# Creating Data-Driven Mobile Applications

**COS80020**

ASSIGNMENT 11 – Experience Report

# Accessibility Map in Melbourne

# Student name: Duy Thuc Pham

# Student number: 101767225

# Introduction

The intention of this application is to support the disabilities people in Melbourne. It includes five features: show accessibility places in a map, bookmark accessibility places, show accessibility detail information, display a direction between current location to accessibility destination via Apple Map, search accessibility places by name.

In this assignment, three ILOs – which are reading data from web API, handling bad conditions to maintain user experience and making asynchronous API to increase performance - are also applied to improve the user experience. Firstly, regarding the ILO 1, this program has integrated with the data from Melbourne Data (<https://data.melbourne.vic.gov.au/Property-Planning/Buildings-with-name-age-size-accessibility-and-bic/pmhb-s6pn/data)> for the accessibility place in Melbourne as well as combined with Google Place API to get images for these places. Secondly, if those requests are failure, the appropriate message will display to notify users regarding this issue. Last but not least, in order to optimize the performance when requesting a large amount of data, in map view, users are only to see accessibility places around them at least 500 meters and at most 5 km. Likewise, when searching data by name, the pagination has been implemented to limit data to 10 places per request.

# Summary

## App Screen

|  |  |
| --- | --- |
| Screen | iPhone 8 |
| Map Screen | ../../../../../../Desktop/Simulator%20Screen%20Shot%20-%20iPhone%208%20-%202018-05 |
| Detail Window Screen | ../../../../../../Desktop/Simulator%20Screen%20Shot%20-%20iPhone%208%20-%202018-05 |
| Search Screen | ../../../../../../Desktop/Simulator%20Screen%20Shot%20-%20iPhone%208%20-%202018-05 |
| Bookmark Screen | ../../../../../../Desktop/Simulator%20Screen%20Shot%20-%20iPhone%208%20-%202018-05 |
| Detail Location Screen | ../../../../../../Desktop/Simulator%20Screen%20Shot%20-%20iPhone%208%20-%202018-05 |
| Filter Screen | ../../../../../../Desktop/Simulator%20Screen%20Shot%20-%20iPhone%208%20-%202018-05 |

## API

Accessibility Map application uses data which is collected by CLUE – City of Melbourne's Census of Land Use and Employment (Socrata Developer Portal | Socrata). The information includes the building name, accessibility rating, accessibility type and building coordinate which is really helpful. In order to communicate with these RESTful API through HTTPs, the developer must use this url to retrieve accessibility places information

<https://data.melbourne.vic.gov.au/resource/q8hp-qgps.json>

### App Tokens

The application must register an app token to identify and add it to X-App-Token in HTTP header.

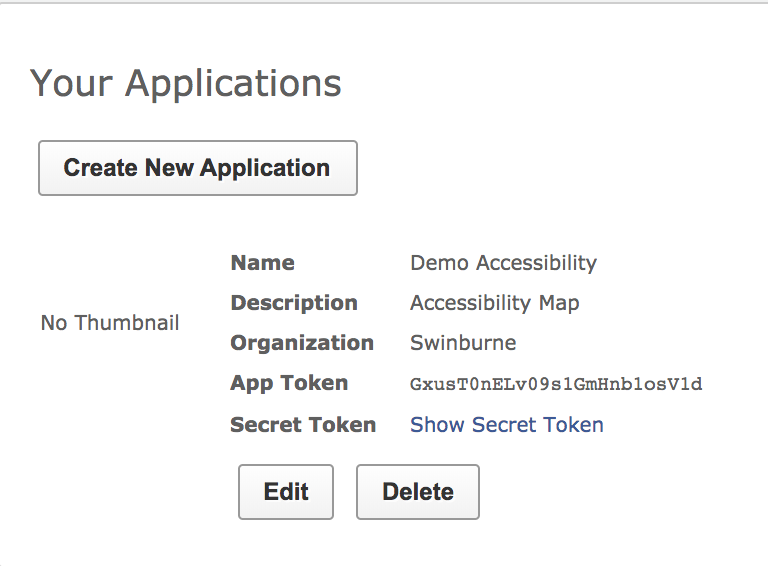


Figure : Create Soda API app token

### API Request Structure

The data is not good as many places are duplicated, some don’t have building name or accessibility type. Therefore, it is suggested to include filter parameters while requesting data. For example, in this program, to search all data with pagination by 10 places per page.

https://data.melbourne.vic.gov.au/resource/q8hp-qgps.json?$limit=10&$offset=0&$where=building\_name!='' AND accessibility\_type!='' AND census\_year = 2016 &$select=block\_id, count(block\_id), accessibility\_rating, accessibility\_type, accessibility\_type\_description, lower(building\_name), location, street\_address,suburb,x\_coordinate,y\_coordinate&$group=block\_id, accessibility\_rating, accessibility\_type, accessibility\_type\_description, lower(building\_name), location, street\_address,suburb,x\_coordinate,y\_coordinate&$order=block\_id

The Soda API works as SQL query, therefore the parameters that are being used are those $select, $where, $limit, $offset, $order, $group.

1. $select: Select fields that necessary for the application
2. $where: only get the data which has building name and accessibility type name
3. $limit: limit the data to 10 items
4. $offset: the beginning of the data
5. $group: group data which has the same value to remove duplicate
6. $order: get the data in order

### API Response Structure

This is the API response data

|  |
| --- |
| [  {  "accessibility\_rating": "0",  "accessibility\_type": "Not determined or not applicable",  "accessibility\_type\_description": "Access has not been rated",  "block\_id": "1",  "location": {  "type": "Point",  "coordinates": [  144.957229905,  -37.82088351  ]  },  "building\_name": "Northbank Place East",  "street\_address": "507-541 Flinders Street",  "suburb": "Melbourne (CBD)",  "x\_coordinate": "144.957229905",  "y\_coordinate": "-37.82088351"  }  ] |

### Integrated API

Three APIs use in this program to retrieve data from server.

**1. Get places with in radius**

This API uses to get all accessibility places with in the given radius and display them as markers in map view. It uses for the map screen to display these places in the map as a marker. By doing this the number of requesting place will be reduced and whereby the performance will improve.

https://data.melbourne.vic.gov.au/resource/q8hp-qgps.json?$where=building\_name!='' AND accessibility\_type!='' AND census\_year = 2016 AND within\_circle(location, -37.813628, 144.963058, 1000)&$select=block\_id, count(block\_id), accessibility\_rating, accessibility\_type, accessibility\_type\_description, lower(building\_name), location, street\_address,suburb,x\_coordinate,y\_coordinate&$group=block\_id, accessibility\_rating, accessibility\_type, accessibility\_type\_description, lower(building\_name), location, street\_address,suburb,x\_coordinate,y\_coordinate&$order=block\_id

It uses the same parameters as the example above except including within\_circle() in the $where request to filter data in the radius 1 km around the user coordinate.

**2. Get all places with pagination**

This API uses to retrieve places with limit is 10 and offset will increase by 10 per request. Due to the fact that the requested data has more than 10000 records, so it is certain that pagination will improve the performance for the application.

https://data.melbourne.vic.gov.au/resource/q8hp-qgps.json?$limit=10&$offset=0&$where=building\_name!='' AND accessibility\_type!='' AND census\_year = 2016 &$select=block\_id, count(block\_id), accessibility\_rating, accessibility\_type, accessibility\_type\_description, lower(building\_name), location, street\_address,suburb,x\_coordinate,y\_coordinate&$group=block\_id, accessibility\_rating, accessibility\_type, accessibility\_type\_description, lower(building\_name), location, street\_address,suburb,x\_coordinate,y\_coordinate&$order=block\_id

**3. Get place by name with pagination**

This API applies to search data based on the building name, which help users to find the appropriate accessibility place.

https://data.melbourne.vic.gov.au/resource/q8hp-qgps.json?$limit=10&$offset=0&$where=building\_name!='' AND accessibility\_type!='' AND census\_year = 2016 AND starts\_with(building\_name, 'A')&$select=block\_id, count(block\_id), accessibility\_rating, accessibility\_type, accessibility\_type\_description, lower(building\_name), location, street\_address,suburb,x\_coordinate,y\_coordinate&$group=block\_id, accessibility\_rating, accessibility\_type, accessibility\_type\_description, lower(building\_name), location, street\_address,suburb,x\_coordinate,y\_coordinate&$order=block\_id

The starts\_with() is added in $where to find the accessibility places with the appropriate given name.

## Request API and Parsing Data

### Model

In order to parsing Json response from API, the Building object is created and used as a model in this application. This object applies the latest API of Swift 4 to parse Json more effectively by implementing Decodable and using enumeration CodingKeys to map the Json key with the object variable. To illustrate let see the Appendix A.

### Request Restful API

This program uses Alamofire to improve the quality of requesting Http methods in Swift. It is considered as the wrapper class for URLRequest, also it reduces number of codes and extendable. For example, this is the code that uses to request and validate http response status code from 200 to 399 with Alamofire which is being called in RequestAPIManager class.

@discardableResult func request(type: RequestService, completionHandler: @escaping (\_ result: Result<Data>) -> Void) -> DataRequest {

        let request = Alamofire.request("\(type.baseURL)\(type.path)", method: type.method, parameters: type.parameters, headers: type.headers).validate().responseData { handler in

            switch handler.result {

            case .success:

                completionHandler(.success(handler.result.value))

                break

            case .failure(let error):

                completionHandler(.failure(error))

                break

            }

        }

        return request

    }

In addition, this application creates a protocol name RequestType including the baseURL, path, HTTP request method, parameters and headers. Thereby, it creates an enumeration RequestService that will implement RequestType protocol so that every new API will configure in the same place.

**RequestType protocol**

import UIKit

import Alamofire

protocol RequestType {

    var baseURL: URL {get}

    var path: String {get}

    var method: HTTPMethod {get}

    var parameters: [String: Any]? {get}

    var headers: [String: String]? {get}

}

**RequestService enumeration (see Appendix B)**

For instance, to integrate with three APIs that mentioned above. Three cases in RequestService enumeration is created with the parameter. Then it implements RequestType protocol and defines the configuration value for each case.

Then in the RequestAPIManager class, when we request an API just simply call the enumeration that we have already configured before. For instance, this is how I request API to retrieve all buildings with pagination and parse json to object.

public func getBuildings(paging: PaginationRequest, filterData: String? = nil, completionHandler: @escaping (\_ result: Result<[Building]>) -> Void) -> DataRequest {

        let dataRequest = request(type: .getBuilding(paging: paging, filterData: filterData)) { responseHandler in

            switch responseHandler {

            case .success(let data):

                do {

                    let buildings = try JSONDecoder().decode([Building].self, from: data!)

                    completionHandler(.success(buildings))

                } catch let error {

                    debugPrint(error.localizedDescription)

                    completionHandler(.failure(error))

                }

                break

            case .failure(let error):

                completionHandler(.failure(error))

                break

            }

        }

        return dataRequest

    }

### Google Place API

The program uses Google Place API to get images of these places by installing Place SDK for iOS to cocoa pods. The code below demonstrates how the Place SDK has been used in this application.

public func getPlaceImage(buildingName: String, suburb: String, completionHandler: @escaping (\_ result: Result<UIImage>) -> Void) {

        if let imageFromCache = cacheManager.getImage(forKey: buildingName) {

            completionHandler(.success(imageFromCache))

            return

        }

        let filter = GMSAutocompleteFilter()

        filter.type = .noFilter

        let query = "\(buildingName) ,Melbourne VIC, Australia"

        GMSPlacesClient.shared().autocompleteQuery(query, bounds: nil, filter: filter) { (result, error) in

            if let error = error {

                completionHandler(.failure(error))

                return

            }

            if let result = result, result.count > 0 {

                self.getPhoto(placeID: (result.first?.placeID)!, completionHandler: { (photo, error) in

                    if let error = error {

                        completionHandler(.failure(error))

                        return

                    }

                    if let photo = photo {

                        self.cacheManager.setImage(image: photo, forKey: buildingName)

                        completionHandler(.success(photo))

                    }

                })

            } else {

                completionHandler(.failure(RequestError.noImage("No Image")))

            }

        }

    }

GMSPlacesClient.shared.autocompleteQuery function uses to retrieve a google place ID and then it will be used to request the image with GMSPlacesClient.shared.lookUpPhotos().

## Project Structure

### MVC

The project follows MVC architecture so it has been separated into three groups Model, View and Controller. Also, it has a Shared group in which includes Manager classes such as APIManager, Network Manager and CoreDataManager as well as Extensions class for supporting the project.

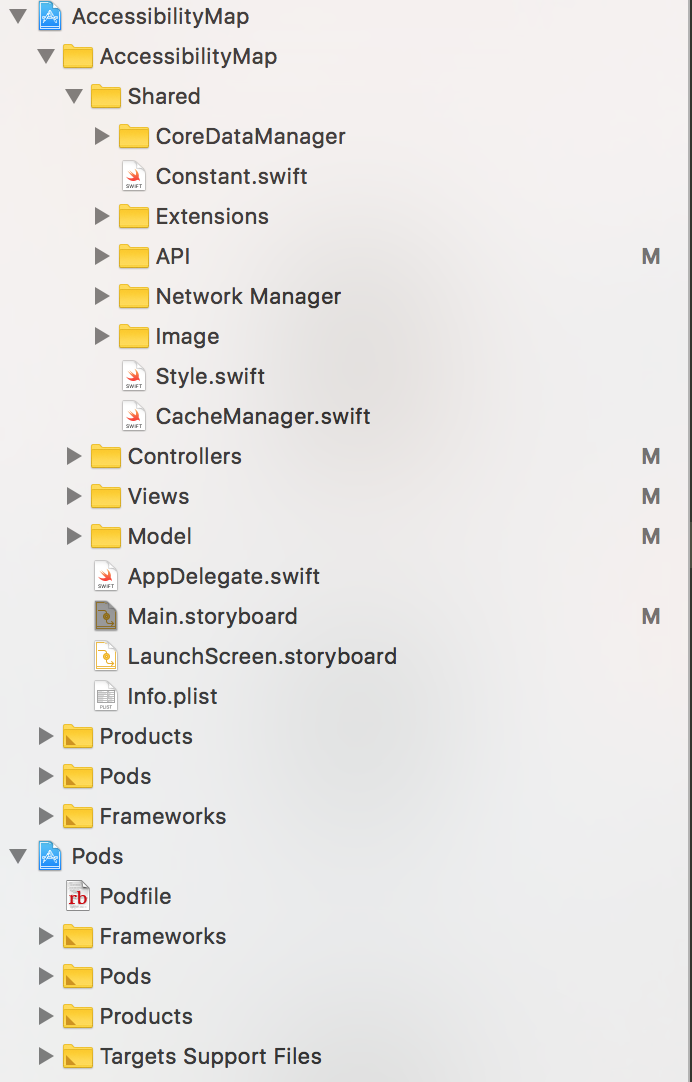


Figure : Project structure



Figure : Model View Controller

**Model**

Model has the Building model, API model – which uses to pagination – and xcdatamodeld Core Data file.

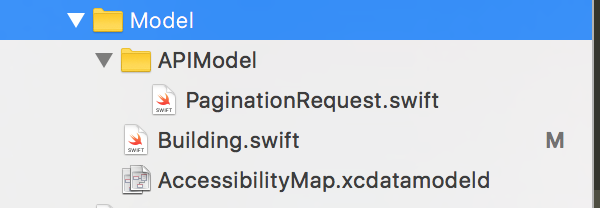


Figure :Model structure

**Controller**

Controller includes Controller classes to get data and update it to view

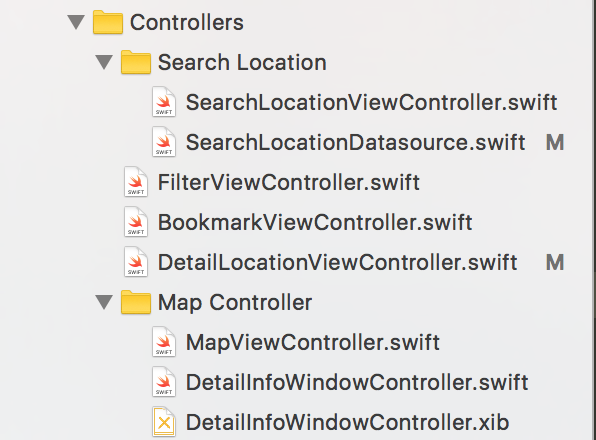


Figure : Controller structure

**View**

View includes all custom views and table view cells which take part in showing UI to user.

### ../../../../../../Desktop/Screen%20Shot%202018-05-29%20at%201

Figure : View Structure

### App Flow

**Main storyboard**

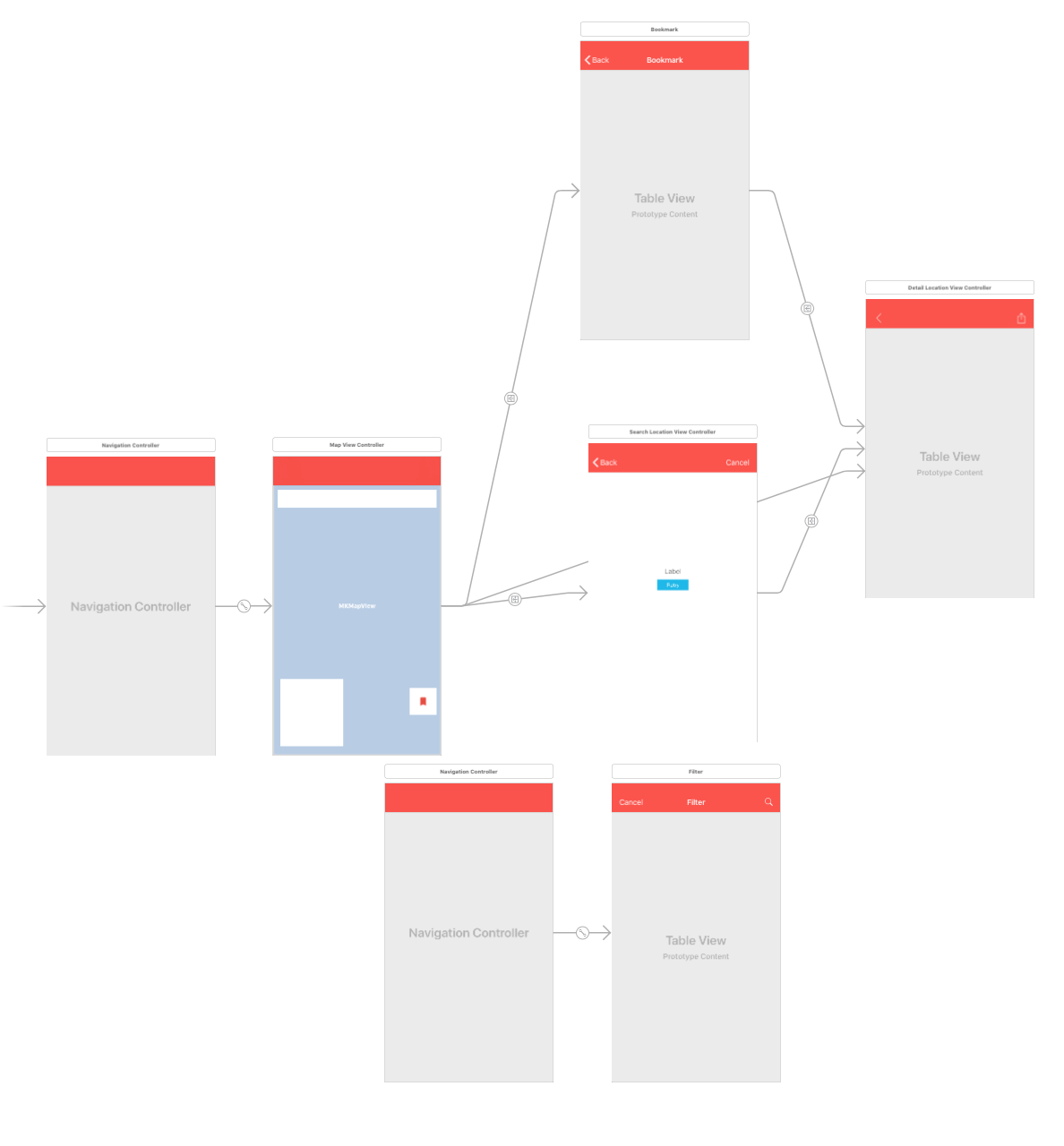
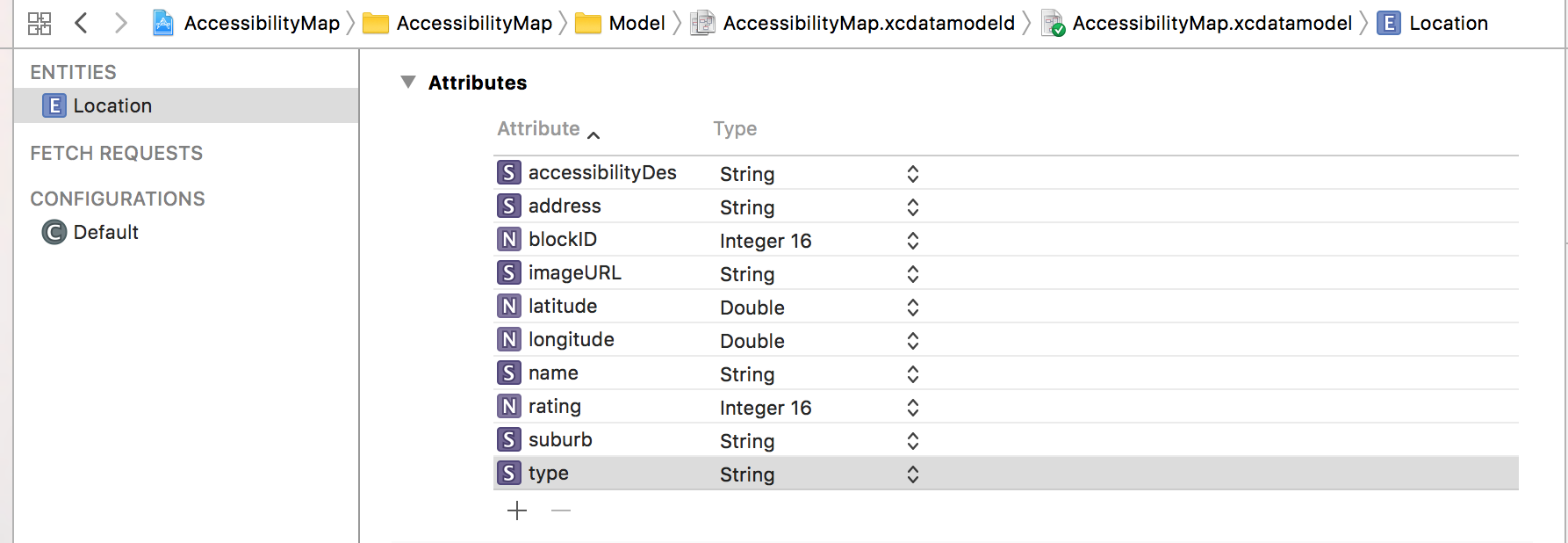


Figure : Main story board

The app begins with Map View, when user select the marker it will display map info window. Then if the user selects detail info window, it will redirect to Detail View. When user selects bookmark button, it will go to Bookmark view, then if user want to see a detail information, they can click on a cell to go to Detail View. This behavior is the same for Search View. Likewise, when user selects the filter button on Map Screen it will direct to Filter screen

### Database Entity



The local database is created to store bookmark accessibility places. The LocationService (Appendix C) object has been implemented to make the CRUD process easier.

### Package Manager

There are 6 third party libraries that are used in this application (

platform :ios, '9.0'

target 'AccessibilityMap' do

use\_frameworks!

pod 'Alamofire', '~> 4.7'

pod 'BLMultiColorLoader'

pod 'ReachabilitySwift'

    pod 'SkeletonView'

    pod 'ParallaxHeader', '~> 2.0.0'

    pod 'GooglePlaces'

end

**Alamofire**

Which uses to request HTTP method

**BLMultiColorLoad**

Which uses to custom loading indicator

**ReachabilitySwift**

This uses for notifying when the network connection is lost

**SkeletonView**

This uses to replace loading indicator and make the loading cell looks like Facebook Shimmer effect

**ParalaxHeader**

This library uses to custom a UIImage header view for table view.

**GooglePlace**

This is Google Place SDK for iOS to request image for specific place.

# Conclusion

This application has covered three ILOs of this unit. The quality of reading and parsing API data improves significantly when combining Alamofire with RequestType protocol. It not only reduces the number of code, but it also reusable in other projects. Likewise, any errors that occur by limit condition of network will be catch and display sufficient information to user in UI. On top of that, these APIs are requested and handled asynchronously which do not freeze the UI and users are able to interact with the UI normally while loading data.

Although there are still some issues in this application, it demonstrates how I understand the ILOs and able to develop a data driven application in iOS platform effectively.

# References

*Socrata Developer Portal | Socrata*, Dev.socrata.com, viewed 29 May 2018, <https://dev.socrata.com/foundry/data.melbourne.vic.gov.au/q8hp-qgps>.

*Simple Filtering | Socrata*, *Dev.socrata.com*, viewed 29 May 2018, <https://dev.socrata.com/docs/filtering.html>.

# Appendix A

### Building class

import UIKit

struct Building: Decodable {

    private(set) var address: String

    private(set) var name: String

    private(set) var blockId: String

    private(set) var longitude: Double

    private(set) var latitude: Double

    private(set) var suburb: String

    private(set) var rating: Int

    private(set) var type: String

    private(set) var accessibilityDes: String

    enum BuildingKeys: String, CodingKey {

        case address = "street\_address"

        case name = "lower\_building\_name"

        case blockId = "block\_id"

        case longitude = "x\_coordinate"

        case latitude = "y\_coordinate"

        case suburb = "suburb"

        case rating = "accessibility\_rating"

        case type = "accessibility\_type"

        case accessibilityDes = "accessibility\_type\_description"

    }

    init(from decoder: Decoder) throws

    {

        let values = try decoder.container(keyedBy: BuildingKeys.self)

        address = try values.decode(String.self, forKey: .address)

        name = try values.decode(String.self, forKey: .name)

        blockId = try values.decode(String.self, forKey: .blockId)

        longitude = try Double(values.decode(String.self, forKey: .longitude)) ?? 0.0

        latitude = try Double(values.decode(String.self, forKey: .latitude)) ?? 0.0

        suburb = try values.decode(String.self, forKey: .suburb)

        rating = try Int(values.decode(String.self, forKey: .rating)) ?? 0

        type = try values.decode(String.self, forKey: .type)

        accessibilityDes = try values.decode(String.self, forKey: .accessibilityDes)

    }

    init (location: Location) {

        self.address = location.address!

        self.name = location.name!

        self.blockId = String(location.blockID)

        self.longitude = location.longitude

        self.latitude = location.latitude

        self.suburb = location.suburb!

        self.rating = Int(location.rating)

        self.type = location.type!

        self.accessibilityDes = location.accessibilityDes!

    }

}

# Appendix B

## Request Service

import UIKit

import Alamofire

enum RequestService {

    case getBuilding(paging: PaginationRequest?, filterData: String?)

    case getBuildings(radius: Int, lat: Double, long: Double, filterData: String?)

    case getBuildingByName(name: String, paging: PaginationRequest?, filterData: String?)

}

extension RequestService: RequestType {

    var baseURL: URL { return URL(string: Constant.baseURL)! }

    var path: String {

        switch self {

        case .getBuilding:

            return ""

        case .getBuildings:

            return ""

        case .getBuildingByName:

            return ""

        }

    }

    var method: HTTPMethod {

        switch self {

        case .getBuilding, .getBuildings, .getBuildingByName:

            return .get

        }

    }

    var parameters: [String: Any]? {

        let filterParam: [String: Any] = ["$select" : "block\_id, accessibility\_rating, accessibility\_type, accessibility\_type\_description, lower(building\_name), location, street\_address,suburb,x\_coordinate,y\_coordinate", "$group" : "block\_id, accessibility\_rating, accessibility\_type, accessibility\_type\_description, lower(building\_name), location, street\_address,suburb,x\_coordinate,y\_coordinate", "$order" : "block\_id"]

        switch self {

        case .getBuilding(let paging, let filterData):

            var searchParam: [String: Any] = filterParam.dictByConcatinating(["$where" : "building\_name!='' AND accessibility\_type!='' AND census\_year=2016"])

            guard let paging = paging else {

                return searchParam

            }

            if let filterData = filterData {

                searchParam = filterParam.dictByConcatinating(["$where" : "building\_name!='' AND accessibility\_type!='' AND census\_year = 2016 AND \(filterData)"])

            }

            return searchParam.dictByConcatinating(paging.toDict())

        case .getBuildings(let radius, let lat, let long, let filterData):

            let radiusParam = "within\_circle(location,\(lat),\(long),\(radius))"

            if let filterData = filterData {

                return filterParam.dictByConcatinating(["$where" : "building\_name!='' AND accessibility\_type!='' AND census\_year = 2016 AND \(radiusParam) AND \(filterData)"])

            }

            return filterParam.dictByConcatinating(["$where" : "building\_name!='' AND accessibility\_type!='' AND census\_year = 2016 AND \(radiusParam)"])

        case .getBuildingByName(let name, let paging, let filterData):

            let searchNameParam = "starts\_with(building\_name,'\(name)')"

            var searchParam: [String: Any] = filterParam.dictByConcatinating(["$where" : "building\_name!='' AND \(searchNameParam) AND accessibility\_type!='' AND census\_year = 2016"])

            guard let paging = paging else {

                return searchParam

            }

            if let filterData = filterData {

                searchParam = filterParam.dictByConcatinating(["$where" : "building\_name!='' AND \(searchNameParam) AND accessibility\_type!='' AND census\_year = 2016 AND \(filterData)"])

            }

            return searchParam.dictByConcatinating(paging.toDict())

        }

    }

    var headers: [String: String]? {

        switch self {

        default:

            return ["X-App-Token": "\(Constant.appToken)"]

        }

    }

}

# Appendix C

## LocationService

import Foundation

import UIKit

import CoreData

extension Location {

    static let entityName = "Location"

}

class LocationService {

    /// NSManageObjectContext of Vehicle

    var context: NSManagedObjectContext

    /// Init with NSManagedObjectContext

    ///

    /// - Parameter context: curent context

    init(context: NSManagedObjectContext) {

        self.context = context

    }

    func insertLocation(building: Building) {

        let newItem =  NSEntityDescription.insertNewObject(forEntityName: Location.entityName, into: context) as! Location

        newItem.blockID = Int16(building.blockId)!

        newItem.accessibilityDes = building.accessibilityDes

        newItem.address = building.address

        newItem.latitude = building.latitude

        newItem.longitude = building.longitude

        newItem.name = building.name

        newItem.rating = Int16(building.rating)

        newItem.type = building.type

        newItem.suburb = building.suburb

    }

    func getAll() -> [Location] {

        return get(withPredicate: NSPredicate(value:true))

    }

    func getLocation(byBlockID blockID: Int) -> Location? {

        let predicate = NSPredicate(format: "blockID == %d", blockID)

        return get(withPredicate: predicate).first

    }

    func get(withPredicate queryPredicate: NSPredicate) -> [Location] {

        let fetchRequest = NSFetchRequest<NSFetchRequestResult>(entityName: Location.entityName)

        fetchRequest.predicate = queryPredicate

        do {

            let response = try context.fetch(fetchRequest)

            return response as! [Location]

        } catch let error as NSError {

            // failure

            print(error)

            return [Location]()

        }

    }

    func delete(location: Location) {

        context.delete(location)

    }

    // Saves all changes

    func saveChanges(completion: (\_ completion: Bool) -> ()) {

        do{

            try context.save()

            completion(true)

        } catch let error as NSError {

            // failure

            debugPrint("Could not save \(error.localizedDescription)")

            completion(false)

        }

    }

}

# Appendix C

Link for source code <https://github.com/duythuc28/ass_iOSMap/tree/develop>