

THE UNIVERSITY OF THE WEST INDIES

Semester l **■** Semester II **□** Supplemental/Summer School **□**

**Software Requirement Specifications**

**Pizza Factory Locator Subsystem – MongoDB**

**Course Code**: SWEN 4001 **Title**: Advanced Database Systems

ID No. \_\_416001313/190904\_ Name. \_\_\_Ron Alexis\_\_\_\_\_

Signature \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ID No. \_\_\_\_417000841/190905\_\_\_\_\_ Name. \_\_\_Sekaji Maynard\_\_

Signature \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ID No. \_\_\_\_416001294/190906\_\_\_\_\_ Name. \_\_\_Jason Charles\_\_

Signature \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date \_\_\_\_\_3rd January 2021\_\_\_\_\_ Pages \_\_\_\_\_\_\_\_\_34\_\_\_\_\_\_\_\_\_\_\_

The following is filled by examiner.

Score \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Examiner signature \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Table of Contents

[1. Introduction 2](#_Toc60533416)

[1.1. Purpose of the document 2](#_Toc60533417)

[1.2. Objectives 2](#_Toc60533418)

[1.3. Background 2](#_Toc60533419)

[1.4. Pizza Factory Locator Subsystem 2](#_Toc60533420)

[1.5. Stakeholders 3](#_Toc60533421)

[1.6. References 3](#_Toc60533422)

[2. Overall Description 4](#_Toc60533423)

[2.1. Project Perspective 4](#_Toc60533424)

[2.2. Product Function 4](#_Toc60533425)

[2.3. Classes and Characteristics 4](#_Toc60533426)

[2.4. Operating Environment 5](#_Toc60533427)

[2.5. DESIGN AND IMPLEMENTATION CONSTRAINTS 5](#_Toc60533428)

[3. System Features 6](#_Toc60533429)

[3.1. External Interface Requirements 6](#_Toc60533430)

[3.2. Functional Requirements Specification 7](#_Toc60533431)

[3.2.1. Use Case Diagram 7](#_Toc60533432)

[3.2.2. Use Case Description 8](#_Toc60533433)

[3.3. Functions implemented 11](#_Toc60533434)

# Introduction

## Purpose of the document

The purpose of this document is to serve as a specification requirements document for the Factory Locator Subsystem for the Pizza Delivered Quickly project. The contents of this document include analysis of the requirements, the applications of various testing techniques, and an explanation of the use of MongoDB as a backend. Two databases were used, MongoDB and Elasticsearch This document was produced to verify that the produced software meets the acceptance qualifications specified by the stakeholders. This document is meant to supplement the documents already provided as part of the various project management knowledge areas.

## Objectives

The objective of this documents is to provide visual representations of the flow of the software and its backend, details about the code used and its significance. This documentation also displays screenshots of the produced software to ensure that the produced applications achieved high quality. The requirements produced by the stakeholders have also been converted into visual representation, this was to ensure that the team clearly understood that which was stipulated by the stakeholders.

## Background

Over the past three months Pizza Delivered Quickly (PDQ) has observed a 30% decrease in revenue due most to a drop in home deliveries, this decrease was caused by a rival who promoted a guaranteed pizza delivery within a less time than that of the business. PDQ previously utilised computers for in-store operations and the usual business functions, however, the processing of home delivery orders was not heavily dependent on software systems. Therefore, the president, Dee Livery, commissioned a project consisting of six (6) application software that will all be subsystems of the overall project. These six projects are Pizza Factory Locator Subsystem, Order Entry Subsystem, Order Submit Subsystem, Logistics Subsystem, Routing Subsystem and Inventory Management Subsystem. This document speaks directly to the development of the Pizza Factory Locator Subsystem.

## Pizza Factory Locator Subsystem

The Pizza Factory Locator Subsystem is a software meant to identify “pizza factory” locations. This subsystem aims to ease the means at which the locations of factories are viewed and determined. This is done by allowing users to view the locations of existing factories and stores all on the same map. The user can also edit and add locations of both store and factories which will then be displayed on the previously mentioned map. This software is then completed with a MongoDB database where all the locations of the factories, as well as retail stores, are stored.

## Stakeholders

A stakeholder is any person involved or affected by project activities. Some of the sponsors specified include the president and CFO of Pizza Delivered Quickly who both are sponsors of the project, this means that they provide things such as consent and budget for the project. The project manager is also a stakeholder who overseas that the various project activities are undertaken and completed correctly. Finally, the end users are also considered stakeholders as they are who the application is made for. During the development of this all the aforementioned stakeholders were closely involved in the various processes.

## References

* Project Charter
* Project Management Plan
* Requirements Traceability Matrix
* Milestone list
* Stakeholder Register
* Scope Statement
* Quality Metrics

# Overall Description

## Project Perspective

The Pizza Delivered Quickly project was launched to help the business increase revenue that was recently decreased due to a competitor. The business previous only used computer for in-store affairs and not that of home deliveries, this project was aimed to improved internal affairs and introduce a digitizes manner to deal with the processing of home deliveries. The overall project was divided into six (6) subsystems all of which had their own unique purpose and goal. This document is specific to one of those subsystems, Pizza Factory Locator. This subsystem is to allow for users to easily manage factory locations as well as that of retail stores. All the information entered to the software is then saved into a MongoDB database. Agile development approach was utilized predominantly during the development cycle for this project.

## Product Function

The produced software allows for the user to add and edit locations of factories and retail stores. These locations are then saved in a MongoDB database. The software then reads the database for the locations entered by the user to create pins on a displayed map.

## Classes and Characteristics

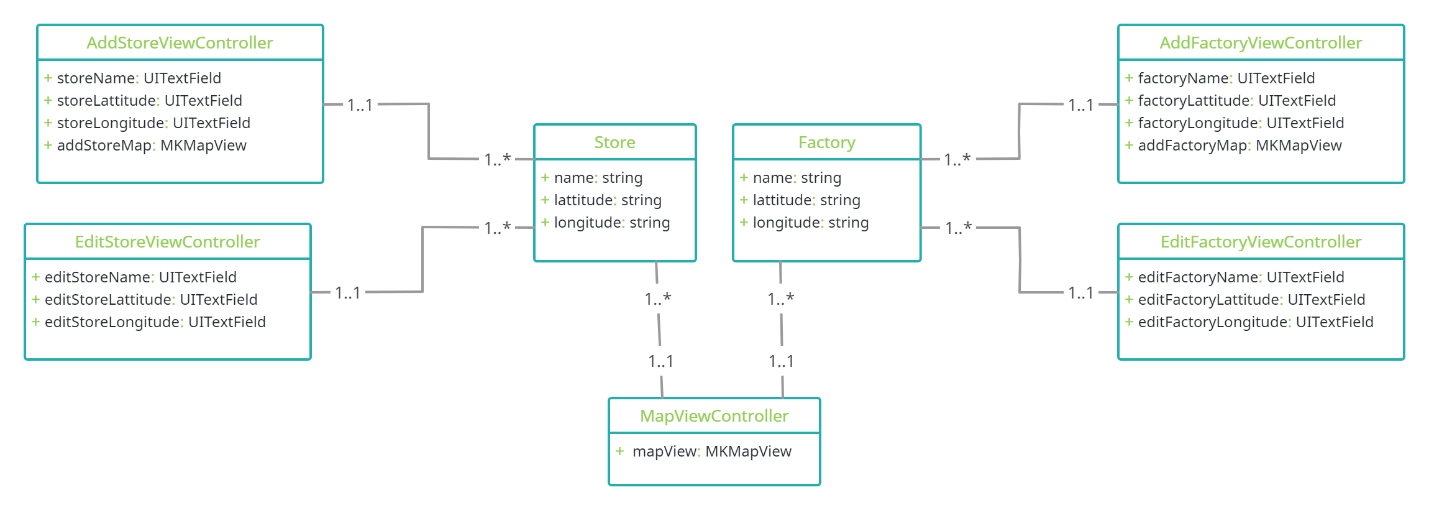


Figure 1 Class Diagram

The image above portrays a simple class diagram for Pizza Factory Locator subsystem. The classes Store and Factory or the classes that captures the data being entered, creates an object with the information and then prepares it to be saved into the database. These classes are also used when a class is calling from the database.

The application was coded in Swift as it produced an iOS application, this explains why four (4) of the classes have “ViewController” within their name. AddStoreViewController and AddFactoryViewController are the classes that were used to allow the users to add stores and factories respectively. These classes connected to the user interface, where the user would make input, it then took that information created its respective object, whether it be a store or factory, to be stored within the database.

EditStoreViewController and EditFactoryViewController classes are also connected to both user interface and the database. These classes pull the information in the database and display them on the user interface to enable the user to edit information saved in the database. They also update the database with the information entered by the user.

The MapViewController pulls the locations from the database and creates pins with the name and states whether it is a store or factory. These pins are then displayed on a map for the user to view on the user interface.

## Operating Environment

The software application shall be programmed in XCode using Swift language. The database, MongoDB, was connected within the code using Realm. On the user end, they will require any device that runs iOS.

## Design and Implementation Constraints

* 1. The interface should be user friendly. As the ages of the end-user’s application were not specified the interface was designed to cater users of all ages.
  2. The allotted time for the project was limited.
  3. The coding of the project must coincide with the standards set by the Institute of Electrical and Electronics Engineers (IEEE).

# System Features

## External Interface Requirements

* + 1. **Hardware Interface:** Any devices the runs iOS.
    2. **Software Interface:** MongoDB was used to create a database. The user interface is an application built in SwiftUI which is an innovative, exceptionally simple way to build user interfaces across all Apple platforms with the power of Swift.
    3. **Communication Interface:** Realm was used within the application to ensure a connection to the MongoDB database that was created. This allows for all the information to be saved and corrected managed on the database.

## Functional Requirements Specification

### Use Case Diagram

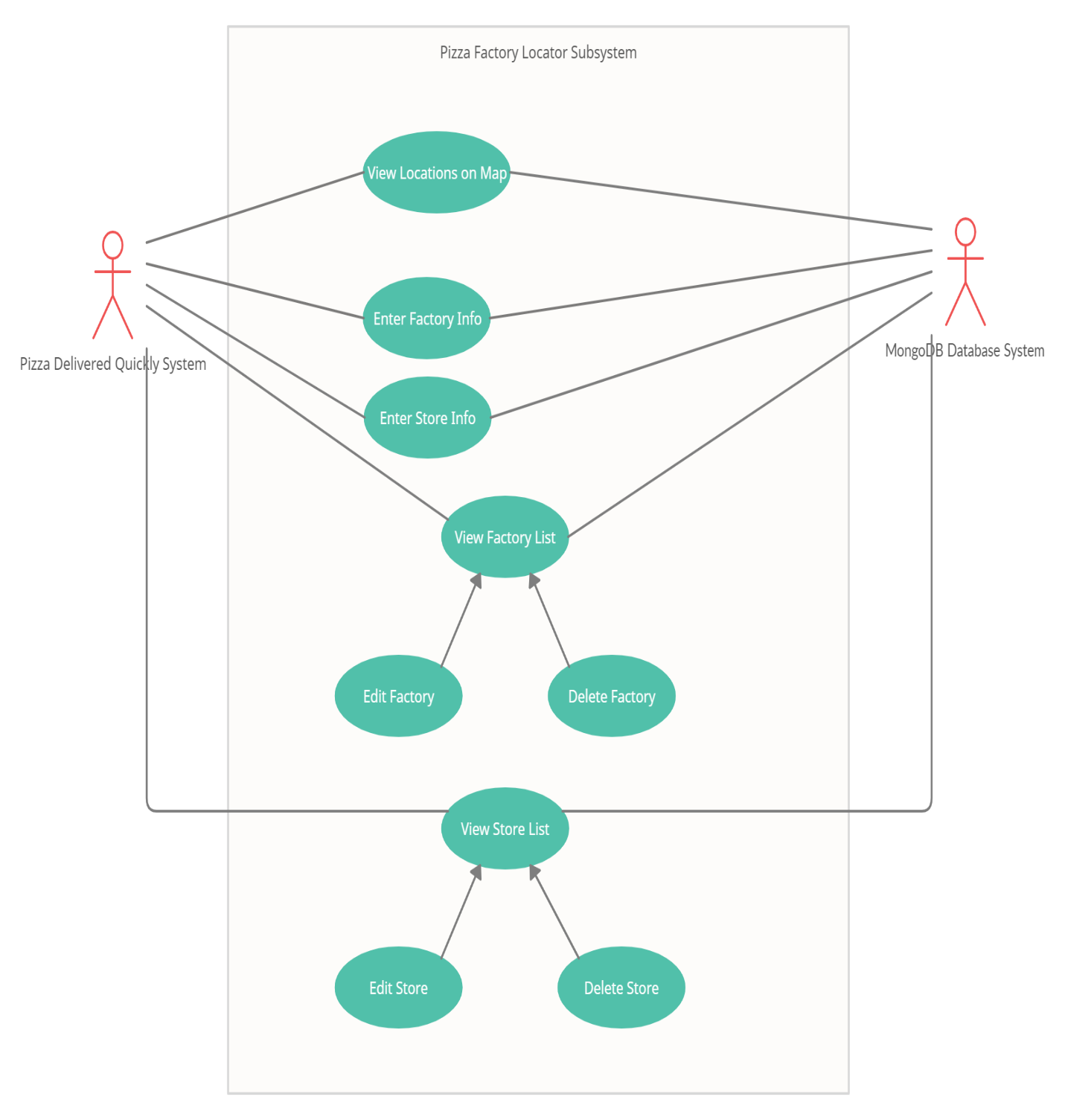


Figure 2 Use Case Diagram

### Use Case Description

**Use Case 1: View Locations on Map**

**Summary:** The actor is able to view the locations of the stores and factory displayed as pins on a map

**Actor:** Pizza Delivered Quickly System

**Precondition:** Locations already exist with the database

**Post Condition:** The actor views the map.

**Main Sequence:**

1. The actor is on the home page.
2. The actor selects “View Map” from the menu list.
3. The system retrieves locations from the database.
4. The system shall use the retrieved info to create pins,
5. The actor is presented with a map with the created pins.

**Use Case 2: Enter Factory Info**

**Summary:** The actor is able to enter the locations of a factory and it is displayed as a pin on a map

**Actor:** Pizza Delivered Quickly System

**Precondition:** None

**Post Condition:** The actor has added a new factory location.

**Main Sequence:**

1. The actor is on the home page.
2. The actor selects “Add New Factory Location” from the menu list.
3. The “Add New Factory Location” page appears where the actor can enter the factory’s information.
4. The actor enters the factory’s name, latitude and longitude then clicks “Save Factory”.
5. The system then saves the new location in the database and creates and displays a pin on the map where the new factory will be.

**Use Case 3: Enter Store Info**

**Summary:** The actor is able to enter the locations of a store and it is displayed as a pin on a map

**Actor:** Pizza Delivered Quickly System

**Precondition:** None

**Post Condition:** The actor has added an additional store.

**Main Sequence:**

1. The actor is on the home page.
2. The actor selects “Add A Retail Store Location” from the menu list.
3. The “Add Store Location” page appears where the actor can enter the store’s information.
4. The actor enters the store’s name, latitude and longitude then clicks “Save Store”.
5. The system then saves the new location in the database and creates and displays a pin on the map where the new store will be.

**Use Case 4: View Factory List**

**Summary:** The actor is able to view a list that contains all the locations for factories saved in the database.

**Actor:** Pizza Delivered Quickly System

**Precondition:** Factory locations already exist within the database

**Post Condition:** The actor has viewed a list with all the factories.

**Main Sequence:**

1. The actor is on the home page.
2. The actor selects “View Factory List” from the menu list.
3. The “Factory List” page appears where the actor can view the information for all the factories in the database.

**Use Case 4.1: Edit Factory**

**Summary:** The actor is able to edit a factory that already exists in the database.

**Actor:** Pizza Delivered Quickly System

**Precondition:** The selected factory location already exists within the database

**Post Condition:** The actor has edited a factory location.

**Main Sequence:**

1. The actor is on the “View Factory List” page.
2. The actor selects the factory they desire to edit from the displayed list.
3. The actor is presented with a screen that allows for them to edit the selected factory’s name, latitude and longitude.
4. The actor edits the factory location and then clicks “Update Factory”.
5. The system then sends the update to the database and redirects the actor to the “View Factory List” page.

**Use Case 4.2: Delete Factory**

**Summary:** The actor is able to delete a factory that already exists in the database.

**Actor:** Pizza Delivered Quickly System

**Precondition:** The selected factory location already exists within the database

**Post Condition:** The actor has deleted a factory location.

**Main Sequence:**

1. The actor is on the “View Factory List” page.
2. The actor selects the factory they desire to edit from the displayed list.
3. The actor is presented with a screen that allows for them to edit the selected factory’s name, latitude and longitude.
4. The actor clicks “Delete Factory”.
5. The system then sends the delete request to the database and the database removes all references to the selected information. The actor is returned to the “View Factory List” page.

**Use Case 5: View Store List**

**Summary:** The actor is able to view a list that contains all the locations for stores saved in the database.

**Actor:** Pizza Delivered Quickly System

**Precondition:** Store locations already exist within the database

**Post Condition:** The actor has viewed a list with all the stores.

**Main Sequence:**

1. The actor is on the home page.
2. The actor selects “View Store List” from the menu list.
3. The “Store List” page appears where the actor can view the information for all the stores in the database.

**Use Case 5.1: Edit Store**

**Summary:** The actor is able to edit a store that already exists in the database.

**Actor:** Pizza Delivered Quickly System

**Precondition:** The selected store location already exists within the database

**Post Condition:** The actor has edited a store location.

**Main Sequence:**

1. The actor is on the “View Store List” page.
2. The actor selects the store they desire to edit from the displayed list.
3. The actor is presented with a screen that allows for them to edit the selected store’s name, latitude and longitude.
4. The actor edits the store’s location and then clicks “Update Store”.
5. The system then sends the update to the database and redirects the actor to the “View Store List” page.

**Use Case 5.2: Delete Store**

**Summary:** The actor is able to delete a store that already exists in the database.

**Actor:** Pizza Delivered Quickly System

**Precondition:** The selected store location already exists within the database

**Post Condition:** The actor has deleted a store location.

**Main Sequence:**

1. The actor is on the “View Store List” page.
2. The actor selects the store they desire to delete from the displayed list.
3. The actor is presented with a screen that allows for them to edit the selected factory’s name, latitude and longitude.
4. The actor clicks “Delete Store”.
5. The system then sends the delete request to the database and the database removes all references to the selected information. The actor is returned to the “View Store List” page.

## Functions implemented

Backend

* Save – Once the user has entered the information required an object is then created using this information and then saved into the database.
* Update – The data requested from the database is displayed, then the entered data is used to update the database.
* Delete - The data requested from the database is displayed, then once confirmed all references to the displayed data is removed from the database

Frontend

* Add Sore / Factory – Allows the user to enter the name and location of a store or factory. It then uses the location to place a pin on the map.
* View Store/ Factory List- Once selected the user can view a list of all stores or all factories.
* Edit Store/ Factory- From the View Store/ Factory function, the user can then select the store or factory they wish to edit. Once on the edit page the user can either save the updated information, delete the information or cancel the edit.
* View Store and Factory Location- The user is allowed to view a map that displays pins where all the entered stores and factories are located.

## 

* 1. **Non-Functional Requirements Specification**

Some of the non-functional requirement of the application software include (but is not limited to):

* + 1. **Usability** – Simplified user interface with limited information to be displayed. The age of intended user was not specified; therefore, the application was designed to my user friendly to users of all ages.
    2. **Performance** – Can support all users within the business who are required to use. Software application needs to be able to handle many users at one time.
    3. **Deployability**- The application’s deployment process shall be fully automated and allow for a number of environments: development, test, and production.

# Course Project Report Requirements

## 4.1. Introduction of the database models

The classes Store and Factory are objects that that capsulate the data from the input of the user and transforms it into readable data for the database. The MongoDB database is connected to the application using Realm. See below for screenshots of the declaration of the objects and some of their uses:

* + 1. Characteristics of Store Object

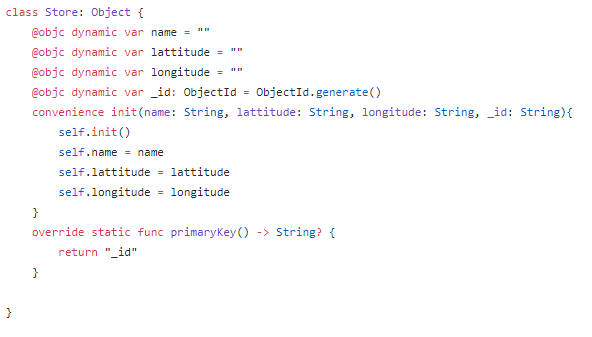


Figure 3 Store Object

The above photo depicted the definition of the variables needed to save the information for the Store object.

* + 1. Characteristics of Factory Object

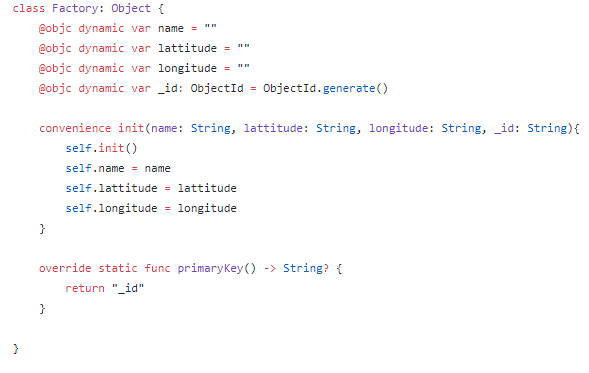


Figure 4 Factory Object

The above photo depicted the definition of the variables needed to save the information for the Factory object.

## Introduction of the Realm Connection

The MongoDB database was connected by Realm within the code. In order for the this to be successful cocoa pods had to be initially downloaded via command line into the project folder. Once the pods were correctly configured, the application is ready to connect to the database. Below are snippets of code where Realm was instantiated and utilized to connect to the database and manipulate the data.

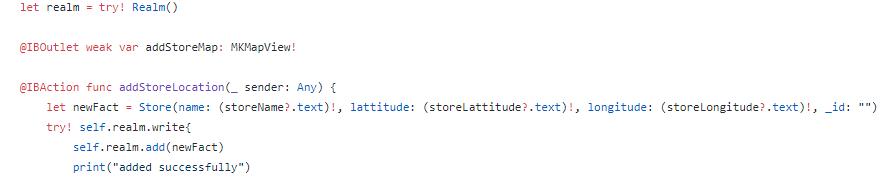


Figure 5 Creating Store in Database

The image above portrays the creation of a Store object in the database. First an instance of Realm is created, then the object using the entered info is created.Ffinally using Realm built in functions the new store object is sent to the database to be added

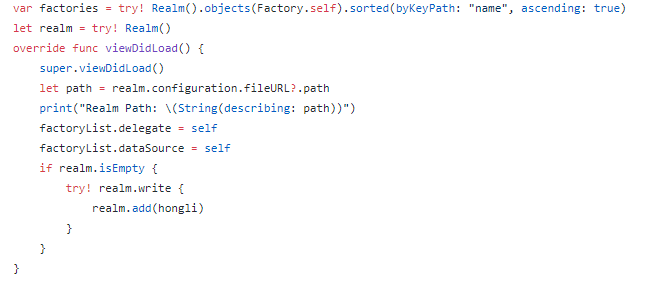


Figure 6 Realm Function

The image above portrays various Realm functions being utilized. At the top of the image a list of all the factories which are being pulled from the database, an instance of Realm is then being created. The path of the local database file is also being created.

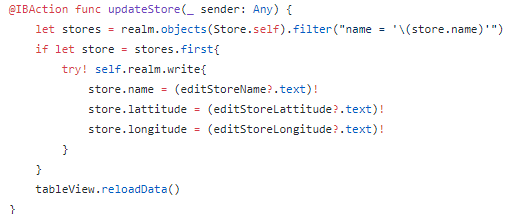


Figure 7 Update Store Object

The above image portrays the processes involved in updating the database using Realm functions. The desired store is first retrieved using its name. that page allowing the user to edit the information will appear. Once the user is ready to save the update and they click update store the application the calls the write function to update the database with the entered information

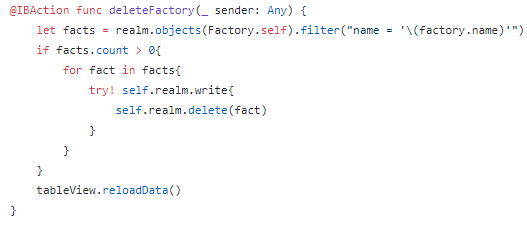


Figure 8 Remove from Database

The above image portrays the process of deleting from the database using Realm. Firstly, it requests the information related to the selected factory name, then using the built-in functions it deletes all references related to that name form the database.

# Appendix

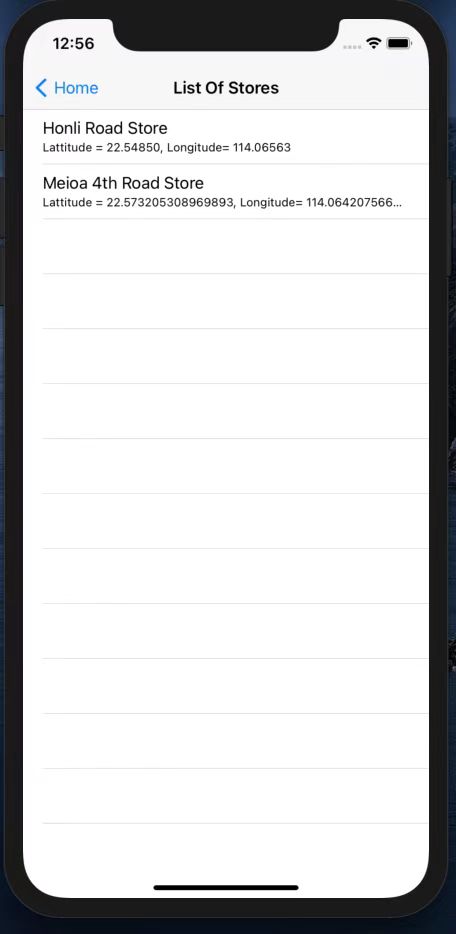
Figures below show screenshots of the functioning application:

Figure 9

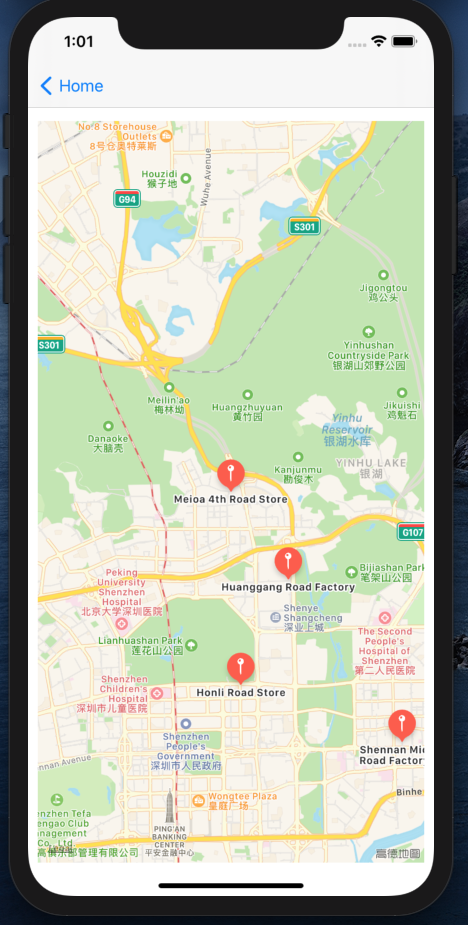


Figure 10

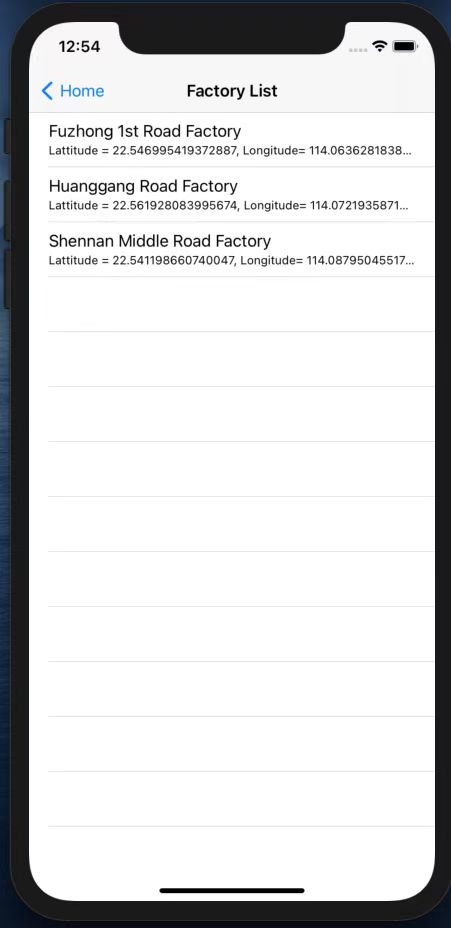


Figure 11

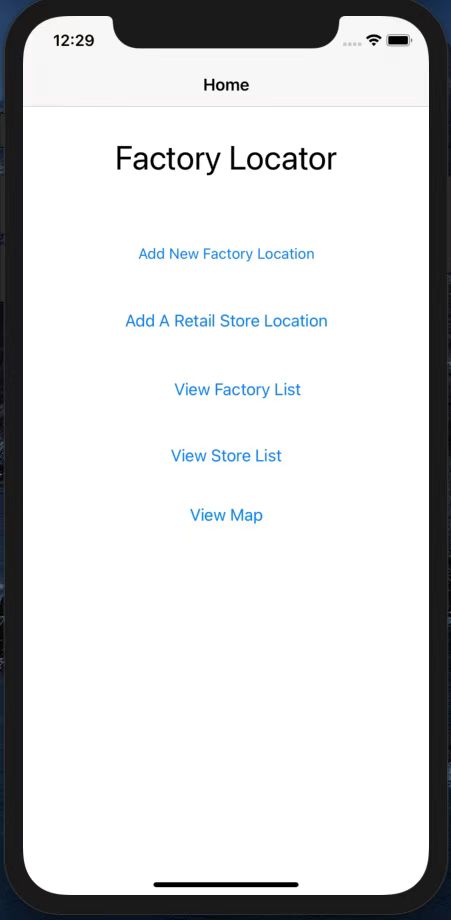


Figure 12

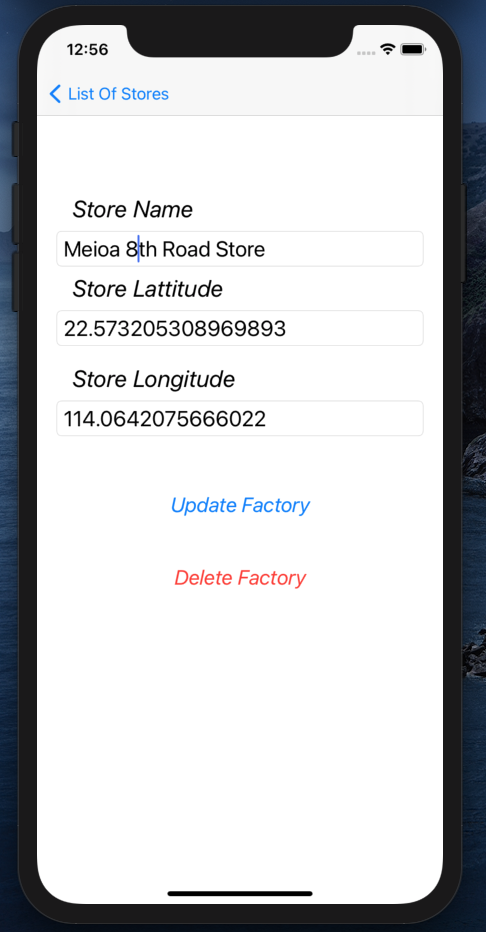


Figure 13

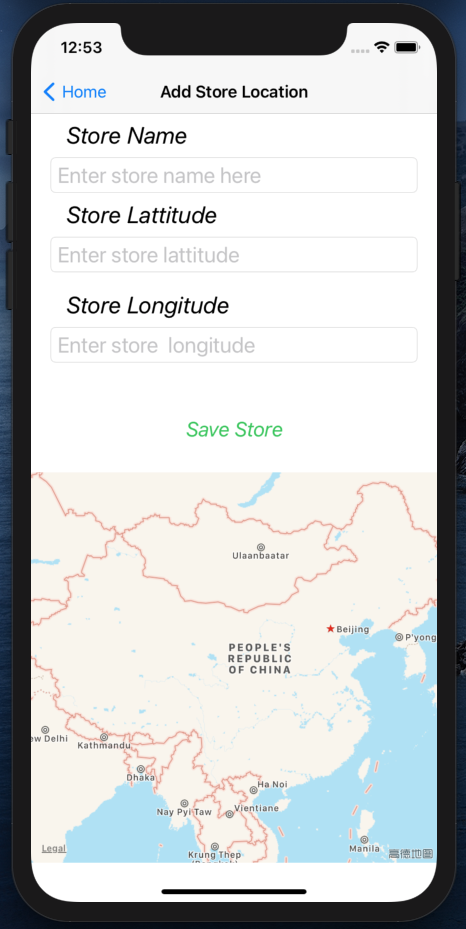


Figure 14

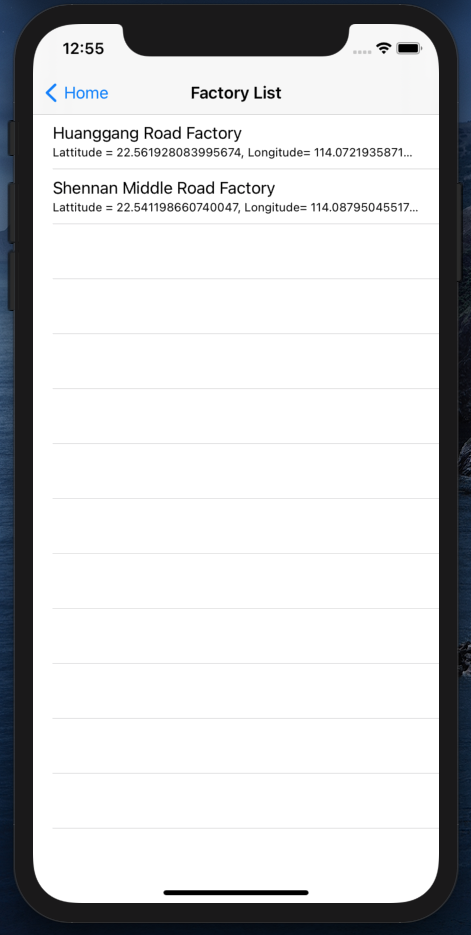


Figure 15

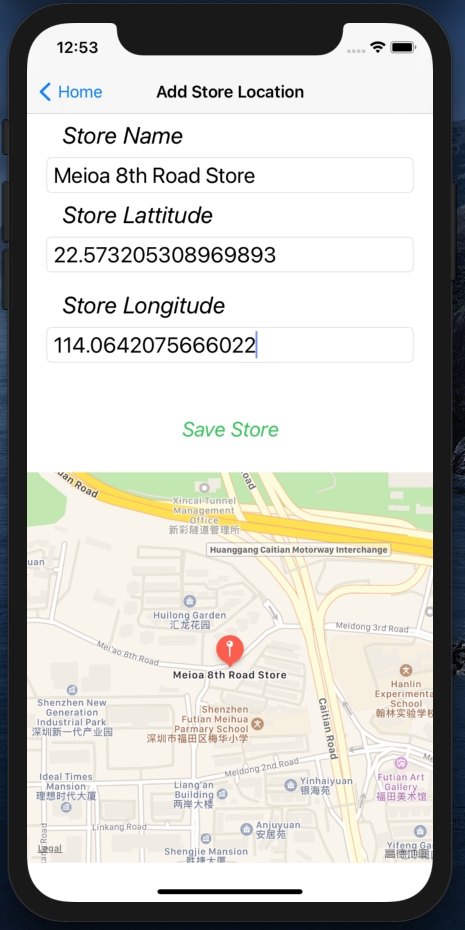


Figure 16

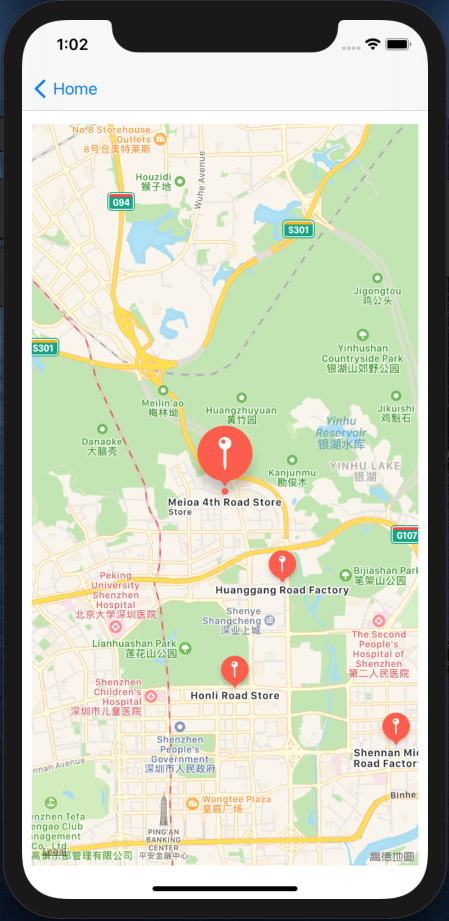


Figure 17

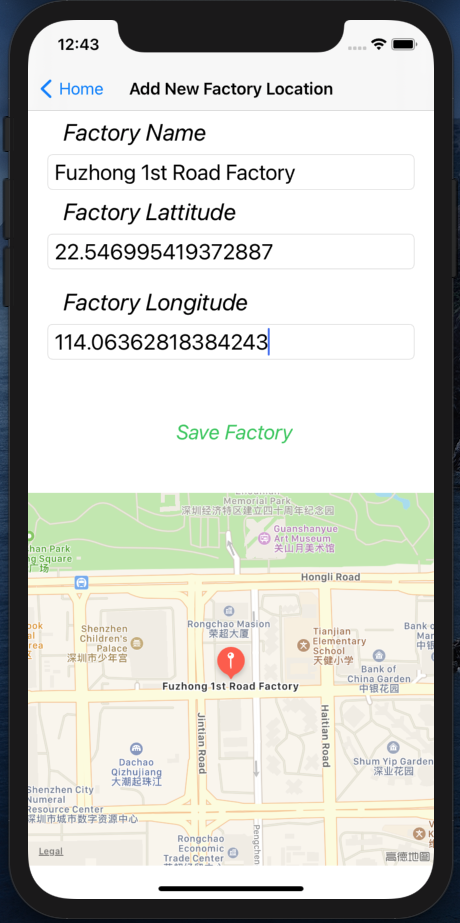


Figure 18

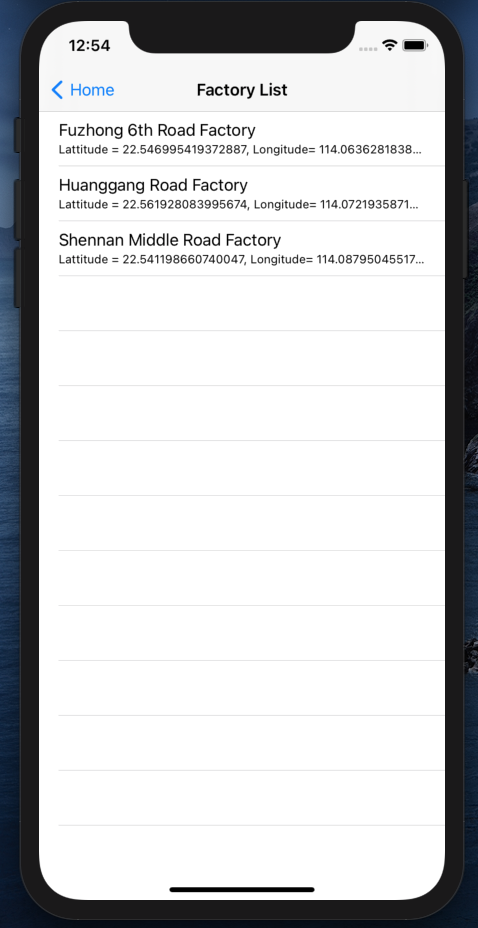


Figure 19

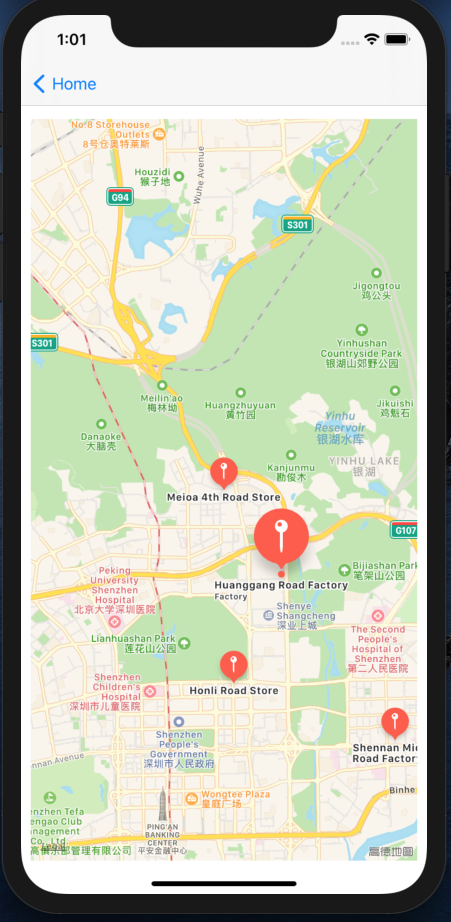


Figure 20

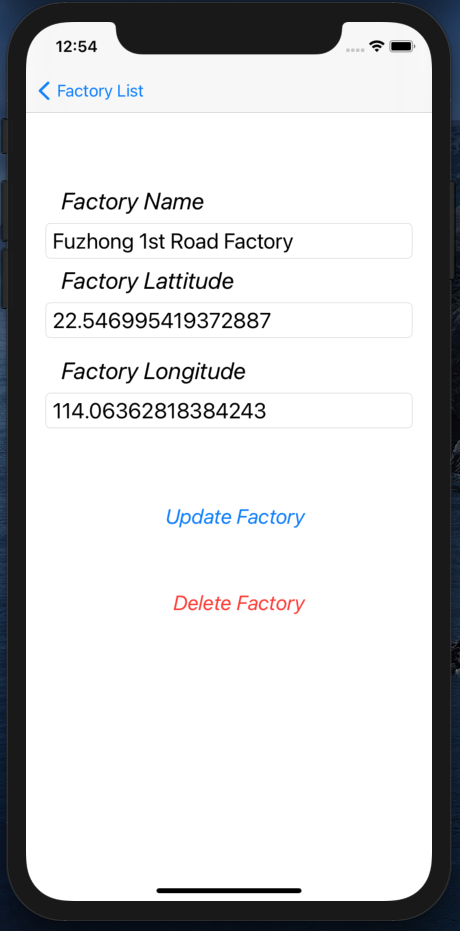


Figure 21

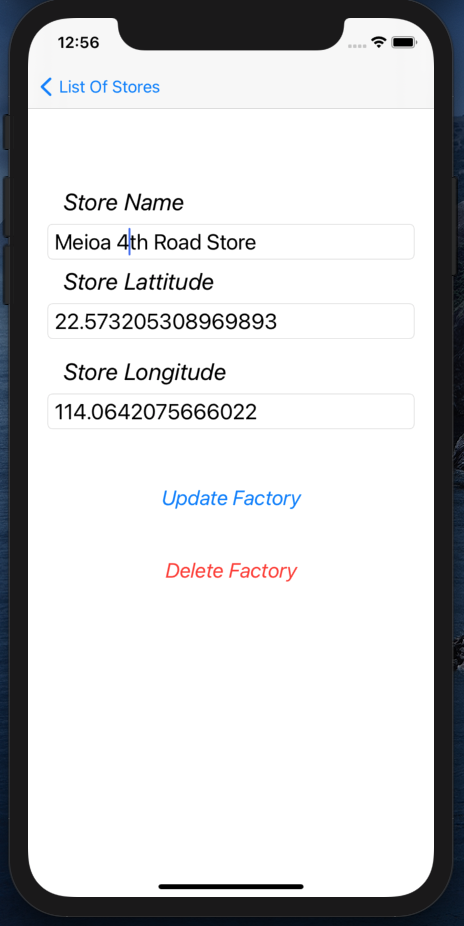


Figure 22

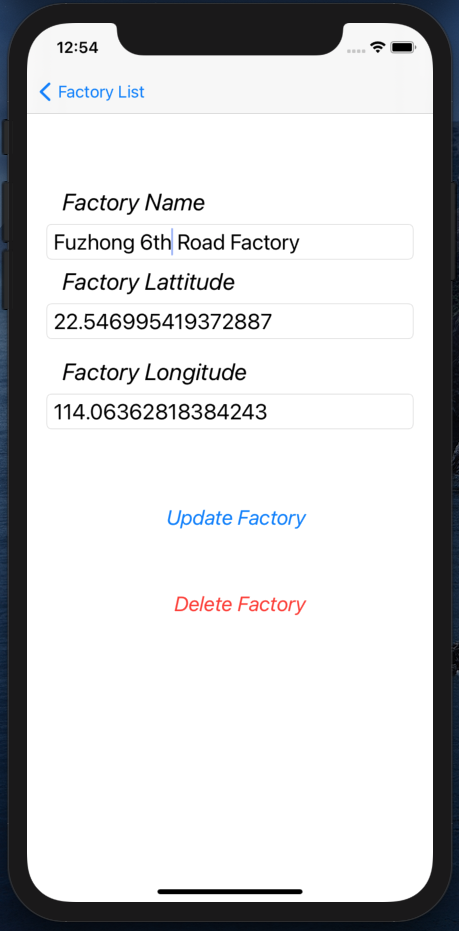


Figure 23

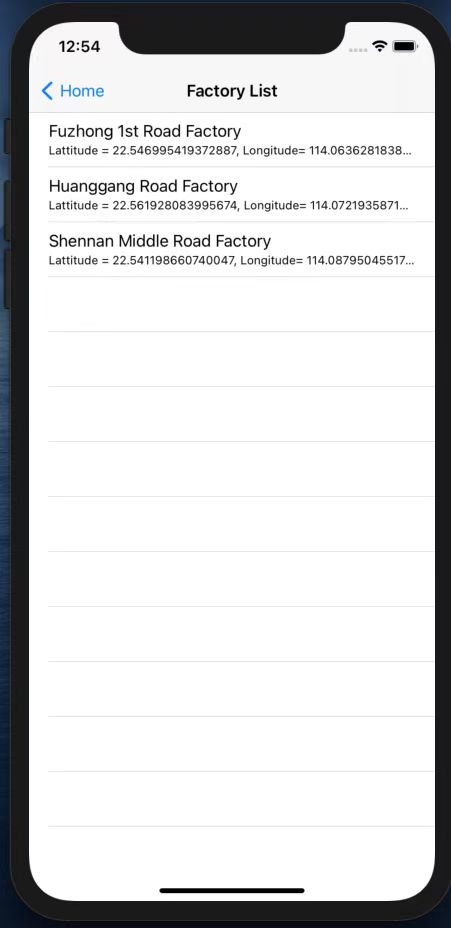


Figure 24

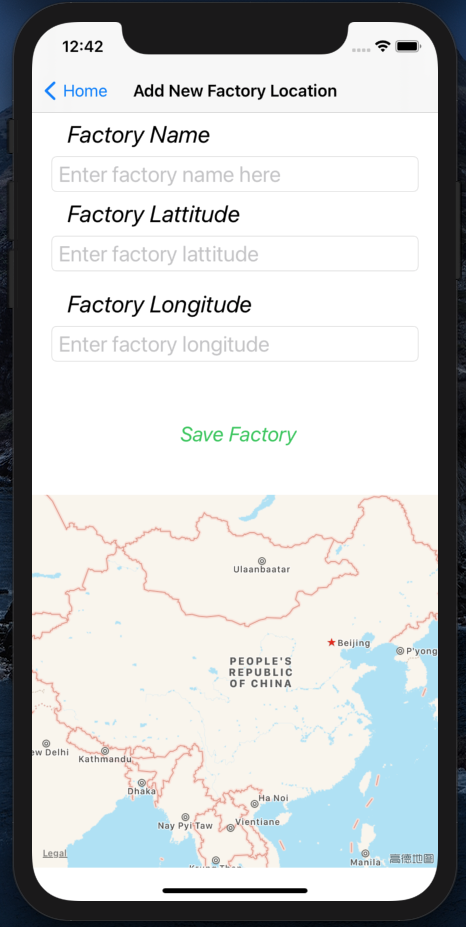


Figure 25