old data, we can provide safety measures in advance. We can even plan to avoid constructions in the flood prone areas.

# 2. Software Specifications:

1. Unity 2017.4.19f1

2. Visual Studio 2015 enterprise edition

3. Visual Studio 2017 enterprise edition

4. Map box 2.0 SDK

5. Firebase

# 3. Document Standards:

Document standards should apply to all documents produced in our project. Documents should have a consistent style and appearance, and documents of the same type should have a consistent structure.

Word processing standards

The size of the paper used is A4(8.3 x 11.7 inches).

Base font used is times new roman.

All the main titles font size is 16, for the sub titles its 14 and the whole content its 12.

Line spacing is 1.15

Each section must start on a new page.

Bold text is used in Main titles and table column headers.

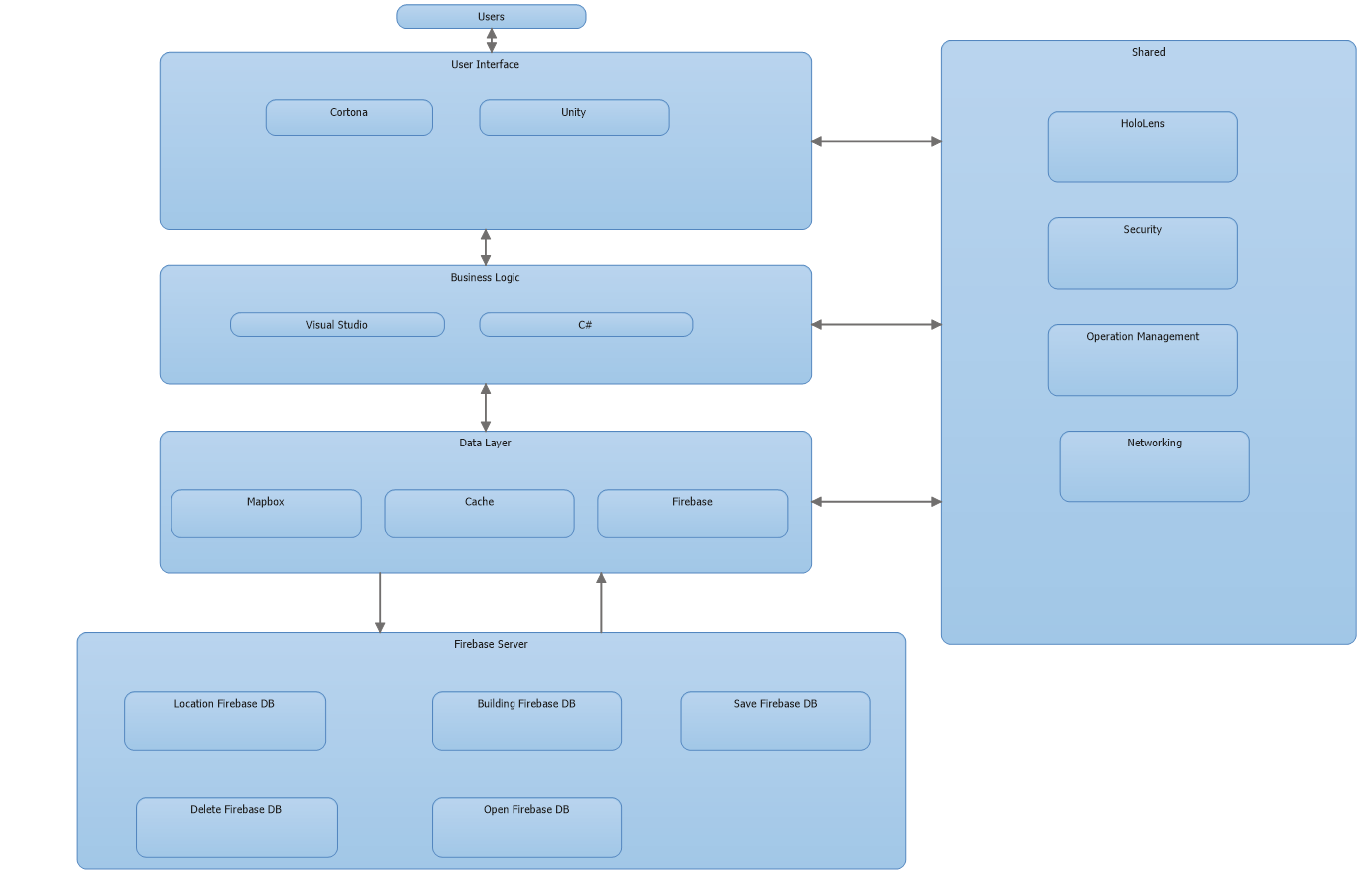
A standard header “UNIVERSITY OF HOUSTON - CLEAR LAKE SWEN 6837 CAPSTONE SPRING 2019 AR FLOOD SIMULATOR” appears on all pages.

A page footer only consists of page numbers according to the section.

Whole content is justified so that the document looks more polished.

In all the UML diagrams we added tables for revision history, group members names and final version.

# 4. Architecture

****

# 5. Use cases

|  |  |
| --- | --- |
| **GROUP MEMBERS** | **Kavya Kammaripalle**  **Aditya Thummala**  **Nikhil Mayakuntla** |
| **FINAL VERSION** | **1.1** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision History** | | | |
| **Name** | **Date** | **Reason for Changes** | **Version** |
| **Kavya Kammaripalle**  **Aditya Thummala**  **Nikhil Mayakuntla** | **06/02/2019** | **Created** | **1.0** |
| **Kavya Kammaripalle**  **Aditya Thummala**  **Nikhil Mayakuntla** | **08/02/2019** | **Modified use-cases** | **1.1** |

## 

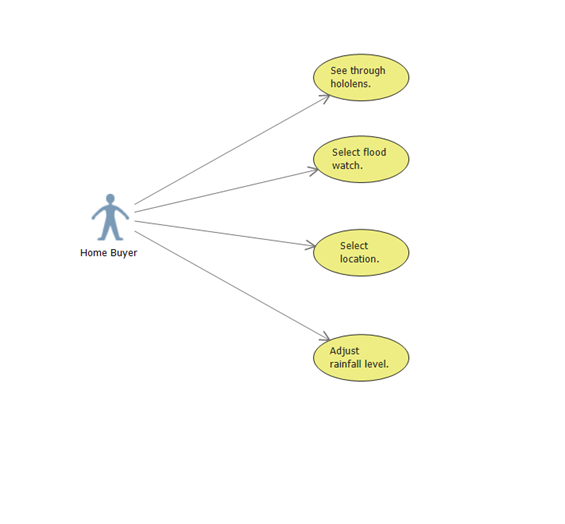
## 

## 

## 

## 

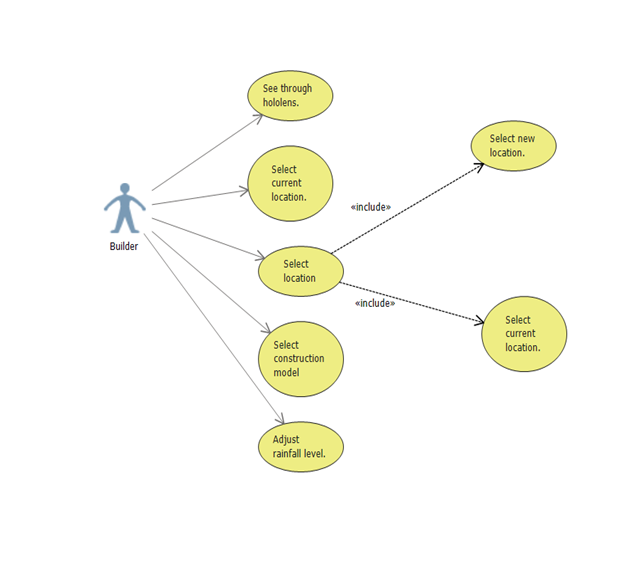
## 5.1 Home-Buyer



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case Template** | | | | |
| **Name:** | **Home Buyer** | **ID:** |  |  |
| **Created By:** |  | **Date Created:** |  |  |
| **Primary Actor:** | **Home Buyer** | **Secondary Actors:** | **None** |  |
| **Description:** | **Home Buyer is an actor who buys the home based on the flood level in that certain area.** | | |  |
| **Trigger:** | **This Triggers whenever a Home Buyer selects the Model.** | | |  |
| **Preconditions:** | **Must be an authorized home buyer.** | | |  |
| **Postconditions:** |  | | |  |
| **Normal Flows:** | **Selects the Flood Watch.**  **Home Buyer selects the location.**  **Adjusts the rainfall level.** | | |  |
| **Alternative Flows:** | **No Alternative Flow** | | |  |
| **Exceptions:** | **No Exceptions** | | |  |
| **Priority:** | **High** | | |  |
| **Frequency of Use:** | **High** | | |  |
| **Business Rules:** | **He must be a valid user** | | |  |
| **Other Information:** | **None** | | |  |
| **Assumptions:** | **None** | | |  |

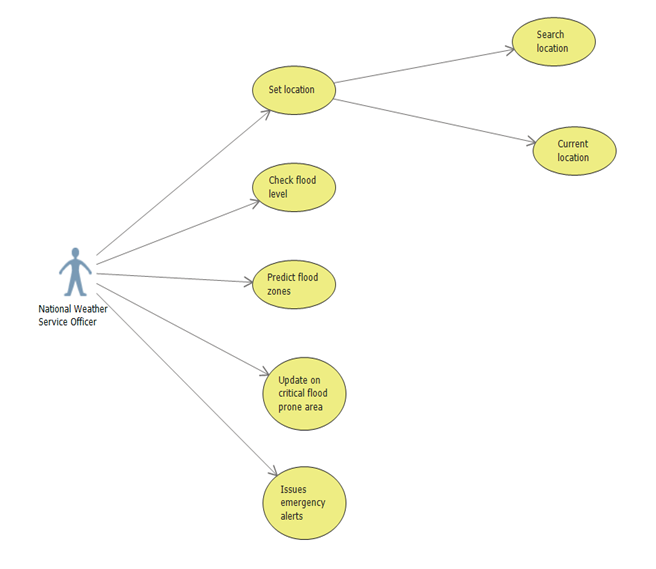
## 

## 5.2 Builder



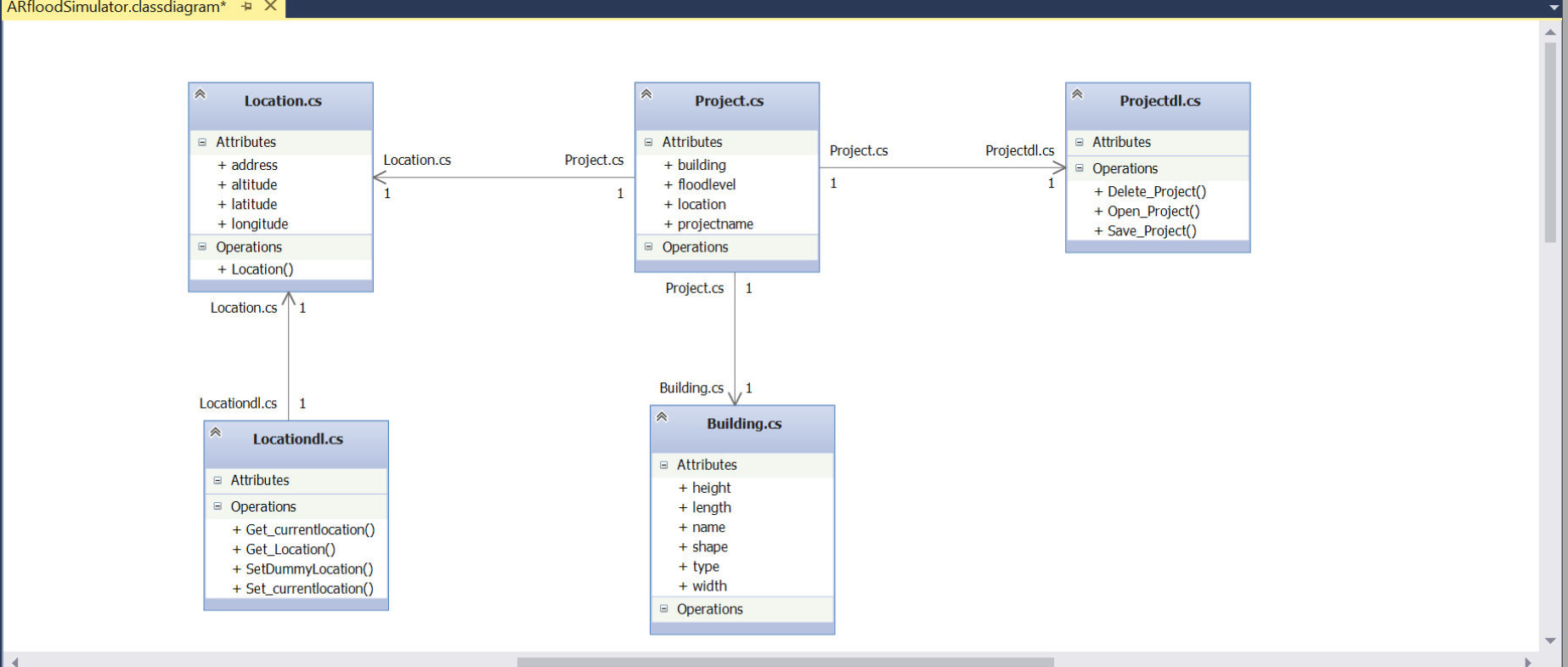
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case Template** | | | | |
| **Name:** | **Builder** | **ID:** |  |  |
| **Created By:** |  | **Date Created:** |  |  |
| **Primary Actor:** | **Builder** | **Secondary Actors:** | **None** |  |
| **Description:** | **Builder is an actor where he can check the material required to build in the construction site to cause no damages to the buildings when there are heavy floods** | | |  |
| **Trigger:** | **This triggers when the builder selects the location and adjusts the rainfall.** | | |  |
| **Preconditions:** | **Builder must have HoloLens.** | | |  |
| **Postconditions:** |  | | |  |
| **Normal Flows:** | **Home Buyers sees through the HoloLens.**  **Selects the Flood Watch.**  **Home Buyer selects the location.**  **Adjusts the rainfall level.** | | |  |
| **Alternative Flows:** | **No Alternative Flows.** | | |  |
| **Exceptions:** | **No Exceptions** | | |  |
| **Priority:** | **High** | | |  |
| **Frequency of Use:** | **High** | | |  |
| **Business Rules:** | **He must be a valid user** | | |  |
| **Other Information:** | **None** | | |  |
| **Assumptions:** | **None** | | |  |

## 5.3 National Weather Service Officer



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case Template** | | | | |
| **Name:** | **National Weather Service Officer.** | **ID:** |  |  |
| **Created By:** |  | **Date Created:** |  |  |
| **Primary Actor:** | **National Weather Service Officer.** | **Secondary Actors:** | **None** |  |
| **Description:** | **National Weather Service Officer is an actor who can predict the flood zone areas.** | | |  |
| **Trigger:** | **This triggers when the actor selects specific location.** | | |  |
| **Preconditions:** | **Builder must have HoloLens.** | | |  |
| **Postconditions:** |  | | |  |
| **Normal Flows:** | **National Weather sees through the HoloLens.**  **Selects the Flood Watch.**  **Predicts the flood prone areas.** | | |  |
| **Alternative Flows:** | **No Alternative Flows.** | | |  |
| **Exceptions:** | **No Exceptions** | | |  |
| **Priority:** | **High** | | |  |
| **Frequency of Use:** | **High** | | |  |
| **Business Rules:** | **He must be a valid user.** | | |  |
| **Other Information:** | **None** | | |  |
| **Assumptions:** | **None** | | |  |

# 6. Class Diagram



# 

# 

# 7. Firebase

## 7.1. Introduction:

Firebase was acquired by Google in 2014 and number of features were introduced later. Firebase is a backend platform for building Web, Android and IOS applications. It offers real time database, different APIs, multiple authentication types and hosting platform. It is scalable, real-time backup for the application and is platform independent.

Firebase is for anybody who wants to write applications without having to run backend servers or write server-side code.

## 7.2. Reasons for choosing firebase:

**Authentication:** Firebase authentication provides a way to add user account creation and sign in capabilities. It supports social login providers Facebook, GitHub, Twitter and Google, but we do not need authentication as there is no user login in our desktop application.

**Real-time Database:** Firebase provides a real-time database and backend as a service. The service provides application developers an API that allows application data to be synchronized across clients and stored on Firebase's cloud.The company provides client libraries that enable integration with Android, iOS, JavaScript, Java, Objective-C, Swift and Node.js applications. The database is also accessible through a REST API and bindings for several JavaScript Frameworks. The REST API uses the Server-Sent Events protocol, which is an API for creating HTTP connections for receiving push notifications from a server. Developers using the real-time database can secure their data by using the company's server-side-enforced security rules. Cloud Fire store which is Firebase's next generation of the Real-time Database was released for beta use.

**Hosting:** Firebase hosting is a static and dynamic web hosting service. The applications can be deployed over secured connection to Firebase servers.

**Storage:** Firebase Storage provides secure file uploads and downloads for Firebase apps, regardless of network quality. The developer can use it to store images, audio, video, or other user-generated content. Google Cloud Storage backs firebase Storage.

# 

# 

# 

# 

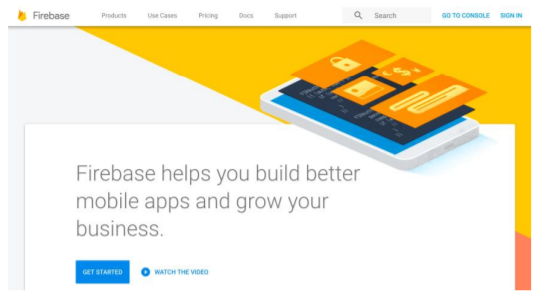
# 

# 8. Getting Started with Firebase:

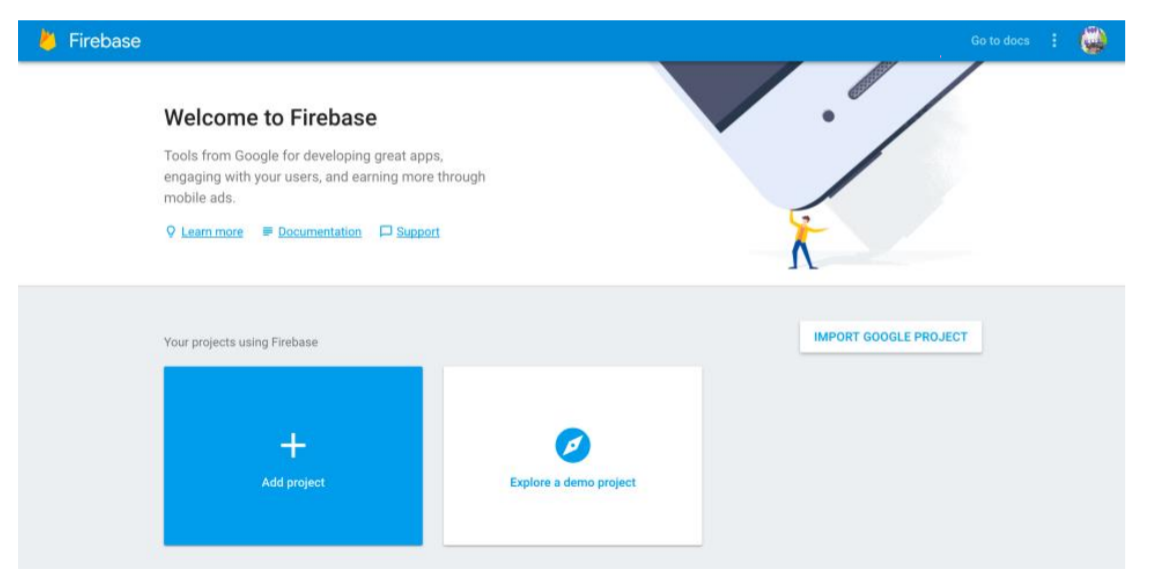
Working with Firebase requires a Google account and at least one Firebase project.

## 8.1. Signing into the Firebase Console:

The first step in working with Firebase involves signing in to the Firebase console. Begin by opening a browser window and navigating to <https://firebase.google.com>.

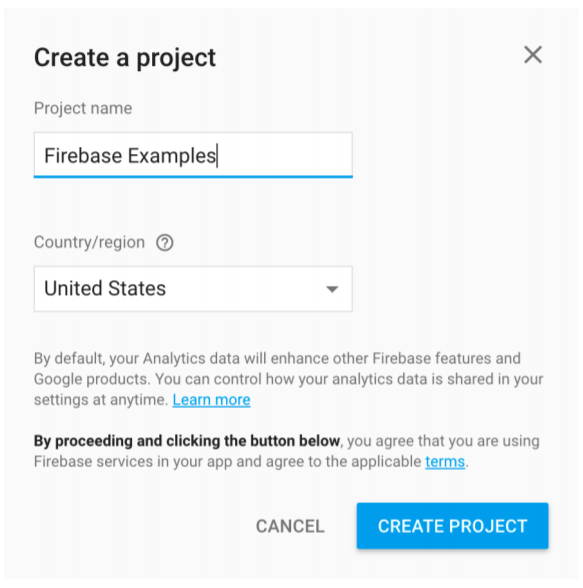


Click on either the Sign In link in the top right-hand corner, or the Get Started button and enter the credentials for your Google account. The following screen appears.

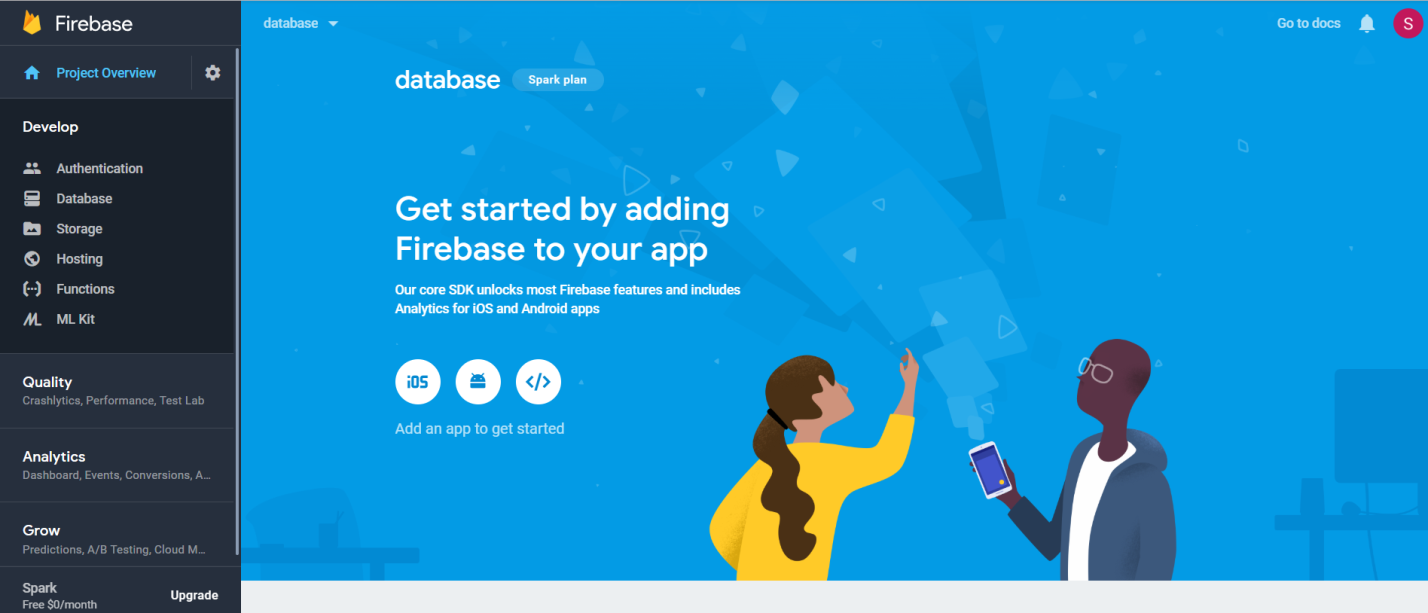


## 8.2 Creating a New Project:

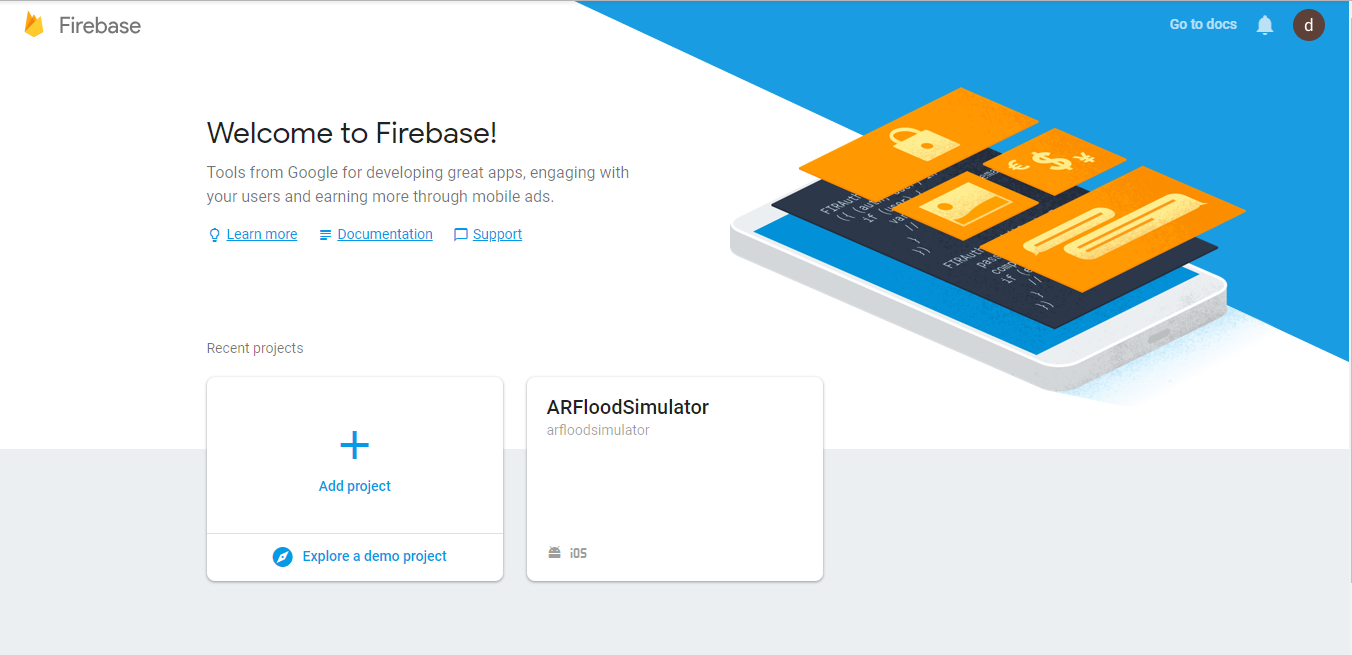
To create a new project, simply click on the Add project in the console, enter Project name field and select your country from the drop-down list:



After completing the form, click on the Create Project button. The project is created after which the main Firebase console screen will appear as illustrated in Figure:



Create a project called AR Flood Simulator in firebase console as illustrated in the figure:



# 9. Integrating Firebase Realtime database through Unity

Once a project is created in firebase, google cloud store gives project information like project number, firebase URL, project ID, package name, api key.

**Firebase URL:** https://console.firebase.google.com/u/1/project/arfloodsimulator/overview

**Project ID:** arfloodsimulator

**Package Name:** com.ar.flood.simulator

**API Key:** AIzaSyBN7I1WlvatA4ddxfwVv3IYvefXfEZDrf0

**Project Number:** 108653049257

## 9.1 Database Structure

* Database is structured in JSON tree.
* Firebase Realtime Database data is stored as JSON objects.
* Unlike a SQL database, there are no tables or records in firebase. When you add data to the JSON tree, it becomes a node in the existing JSON structure with an associated key.

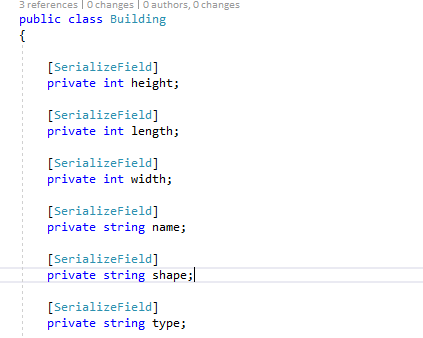
## 9.2 Saving Data

* The basic database write operation is a set which saves new data to the specified database reference.
* The data for the app is stored at this database reference.
* create a database reference to your user data. Then use set() / set Value() to save a user object to the database.
* When a JSON object is saved to the database, the object properties are automatically mapped to database child locations in a nested fashion.
* Now if you navigate to the URL, you will see the value. You can also save data directly to a child location.

# 10. Writing data into Firebase

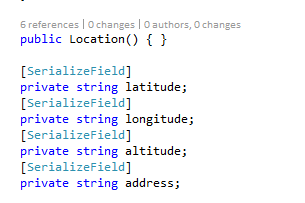
## 10.1 Building

* Add a script named Building.cs that has serialized fields. Serialization is done to store the data in firebase, there is a method called JSON utility.
* It only creates JSON when private fields have serialized attributes.



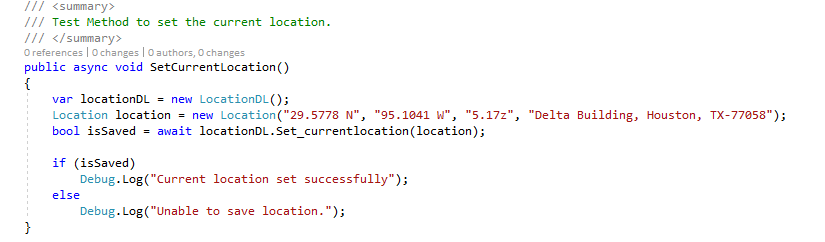
## 10.2 Location

* Add a script named Location.cs that has serialized fields. Serialization is done to store the data in firebase, there is a method called JSON utility.
* It only creates JSON when private fields have serialized attributes.



**SetCurrentLocation:**

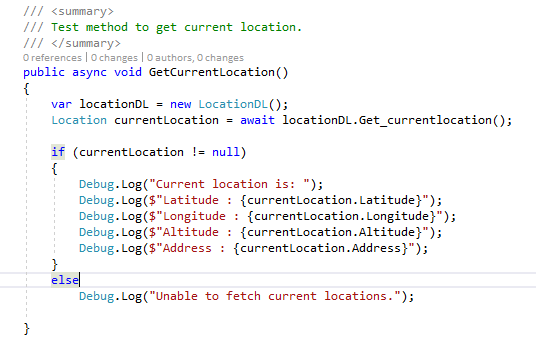
* TestScript.SetCurrentLocation() is called on click of the Set Current Location button.



* Add a script named LocationDL.cs. locationdl will create object of class LocationDL.
* In this we create location object and save the location record to the firebase database. To generate a JSON representation of the public fields of an object static method Json is called. The data is stored at the database reference.
* When a JSON object is saved to the database, the object properties are automatically mapped to database child locations in a nested fashion.
* async void SetCurrentLocation() is called to save location records to the database.

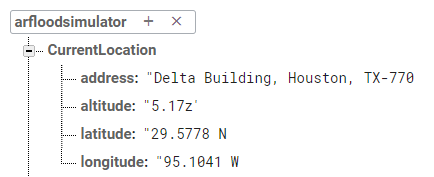
**GetCurrentLocation:**

* TestScript.GetCurrentLocation() is called on click of the Get Current Location button.



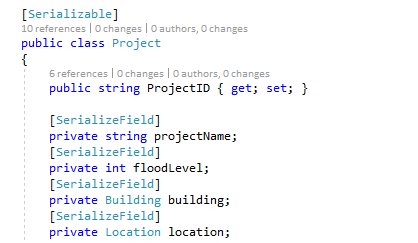
* async void GetCurrentLocation() is called to save location records to the database

The Tree-like structure of the Firebase database can clearly be seen in this Firebase Console representation of our project’s database.

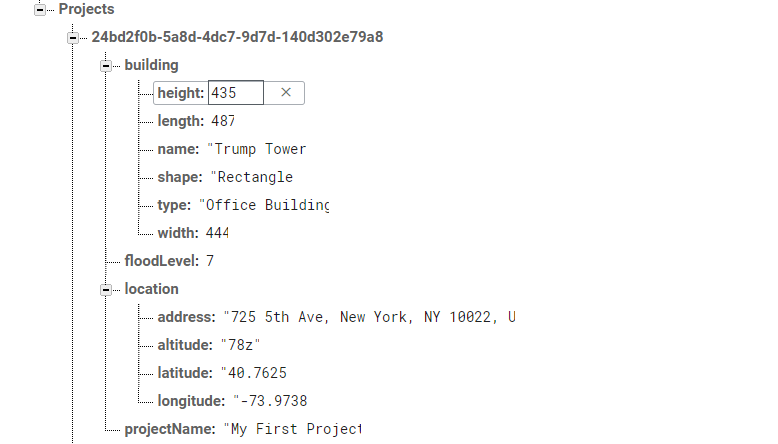


## 10.3 Project.cs

* Add a script named project.cs that has serialized fields. Serialization is done to store the data in firebase, there is a method called JSON utility.
* It only creates JSON when private fields have serialized attributes.

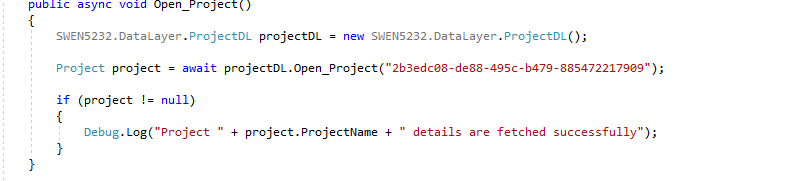


* async void Save Project()is called to save project records to the database.



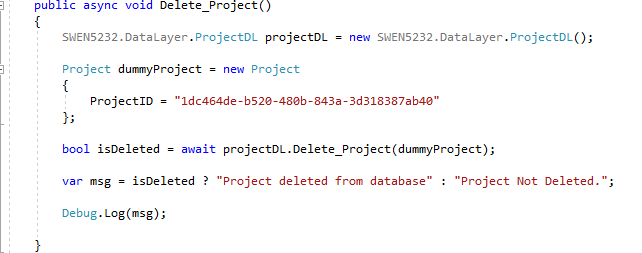
**Open Project:**

* async void Open\_Project()is called to open project records from the database.



**Delete Project:**

* async void Delete\_Project()is called to delete project records from the database.



# 11. Code

|  |  |  |  |
| --- | --- | --- | --- |
| Revision History | | | |
| Name | Date | Reason for Changes | Version |
| Kavya Kammaripalle | 02/08/2019 |  | 1.0 |
| Aditya Thummala | 02/27/2019 |  | 1.1 |
| Nikhil Mayakuntla | 03/15/2019 |  | 1.2 |

## 11.1 FirebaseManager.cs

using UnityEngine;

using Firebase.Database;

using System;

using System.Collections;

using UnityEngine.Networking;

public static class FirebaseManager

{

//private static readonly string \_databaseURL = "";

public static DatabaseReference dataRef;

static FirebaseManager()

{

InitialiseFirebase();

}

private static void InitialiseFirebase()

{

try

{

if (dataRef == null)

{

try

{

dataRef = FirebaseDatabase.DefaultInstance.RootReference;

Debug.Log("Firebase database initialization completed.");

}

catch (Exception)

{

Debug.LogError("Unable to connect to firebase. Check internet connection.");

}

}

}

catch (Exception ex)

{

if (ex.Message != null)

{

Debug.LogError(ex.Message);

}

if (ex.InnerException != null && ex.InnerException.Message != null)

{

Debug.LogError(ex.InnerException.Message);

}

}

}

}

## 11.2 Building.cs

using System;

using UnityEngine;

namespace SWEN5232.Models

{

[Serializable]

public class Building

{

[SerializeField]

private int height;

[SerializeField]

private int length;

[SerializeField]

private int width;

[SerializeField]

private string name;

[SerializeField]

private string shape;

[SerializeField]

private string type;

public string Type

{

get { return type; }

set { type = value; }

}

public string Shape

{

get { return shape; }

set { shape = value; }

}

public string Name

{

get { return name; }

set { name = value; }

}

public int Width

{

get { return width; }

set { width = value; }

}

public int Height

{

get { return height; }

set { height = value; }

}

public int Length

{

get { return length; }

set { length = value; }

}

}

}

## 11.3 Location.cs

using System;

using UnityEngine;

namespace SWEN5232.Models

{

[Serializable]

public class Location

{

public Location(string lat, string longi, string alti)

{

latitude = lat;

longitude = longi;

altitude = alti;

}

public Location(string lat, string longi, string alti, string address)

{

latitude = lat;

longitude = longi;

altitude = alti;

this.address = address;

}

public Location() { }

[SerializeField]

private string latitude;

[SerializeField]

private string longitude;

[SerializeField]

private string altitude;

[SerializeField]

private string address;

public string Address

{

get { return address; }

set { address = value; }

}

public string Altitude

{

get { return altitude; }

set { altitude = value; }

}

public string Longitude

{

get { return longitude; }

set { longitude = value; }

}

public string Latitude

{

get { return latitude; }

set { latitude = value; }

}

}

}

## 11.4 LocationDL.cs

using Firebase.Database;

using SWEN5232.Models;

using System;

using System.Collections;

using System.Collections.Generic;

using System.Threading.Tasks;

using UnityEngine;

namespace SWEN5232.DataLayer

{

/// <summary>

/// Class to save/fetch data related to location.

/// </summary>

public class LocationDL

{

/// <summary>

/// Method to save the current location to firebase.

/// </summary>

/// <param name="locationData"></param>

/// <returns></returns>

public async Task<bool> Set\_currentlocation(Location locationData)

{

var json = JsonUtility.ToJson(locationData);

var isSaved = false;

await FirebaseManager.dataRef.Child("CurrentLocation").SetRawJsonValueAsync(json).ContinueWith(T =>

{

if (T.IsCompleted && !T.IsFaulted)

{

isSaved = true;

}

});

return isSaved;

}

/// <summary>

/// Method to get the current location from firebase which was set by Set\_currentlocation method.

/// </summary>

/// <returns></returns>

public async Task<Location> Get\_currentlocation()

{

Location locationToFetch = null;

try

{

DataSnapshot snapshot = await FirebaseManager.dataRef.Child("CurrentLocation").GetValueAsync();

if (snapshot != null && snapshot.Value != null)

{

locationToFetch = JsonUtility.FromJson<Location>(snapshot.GetRawJsonValue());

}

}

catch (Exception)

{

locationToFetch = null;

Debug.LogError("Unable to fetch location data. Check internet connection.");

}

return locationToFetch;

}

/// <summary>

/// Dummy method to save list of pre defined locations in firebase.

/// </summary>

public void SetDummyLocations()

{

List<Location> locations = new List<Location>

{

new Location

{

Address = "Rixos The Palm Dubai",

Latitude = "25.1212",

Longitude = "55.1535",

Altitude = "XX"

},

new Location

{

Address = "Shangri-La Hotel",

Latitude = "25.2084",

Longitude = "55.2719",

Altitude = "XX"

},

new Location

{

Address = "Grand Hyatt",

Latitude = "25.2285",

Longitude = "55.3273",

Altitude = "XX"

},

new Location

{

Address = "151 East 77th Street, Manhattan, NY",

Latitude = "-73.9596265",

Longitude = "40.7736566",

Altitude = "XX"

}

};

foreach (var item in locations)

{

var json = JsonUtility.ToJson(item);

var uniqueID = Guid.NewGuid().ToString();

FirebaseManager.dataRef.Child("LocationList").Child(uniqueID).SetRawJsonValueAsync(json);

}

}

/// <summary>

/// Method to fetch the location and coordinates with address string.

/// </summary>

/// <param name="address"></param>

/// <returns></returns>

public async Task<Location> Get\_Location(string address)

{

if (address == null)

{

throw new ArgumentNullException();

}

Location foundLocation = null;

DataSnapshot snapshot = await FirebaseManager.dataRef.Child("LocationList").GetValueAsync();

if (snapshot != null)

{

var dict = snapshot.Value as Dictionary<string, object>;

foreach (var obj in dict)

{

var internalDict = obj.Value as Dictionary<string, object>;

string addrss = internalDict["address"].ToString();

if (addrss.ToUpperInvariant().Contains(address.ToUpperInvariant()))

{

foundLocation = new Location

{

Address = internalDict["address"].ToString(),

Latitude = internalDict["latitude"].ToString(),

Longitude = internalDict["longitude"].ToString(),

Altitude = internalDict["altitude"].ToString()

};

break;

}

}

}

return foundLocation;

}

}

}

## 11.5 Project.cs

using System;

using UnityEngine;

namespace SWEN5232.Models

{

[Serializable]

public class Project

{

public string ProjectID { get; set; }

[SerializeField]

private string projectName;

[SerializeField]

private int floodLevel;

[SerializeField]

private Building building;

[SerializeField]

private Location location;

public Location Location

{

get { return location; }

set { location = value; }

}

public Building Building

{

get { return building; }

set { building = value; }

}

public string ProjectName

{

get { return projectName; }

set { projectName = value; }

}

public int FloodLevel

{

get { return floodLevel; }

set { floodLevel = value; }

}

}

}

## 11.6 ProjectDL.cs

using SWEN5232.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using UnityEngine;

namespace SWEN5232.DataLayer

{

/// <summary>

/// Class to save/fetch data related to Projects.

/// </summary>

public class ProjectDL

{

/// <summary>

/// Method to save whole project into firebase.

/// </summary>

/// <param name="project">Project object</param>

/// <returns>True if saved successfully otherwise false.</returns>

public async Task<bool> Save\_Project(Project project)

{

if (project == null)

throw new ArgumentNullException("project");

bool isSaved = false;

project.ProjectID = Guid.NewGuid().ToString();

var json = JsonUtility.ToJson(project);

await FirebaseManager.dataRef.Child("Projects").Child(project.ProjectID).SetRawJsonValueAsync(json).ContinueWith(T =>

{

if (!T.IsFaulted && T.IsCompleted && T.Exception == null)

isSaved = true;

});

if (isSaved)

{

Debug.Log("Project details saved with id : " + project.ProjectID);

}

return isSaved;

}

/// <summary>

/// Method to delete project from firebase based on project ID.

/// </summary>

/// <param name="project">Project Obj with project ID.</param>

/// <returns>Returns true if project is deleted successfully otherwise false.</returns>

public async Task<bool> Delete\_Project(Project project)

{

if (project == null)

throw new ArgumentNullException("project");

if (string.IsNullOrEmpty(project.ProjectID))

throw new ArgumentNullException("ProjectID");

bool isDeleted = false;

await FirebaseManager.dataRef.Child("Projects").Child(project.ProjectID).RemoveValueAsync().ContinueWith(T =>

{

if (!T.IsFaulted && T.IsCompleted && T.Exception == null)

isDeleted = true;

});

return isDeleted;

}

/// <summary>

/// Method to get the project data by project id from firebase.

/// </summary>

/// <param name="projectId"></param>

/// <returns></returns>

public async Task<Project> Open\_Project(string projectId)

{

if (projectId == null) throw new ArgumentNullException("projectId");

Project projectData = null;

var snapshot = await FirebaseManager.dataRef.Child("Projects").Child(projectId).GetValueAsync();

if (snapshot.Value != null)

{

var projectDict = snapshot.Value as Dictionary<string, object>;

var json = snapshot.GetRawJsonValue();

projectData = JsonUtility.FromJson<Project>(json);

}

else

{

Debug.Log($"Project with id {projectId} not found.");

}

return projectData;

}

}

}

## 11.7 TestScript.cs

using System;

using System.Net;

using UnityEngine;

using SWEN5232.Models;

using SWEN5232.DataLayer;

/// <summary>

/// Dummy script to test the data layer.

/// </summary>

public class TestScript : MonoBehaviour

{

/// <summary>

/// Test Method to set the current location.

/// </summary>

public async void SetCurrentLocation()

{

var locationDL = new LocationDL();

Location location = new Location("29.5778 N", "95.1041 W", "5.17z", "Delta Building, Houston, TX-77058");

bool isSaved = await locationDL.Set\_currentlocation(location);

if (isSaved)

Debug.Log("Current location set successfully");

else

Debug.Log("Unable to save location.");

}

/// <summary>

/// Test method to get current location.

/// </summary>

public async void GetCurrentLocation()

{

var locationDL = new LocationDL();

Location currentLocation = await locationDL.Get\_currentlocation();

if (currentLocation != null)

{

Debug.Log("Current location is: ");

Debug.Log($"Latitude : {currentLocation.Latitude}");

Debug.Log($"Longitude : {currentLocation.Longitude}");

Debug.Log($"Altitude : {currentLocation.Altitude}");

Debug.Log($"Address : {currentLocation.Address}");

}

else

Debug.Log("Unable to fetch current locations.");

}

/// <summary>

/// Test method to get current location with address.

/// </summary>

public async void Get\_Location()

{

SWEN5232.DataLayer.LocationDL locationDL = new SWEN5232.DataLayer.LocationDL();

Location fetchedLocation = await locationDL.Get\_Location("The Palm Dubai");

if (fetchedLocation != null)

{

Debug.Log("Location for address found");

}

}

/// <summary>

/// Test method to save project data in database.

/// </summary>

public async void Save\_Project()

{

SWEN5232.DataLayer.ProjectDL projectDL = new SWEN5232.DataLayer.ProjectDL();

Project dummyProject = new Project

{

ProjectName = "My First Project",

Building = new Building

{

Height = 435,

Length = 487,

Width = 444,

Shape = "Rectangle",

Name = "Trump Tower",

Type = "Office Building"

},

Location = new Location

{

Address = "725 5th Ave, New York, NY 10022, USA",

Latitude = "40.7625",

Longitude = "-73.9738",

Altitude = "78z"

},

FloodLevel = 7

};

bool isSaved = await projectDL.Save\_Project(dummyProject);

var msg = isSaved ? "Project data saved in database" : "Project Not Saved";

Debug.Log(msg);

}

/// <summary>

/// Test method to delete a project.

/// </summary>

public async void Delete\_Project()

{

SWEN5232.DataLayer.ProjectDL projectDL = new SWEN5232.DataLayer.ProjectDL();

Project dummyProject = new Project

{

ProjectID = "1dc464de-b520-480b-843a-3d318387ab40"

};

bool isDeleted = await projectDL.Delete\_Project(dummyProject);

var msg = isDeleted ? "Project deleted from database" : "Project Not Deleted.";

Debug.Log(msg);

}

/// <summary>

/// Test method to open a project from database.

/// </summary>

public async void Open\_Project()

{

SWEN5232.DataLayer.ProjectDL projectDL = new SWEN5232.DataLayer.ProjectDL();

Project project = await projectDL.Open\_Project("2b3edc08-de88-495c-b479-885472217909");

if (project != null)

{

Debug.Log("Project " + project.ProjectName + " details are fetched successfully");

}

}

/// <summary>

/// To set the list of dummy locations in firebase.

/// </summary>

public void setdummylocation()

{

SWEN5232.DataLayer.LocationDL locationDL = new SWEN5232.DataLayer.LocationDL();

locationDL.SetDummyLocations();

}

}

# 12. Sprint Schedule

|  |  |  |
| --- | --- | --- |
| **S/No** | **TASKS** | **SPRINTS DUE** |
| **1** | **Project Decision, Requirement Analysis, Software Installations, GitHub setup, Kanban** | **(01-23-2019) to (02-06-2019)** |
| **2** | **Brainstorming, Architecture, Use Cases, Paper Prototype** | **(02-06-2019) to (02-20-2019)** |
| **3** | **Dll creation and test Harness, Class Diagram, Agreed upon function calls, Establish connection between Unity and Firebase** | **(02-20-2019) to (03-06-2019)** |
| **4** | **Create and Store data, Save and Delete Project data, Test data** | **(03-06-2019) to (03-20-2019)** |

# 13. References

1. **[https://firebase.google.com/docs/database/](https://www.youtube.com/watch?v=jZMwwZHJXJc)**
2. **<https://www.youtube.com/watch?v=jZMwwZHJXJc>**
3. **<https://unity3d.com/learn/tutorials>**
4. **<https://savvyapps.com/blog/firebase-realtime-database-vs-cloud-firestore-for-your-app>**
5. **<https://firebase.google.com/docs/database/unity/retrieve-data>**
6. **<https://firebase.google.com/docs/libraries/>**