



# Eksibition - Virtual Exhibition Guide on Smart Phone

LYU 1503

Prepared by JIN, Peng (1155014559)

Supervised by Michael R. Lyu

*this page is intentionally left blank*

# Contents

<b>Contents</b>	<b>3</b>
<b>Abstract</b>	<b>6</b>
<b>1. Introduction</b>	<b>7</b>
1.1. Overview	7
1.2. Motivation	7
1.3. Objectives	8
1.4. Milestones	9
<b>2. Wireframe Design</b>	<b>10</b>
<b>3. Study of BLE Technology</b>	<b>11</b>
3.1. Hardware	11
3.1.1. Advertisement Frame	11
3.1.2. Ranging	11
3.2. Estimote Beacons	12
<b>4. Study of Indoor Positioning</b>	<b>13</b>
4.1. Approach by Analyzing RSSI	13
4.1.1. Trilateration	13
4.1.2. Accuracy	14
4.2. Approach by Using Estimote Indoor Location SDK	14
<b>5. Study of Custom Indoor Map Design</b>	<b>16</b>
5.1. Approach by Using Mapbox	16
5.2. Approach by Using a High-resolution Image	17
<b>6. Study of Developing Backend Using Node.js</b>	<b>19</b>
6.1. Building RESTful APIs With Express4	19
6.2. Using MongoDB to Save Data	21
<b>7. Developing Application With Swift</b>	<b>23</b>
7.1. Model-View-Controller	23
7.2. 3D Touch	23
7.2.1. Home Screen Quick Actions	24
7.2.2. Peek and Pop	25

7.3. Centralized Soundtrack Player	26
<b>8. Google Maps</b>	<b>27</b>
8.1. Google Maps SDK for iOS	27
8.2. Map View Animations	27
8.3. Marker customization	28
8.4. Info Window	28
8.5. Google Maps Directions API	29
8.5.1. Request format	29
8.5.2. Directions Responses	30
8.6. Navigation Information Panel	32
<b>9. Apple Pay</b>	<b>34</b>
9.1. Workflow of Apple Pay	34
9.2. Client-side	35
9.3. Server-side	35
9.4. Stripe	36
<b>10. Apple Wallet</b>	<b>38</b>
10.1. Turning a Ticket into a Pass	38
10.2. Sign and Distribute Passes	43
<b>11. Redeem System</b>	<b>44</b>
<b>12. News System</b>	<b>45</b>
12.1. UI Design	45
12.1.1. News List	45
12.1.2. News View	46
12.2. Study of Apple Push Notification Services (APNs)	46
12.2.1. The Path of a Remote Notification	47
12.2.2. Security Architecture	49
12.3. Pushing Notifications with Node.js	51
12.3.1. Node.js Module: APN Agent	51
12.4. Receiving and Processing News Notification in iOS	52
12.5. Core Data	53
<b>13. Multiple Language Support</b>	<b>54</b>

13.1.Client Side Design	54
13.2.Server Side Support	54
13.2.1.Database	54
13.2.2.Format of Response Data	55
<b>14. Account Managing System</b>	<b>56</b>
14.1.Client Side Design	56
14.1.1.Account Manager	56
14.2.Server Side Implementation	57
<b>15. Web Admin Panel</b>	<b>61</b>
<b>16. Backend Code Revision</b>	<b>67</b>
16.1.Standardize HTTP Response	67
16.2.Protecting APIs with Middleware	68
<b>17. Difficulties in the Project</b>	<b>71</b>
<b>18. Contribution and Reflection</b>	<b>73</b>
18.1.Contribution	73
18.1.1.Fall 2015	73
18.1.2.Spring 2016	73
18.2.Reflection	74
<b>19. Summary</b>	<b>75</b>
<b>Reference</b>	<b>76</b>

# Abstract

Eksibition is a mobile smart guide who can provide a great tour experience for visitors in museums and exhibitions. Our goal of the project is to implement the full features of this application on iOS devices.

In this report, we will go through all the topics that we have been studied, all the achievements we have made and all the problems we have encountered.

The first part is the introduction of this project, indicating the motivation of this project and how we are going to implement this idea. The second part is the original wireframe design of this application, briefly showing the features of it. Then we will describe our studying process of the Bluetooth Low Energy (BLE) technology. Since we are going to make use of iBeacon™, a sub-category of BLE device, to provide the guide service, a study on the hardware is very necessary. The rest part are the showcases of features we developed. We will also talk about the key technologies being used in each part and all the technical problems we overcame.

At the end of the report, a summary will be given to explain the division of labor and my contributions in greater detail.

# 1. Introduction

## 1.1. Overview

Apple introduced a protocol called iBeacon™ at the Apple Worldwide Developers Conference in 2013 [1]. iBeacon™ is a protocol allowing the Bluetooth Low Energy (BLE) devices to broadcast their identifier information to nearby portable electronic devices including BLE enabled smart phones [2]. Although it seems quite simple, applications can still do a lot of things by taking advantage of this technology. Our application Eksibition is one of them. Imaging a mobile application can know where you are in a museum and introduce the item or artwork in front of you through both text and audio, visiting a museum could be a more interesting activity even without a real guide explaining what is happening. Eksibition is not only a tour guide, but also an integrated smart extension for the museums, which can help visitors buy tickets, schedule visit time and redeem tickets, etc.

## 1.2. Motivation

Nowadays, people don't appreciate exhibitions and museums in a way it used to be. Sometimes when time is limited, we just go to the museums to see one or two items we are interested in, but we don't know the exact location of them. Sometimes when we see our favorite item, it may be inconvenient for us to learn the interesting background of it unless we Google it or take an extra device from the museum to listen to it. Or for some people, they usually have a real tour guide to guide them and introduce to them, with a cost ranging from 20 USD/hour up. On account of these facts, we feel it is really necessary to have an application that can make everything easier and more interesting when visiting a museum.

When we are trying to figure out how to make an application for users as if a guide will be walking next to them, we found the iBeacon™ and the Bluetooth Low Energy (BLE) technology very useful, which perfectly help to enhance the features of our application.

Such that, we had this project started, named Eksibition - Virtual Exhibition Guide on Smart Phone.

## 1.3. Objectives

As stated in the last section, we are developing an application to fulfill the demands from both visitors and museum organizations and also to improve the whole visiting experience. The features below are to be implemented:

- Ticketing system: Users can purchase tickets within our application and retrieve the tickets as Ticket Pass, which can be stored in the newly introduced Apple Wallet (it's called Passbook before iOS 9). The electronic tickets on the user's side can later be redeemed at the entrance of the museum. We also developed a redeem application, which is a separated companion application distributed to museum staffs to redeem users' tickets at the entrance. These whole process is paperless and environment-friendly.
- Showing nearby items: Users can see the nearest item in our application when getting close to it. They can also choose any one of the already visited items to view the detail information, which includes more images and descriptions. Users can even listen to the introduction soundtrack of items.
- Like and share: Users can like an item and share the related information to social networks.
- Map view of the museum: Users can see his location and all the items on a map of the museum, so that one can decide where to explore his favorite items.
- Direction guide: Application can help generate a direction based on the items selected by the user and his/her location.
- News notification system: Users can get notifications from the museum promoting the coming events when they are not physically in the museum.
- Multiple language support: Application can support switching language on the fly without restart the application.
- Analyzing of users' behavior: By analyzing the users' behavior from the ticket purchase history, liked items, etc, we can draw the conclusion of which items are users' favorites and what is the most popular routine, which can help the museums improve their service.

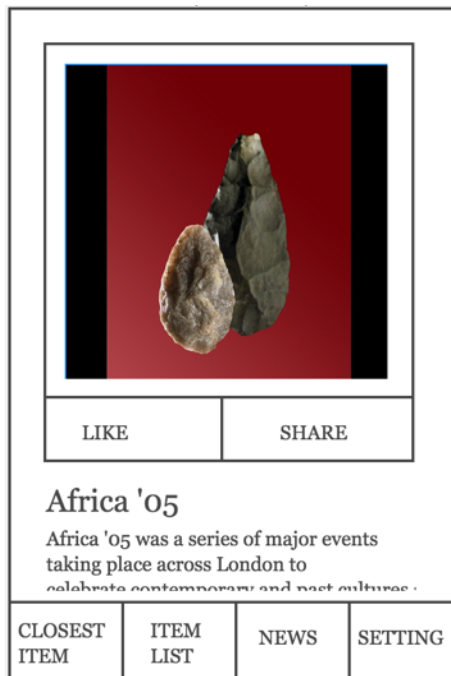
## 1.4. Milestones

To achieve those objectives above, there are some technical milestones required to be reached:

- A well-designed backend server with all the applications programming interfaces (API) implemented to handle the possible requests from the mobile client side.
- A Bluetooth signal analyzing mechanism within the application to detect the distances from iPhone to items.
- A map system to show the customized map of the museum, which also carries information of the items in exhibition.
- A positioning system analyzing the users' estimated position according to the different signals broadcasted by iBeacons.
- A ticketing system allowing the users to purchase and redeem tickets within our app.
- Apple Pay integration, which helps to make the purchasing process easier.
- Supporting tickets to be added and managed by Apple Wallet.
- Multi-language support.
- An analyzing mechanism to collect the related information and analyze the users' behavior.
- A web platform for event organizers to post and edit item.
- A web platform to promoting activities and send the notification to users.

## 2. Wireframe Design

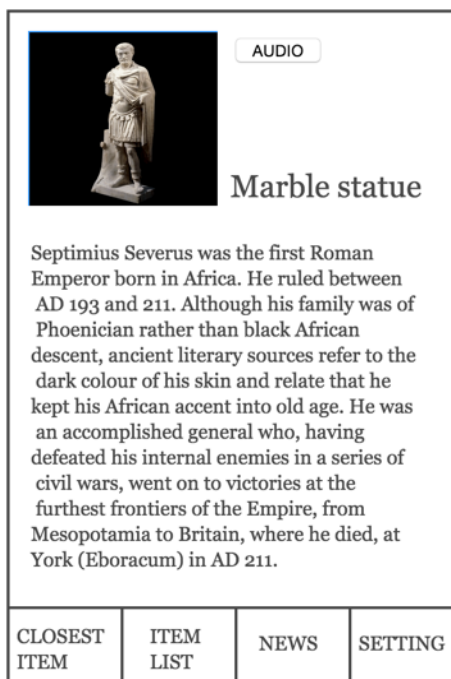
This part shows the initial wireframe design of our application in the beginning. Some basic features are shown.



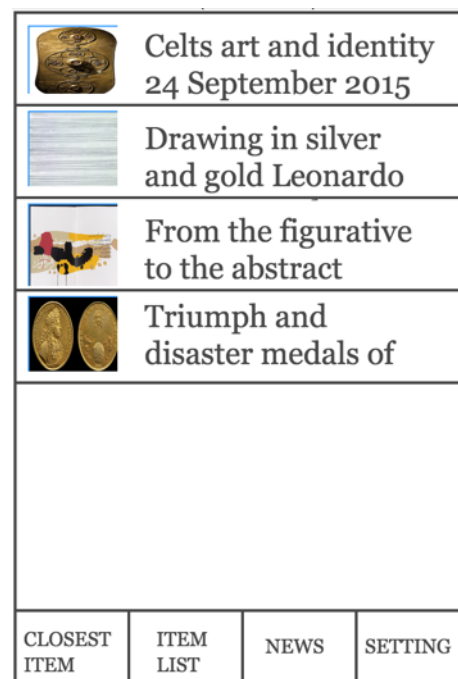
**Figure 1:** Closest tab shows the image and detail introductions of the nearest item, allowing users to like or share the related information.



**Figure 2:** Item list tab has a zoomable map view on the top area showing the museum map and item locations. Under the map is an item list showing all the items in the museum.



**Figure 3:** Item detail page let users check the detail information of this item



**Figure 4:** News tab shows the the promotions of museum events

## 3. Study of BLE Technology

### 3.1. Hardware

Bluetooth Low Energy (BLE) is one kind of Bluetooth technology developed by the Bluetooth Special Interest Group. BLE devices broadcast signals in a specific area.[3] Different from the traditional Bluetooth technology, BLE can provide consistent signal with very low energy consumption.

Devices adopting iBeacon™ protocol are typically called beacons. In our development, we use Estimote beacons as our BLE devices. Smartphones, tablets and other smart devices will get notified when they are in close proximity to a beacon.[2]

#### 3.1.1. Advertisement Frame

Beacons work in an advertisement mode, in which it broadcasts its signal and notify its presence to those smart devices who can receive the signal.[4] The signal a beacon broadcasts follows a fixed format defined by Apple's iBeacon™ protocol prefixed together with UUID, major and minor identifiers[5]. Below is an example of what an iBeacon™ advertisement frame will be like:

`fb0b57a2-8228-44 cd-913a-94a122ba1206 Major 1 Minor 2`

With this frame, we can make the signal broadcasted by each beacon carry different information. For example, we can set the UUID to represent the building where the beacon locates and set the Major and Minor numbers to represent different locations in this building.

#### 3.1.2. Ranging

Using the strangeness of the signal broadcast by beacons, iOS SDK will give an estimated distance between beacons and the phone. The result is categorized into the following 3 distinct ranges [6]:

Immediate: within 0.5 meters.

Near: within 3 meters.

Far: within 70 meters.

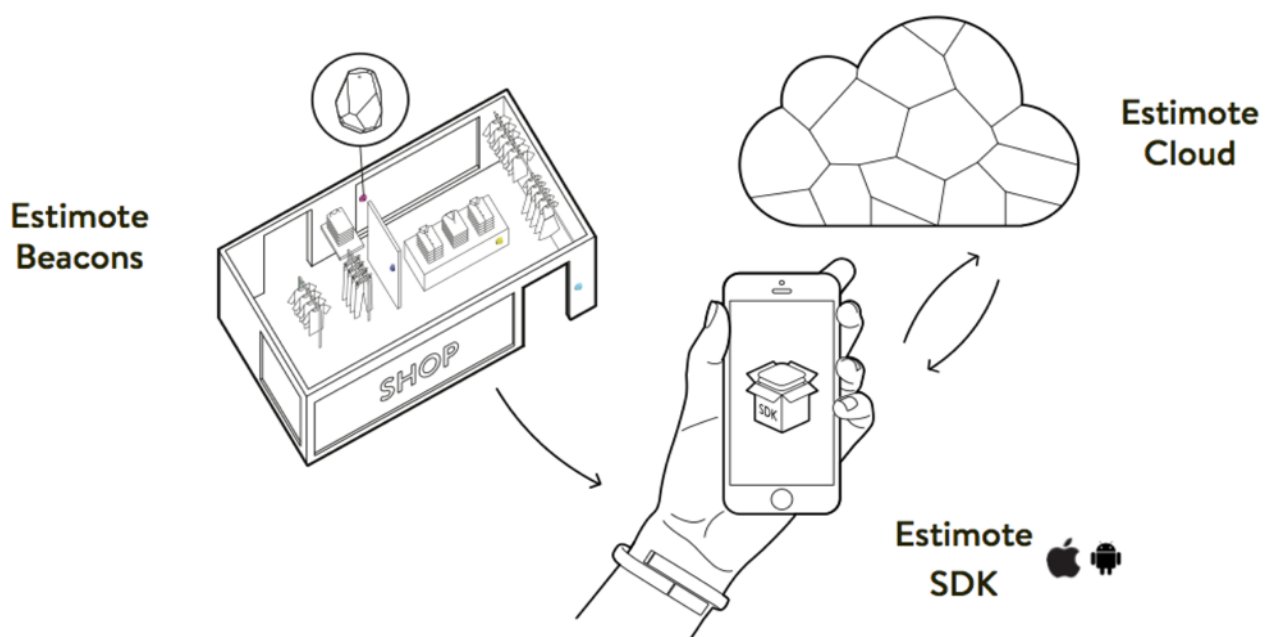
## 3.2. Estimote Beacons

In this project, we use Estimote beacons as the BLE devices. Estimote beacons and stickers are small beacons that can be attached to any location or object. By making use of the Estimote SDK, our application can know the proximity of nearby beacons and recognize their type, ownership, approximate location, temperature and motion [7].



**Figure 5:** Estimote beacons packaging

Besides the Estimote SDK, Estimote also provides another SDK especially for indoor positioning - Estimote Indoor Location SDK, which is a set of tools for building precise, blue-dot location services indoors [8]. We can upload the location's map to Estimote cloud after we set up the target location using at least 4 beacons.



**Figure 6:** Estimote SDK

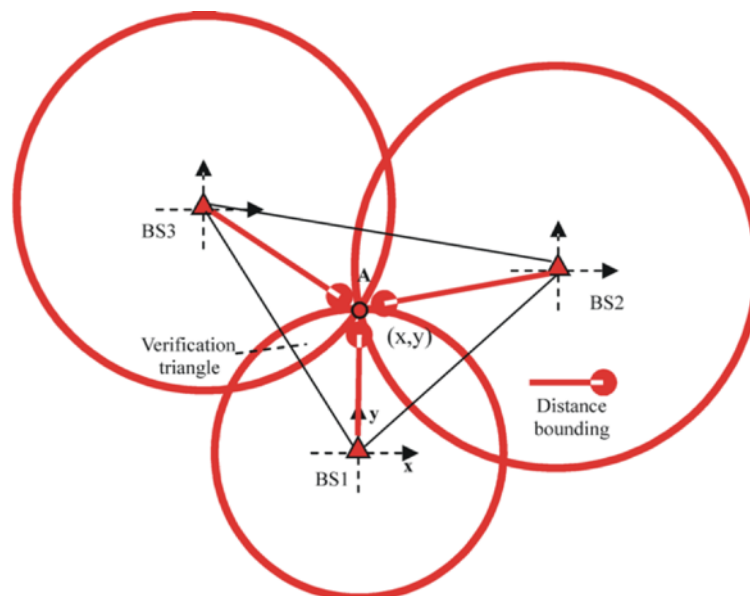
## 4. Study of Indoor Positioning

### 4.1. Approach by Analyzing RSSI

We started to do research on indoor positioning at the very beginning of our project. The first approach we did to calculate the indoor location is to analyze the received signal strength indicator (RSSI) from each beacon.

#### 4.1.1. Trilateration

Trilateration is a positioning technique using different distances from different points to determine the position of a certain point. This system has to measure the distance from the target point to at least 3 reference points whose positions are already known. Using these distances, we can form 3 circles surrounding each point. Then the only intersection of these three circles is position of the target point.



**Figure 7:** Trilateration [15]

Theoretically, we can get the position of the target point by collecting at least 3 relative distances of the reference points. As can be seen from the picture, every two circles form two intersection points, one of which is our target point. But in real situation, we know that the distance calculated from the value of RSSI may not be accurate. Thus the 3 circles may not have a common intersection. To ensure we can get a single solution, we made an optimization on this approach:

Step 1: calculate two possible position p1 and p2 by the distances from two of the reference points.

Step 2: calculate the distance d3 between the target point and a third reference point.

Step 3: get the distance d1' between p1 and the third point, d2' between p2 and the third point.

Step 4: if d1' is closer to d3 then p1 is the result position. If not then p2 is the result position.

### 4.1.2.Accuracy

As we can see, it is not difficult to get the target position from 3 distances of reference points, but it is still a problem to calculate the distance from the value of RSSI accurately. According to the researches we did, we found that the value of RSSI follows a lognormal distribution over distance [12]:

$$PL = P_{Tx_{dBm}} - P_{Rx_{dBm}} = PL_0 + 10\gamma \log_{10} \frac{d}{d_0} + X_g,$$

So the same RSSI may result in different estimated distance in indoor environment, the relation between the value of RSSI and the distance is different. So that if we really want to get an accurate value of the distance, we need to do multiple testing under the target environment which is unrealistic and time-consuming.

Finally, we gave up this approach and changed to another one.

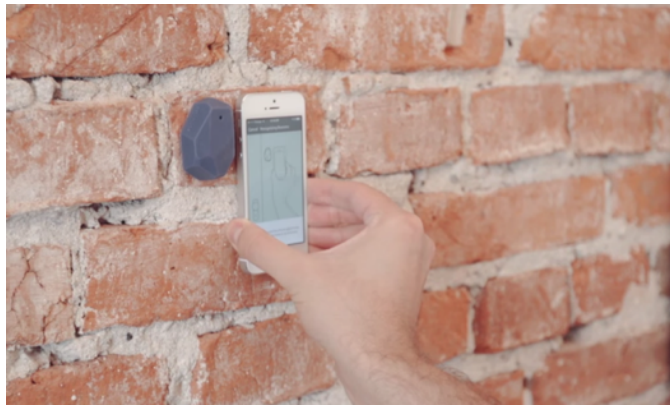
## 4.2. Approach by Using Estimote Indoor Location SDK

As we mentioned above, Estimote provides its own SDK for developers to implement indoor positioning by creating their own indoor map in Estimote cloud. Estimote Indoor Location SDK requires the following things [9]:

- 1 OS X computer with Xcode 7.
- 1 iOS device preferably newer than 5.
- 1 Estimote Account.
- At least 4 Estimote beacons related to the account.

The map creation steps are as follows [9]:

- Use the Estimote Indoor Location app to map the room into Estimote cloud and tune the location setups to make the positioning more accurate.
- Install the Estimote Indoor Location SDK through CocoaPods.
- Add Indoor Location Manager.
- Connect the app to Estimote Cloud and then fetch and update the location.



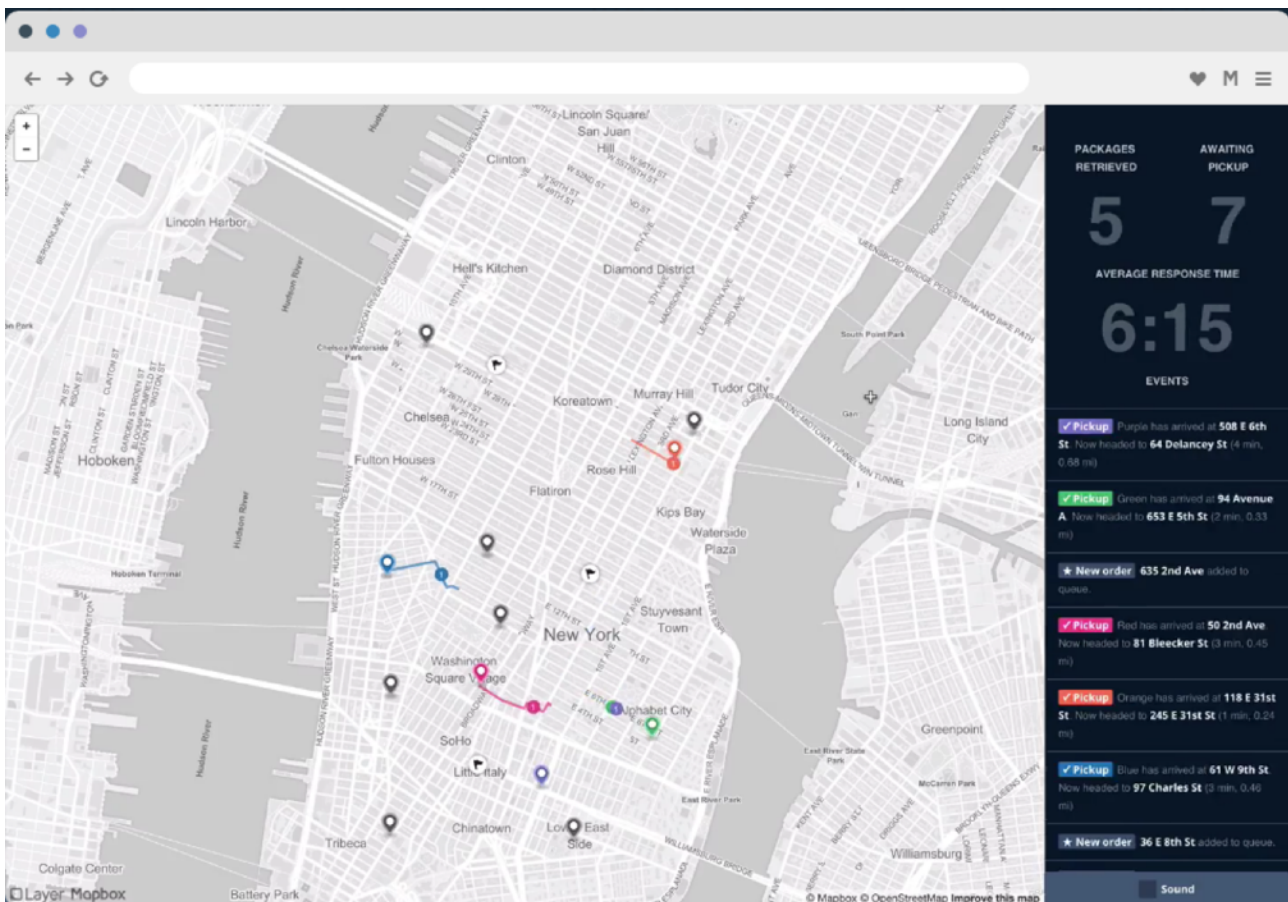
**Figure 8:** The developer is mapping his room to Estimote Cloud using the Estimote Indoor Location app. [7]

This approach requires at least 4 Estimote beacons while we just have 3 beacons until the end of this semester, so that this approach will be left to next semester's development.

## 5. Study of Custom Indoor Map Design

### 5.1. Approach by Using Mapbox

The first approach we did to implement our customized indoor map is to use an third party platform called Mapbox. Mapbox is a map platform for developers to design their own map customized with their own style and data.



**Figure 9:** Mapbox online studio interface

Basically, developers can do two things on this platform. The first one is to customize the style of the map. Developers can change the color and size of some geographical utilities, which makes the map looks different from the original one. The second one is to add their customized data like markers, lines and polygons over the map.

#### **Customized Style & Data**

Since no matter how we change the style of a map, it is still a world map with the coordinate system based on longitude and latitude. So we have two alternatives here. One is to set the

style to blank and transparent the world coordinate system to our own indoor coordinate system and then draw some data as the map of the indoor map. In another word, it is using the world as a room. Another solution is to keep the style on the map and draw the data on the exactly position of the room indicating the in door structure of it.

During implementing this two solutions, we found the shortcomings of each solution.

- **Cons of the first solution:** According to the Mapbox SDK documentation, Mapbox draws the data onto the map by rendering the provided geojson data so that when we are trying to draw an indoor map as large as the earth, it will be very CPU consuming and fail sometimes.
- **Cons of the second solution:** There are two failures of this solution. One is that it is nearly impossible to location the room accurately. When we try to render our customized data to show the indoor map, we have to provide the border coordinates of the room in latitude and longitude, in which all the points owns nearly the same value of latitude and longitude because it will be only different in 0.00001. Another problem is that even though we manage to construct the room above the world's map in the right position, the map cannot be zoomed to the level where the detail of the room is shown well.

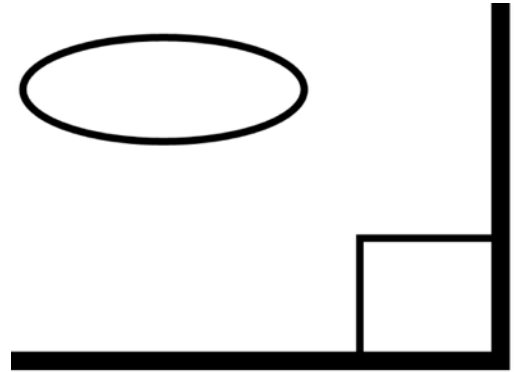
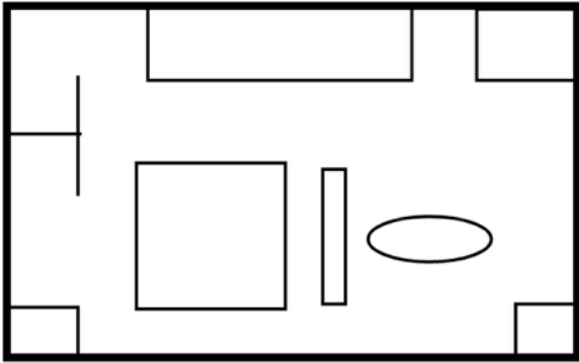
Finally we gave up this approach and chose a simple way to design our own indoor map.

## 5.2. Approach by Using a High-resolution Image

The reason why we chose Mapbox as the first approach is based on the following points:

- Map box can provide a map view through its SDK implemented with the functions like panning, zooming and rotating.
- The data we drew on the map will be shown in a vector way which means it won't change its resolution no matter how large the user zoom it.
- The map is stored in a JSON file which can be edited conveniently by just adding or deleting a few keys and values.

Based on the reasons mentioned above, we were looking for another solution which has the same advantages. At last, we chose to use a high-resolution image as the map since it can be zoomed smaller without losing its sharpness.



From the comparison between the two images above, we can see that this image remains clear after being zoomed in.

## 6. Study of Developing Backend Using Node.js

Node.js is an open-source framework for developing network applications such as server-side web application. Node.js is distinguished for its cross-platform environment and event-driven architecture. Additionally, in the framework of Node.js, developers do not have to take care of the thread-related problem since Node.js is not a thread-based framework.

### 6.1. Building RESTful APIs With Express4

Express is a module in Node.js helping developers handle routing when developing APIs. Making use of the event-driven architecture, we can implement an API in following steps:

- Create a .js file with the same name as the target router.
- Handle all the possible RESTful request to this router in this .js file by implementing the event handlers.
- Export this .js file as a module and add pass it a handler to the Express module.

Below is a simple code snippet implementing an API on router '/':

```
router.get('/', function(req, res) {  
    res.json({ message: 'hooray! welcome to our api!' });  
});
```

Up to the end of this semester, we have implemented 11 APIs over 3 routers. Below is the documentation for those APIs:

Router	API	Method	Arguments	Usage
/ticket	/redeem	POST	name: <i>ticketId</i> type: String stored in: body	To let the server check if the ticket with ticket number <i>ticketId</i> is valid.
	/getTicketId	GET	no	To generate a ticket number for testing.
	/generatePass	POST	name: <i>ticketId</i> type: String stored in: body	To let the server create, sign and return a pass according to the ticket number <i>ticketId</i> .

Router	API	Method	Arguments	Usage
	/orderHistory	GET	name: <i>deviceId</i> type: String stored in: body	To get all the tickets that has been purchased by the device with <i>deviceId</i> .
	/setUp	GET	no	To set up database of types for testing.
/ticket	/pay	POST	name: <i>stripe_token</i> type: String stored in: body  name: <i>amount</i> type: String stored in: body  name: <i>description</i> type: String stored in: body  name: <i>contact</i> type: Dictionary stored in: body  name: <i>deviceId</i> type: String stored in: body  name: <i>quantity</i> type: Integer stored in: body	To buy amount number of tickets from the server providing enough information. The server will return the client a string as ticket number if succeed.

Router	API	Method	Arguments	Usage
/item	/:uuid/:major/:minor	GET	name: <i>deviceId</i> type: String stored in: headers	To get an item object containing all the related information. The argument <i>deviceId</i> is for server to know if this device has liked one item.
	/all	GET	name: <i>deviceId</i> type: String stored in: headers	To get all the items of the museum in an array.
	/like	PUT	name: <i>deviceId</i> type: String stored in: body  name: <i>_id</i> type: String stored in: body	To indicate the server that an item with identifier <i>_id</i> is liked by a device with identifier <i>deviceId</i> .
	/share	PUT	name: <i>_id</i> type: String stored in: body	To indicate the server that an item with identifier <i>_id</i> is shared by a device with identifier <i>deviceId</i> .
	/view	PUT	name: <i>_id</i> type: String stored in: body	To indicate the server that an item with identifier <i>_id</i> is viewed by a device with identifier <i>deviceId</i> .

## 6.2. Using MongoDB to Save Data

Different from the traditional SQL database, MongoDB is a NoSql database which is implemented in a document-oriented way. There are several reasons why we chose MongoDB to store our data instead of the traditional SQL databases:

- MongoDB is perfectly integrated with Node.js. We can add MongoDB into our modules by adding just one line of code in file package.json.
- MongoDB stores data in document with JSON style which is very readable and editable to developers. JSON style data is also easy to use for client side.

- Developers do not have to set the property like primary key or default value when creating a schema for a table. The only thing developers have to do is to set the index name and the value type of each column.
- Updating data becomes easier. We can update a row of data by calling the method 'save()'.
- We do not have to care about if one of the column value is not given when adding data into each row of the table.

We use MongoDB to store our item data and order data, and here is the schema of these two table.

```
var ItemSchema = new Schema({
  beaconUUID: String,
  beaconMajor: String,
  beaconMinor: String,
  title: String,
  coverImage: String,
  author: String,
  country: String,
  description: String,
  soundtrack: String,
  images: Array,
  coordinateX: Number,
  coordinateY: Number,
  introduceTime: Date,
  inExhibition: Boolean,
  views: [String],
  likes: [String],
  shareCount: Number
});
```

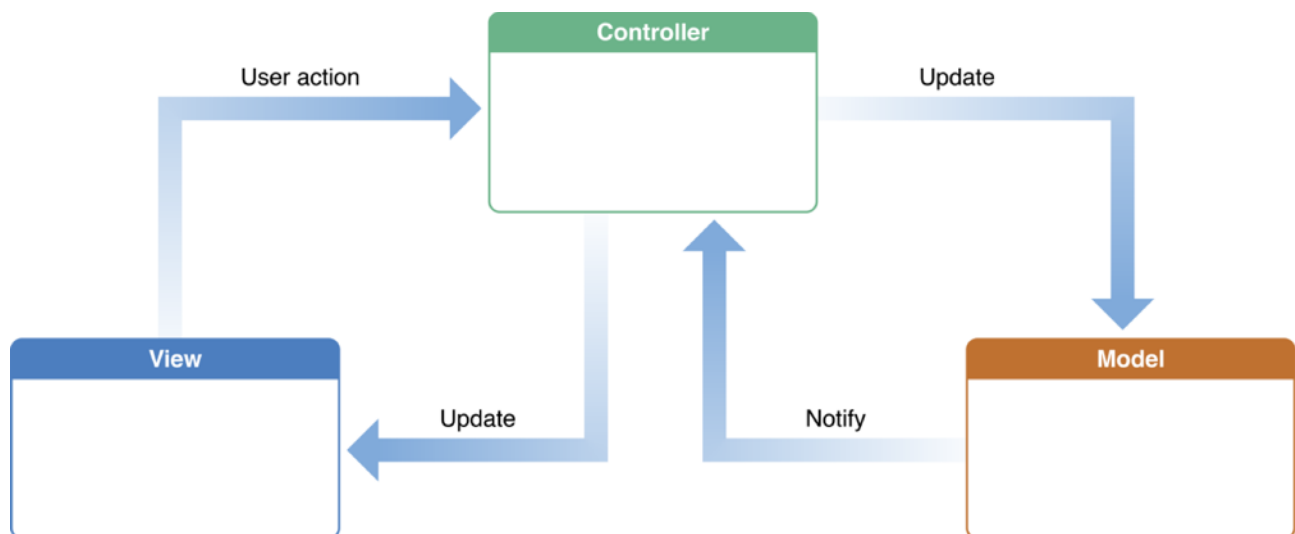
```
var OrderSchema = new Schema({
  email: String,
  amount: Number,
  purchaser: String,
  generateDate: Number,
  redeemedDate: Number,
  generateDateObject: Date,
  redeemedDateObject: Date,
  ticketId: String,
  valid: Boolean,
  deviceId: String
});
```

## 7. Developing Application With Swift

### 7.1. Model-View-Controller

We designed our application in a Model-View-Controller(MVC) pattern which is to separate objects into three roles: Model, View and Controller. Model objects is the representation of data. Views are what users can see and interact with. Controller objects will be in charge of updating model and view according to user's actions. MVC is a good pattern to design our application for the following reasons.

- It makes most of our objects reusable. The interfaces between objects are well defined.
- Our application will become more extendable since it would be relatively easy to add new features by add a set of Model-View-Controller.
- We can use many Cocoa technologies that are based on MVC pattern.



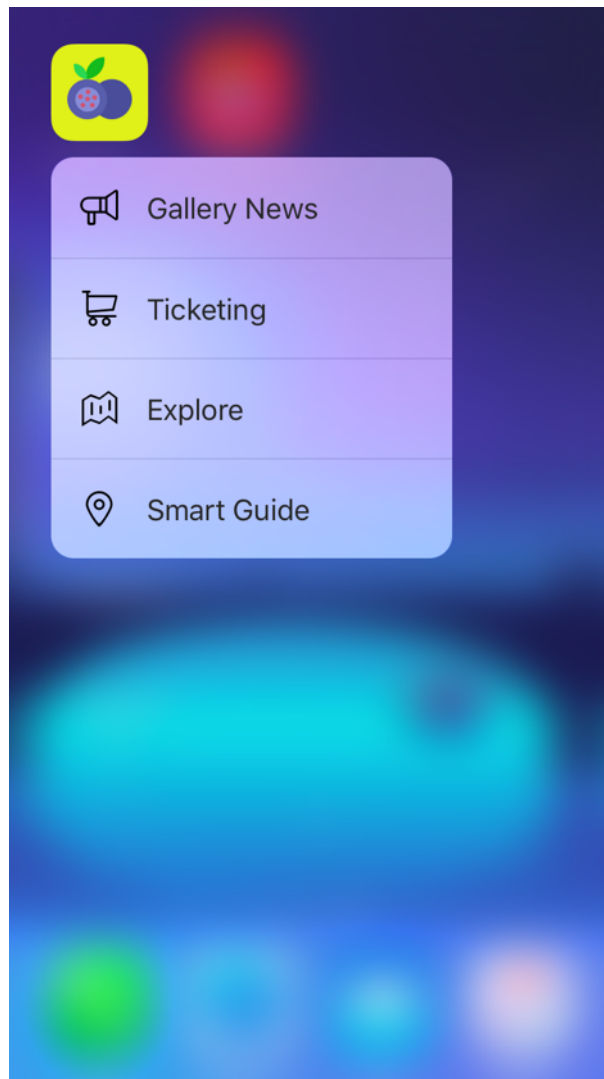
**Figure 10:** MVC illustration [10]

### 7.2. 3D Touch

3D touch is a new technology Apple introduced this year together with iPhone 6s / 6s plus and iOS 9. This technology allows iPhone 6s / 6s plus to know how hard a user is pressing the screen. Integrating with 3D Touch technology can make users have more actions over our application.

### 7.2.1.Home Screen Quick Actions

The traditional way to launch an application is by tap it on the home screen. With the help of 3D Touch, users can have quick actions on our application by pressing our icon in the home screen and select a quick way to launch our application, which can anticipate and accelerate a user's interaction with our application.

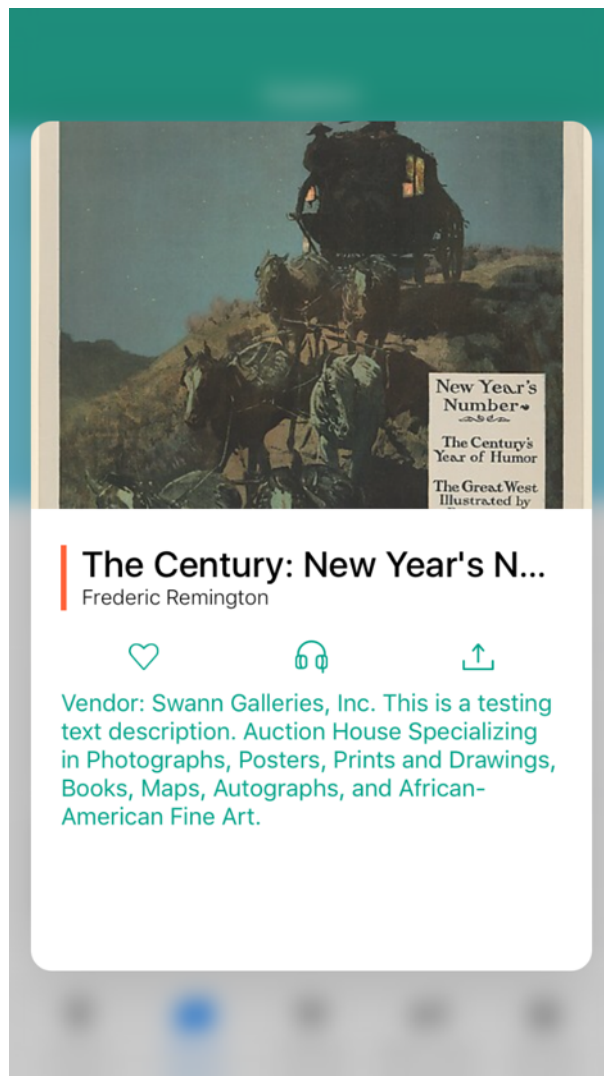


**Figure 11:** Quick Actions

## 7.2.2. Peek and Pop

“Peek and Pop” is a new way for users to interact with iOS device supporting 3D Touch. It allows users to take a peek or have a preview of the view, which is going to be opened.

We can make our application work in a more efficient way by implement this feature. From our perspective, we think a user should not spend too much time on operating his iPhone when he or she is visiting a museum. When a user is interested in one of the items in the item list, he or she can decide whether to look into detail of it by having a peek. This can make a user save more time and get exactly what he or she needs from our application.



**Figure 12:** Peek and Pop

## 7.3. Centralized Soundtrack Player

Our application has a function to play the item introduction soundtrack. Since we are not allowed to play multiple soundtracks at the same time, we need only one instance of the player object to do the playing job.

We implemented a singleton player inside SoundtrackPlayer class which can only be instantiated once. Although the player can be seen in every tab of our application, it is only controlled by the methods in class AppDelegate, which is the heart of the application.

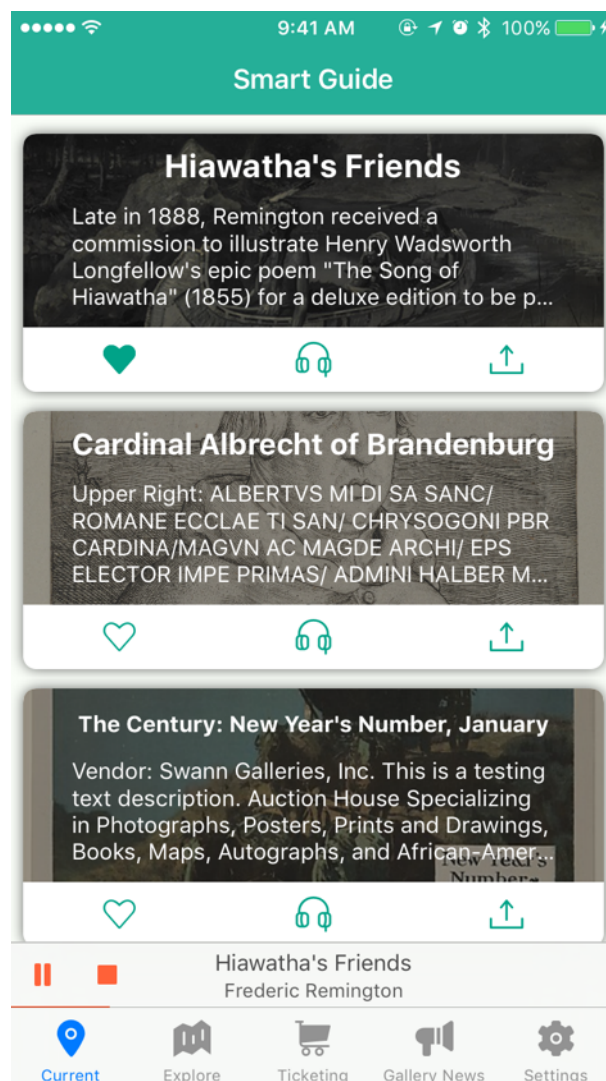


Figure 13: Centralized player

## 8. Google Maps

In the second term, we changed our direction from indoor navigation in one single room to multi-functional navigation of an area with a collection of items. We migrated to Google Maps SDK for implementing new features because of its rich data set and easiness to integrate.

### 8.1. Google Maps SDK for iOS

Google Maps SDK for iOS [21] provides various services, such as handling user gestures, showing customized markers and their corresponding information windows, etc. Google Maps SDK automatically handles access to the Google Maps servers for more advanced features.

To start using Google Maps SDK for iOS, we first need to create a project in Google developer console. By registering in the console, access keys need to be generated to allow using its services from iOS devices or web browsers.

### 8.2. Map View Animations

The Google Maps SDK for iOS represents the real world map on the device's flat screen. Locations of objects are represented using latitude and longitude. The map view is modeled as a camera looking down on a flat plane. The camera view can be controlled by several properties: location, zoom level, bearing, viewing angle.

By modifying zoom level, bearing and viewing angle within an UIView animation block, one can create a nice animation of initiating the map views for users while loading data from the server. This creates a more friendly user interaction flow.

When initiating the map view, the map is centered at CUHK campus (latitude: 22.4215, longitude: 114.2078) with zoom level 16, 120 degree bearing and 0 degree viewing angle. Then the map view will animate its camera to 0 degree bearing and 80 degree viewing angle. The effects are from looking down to a more natural angle, indicating the services are ready to use.

When a walking direction is calculated and needs to be shown on the map, the camera will be animated to zoom level so that it just contains the origin location and destination location. The bearing angle stays at 0 degree but viewing angle changed to 0 degree to let the user have a

better view of the overview route. All these changes are completed in 1 second of animation with spring effect.

### 8.3. Marker customization

The markers of the map can be customized using a simple image or a UIView. Our approach is using a pre-designed image to customize the markers to be more identifiable and unique.



**Figure 14:** Customized marker

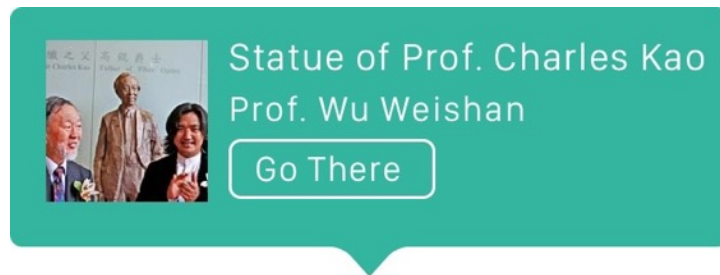
Other customizations include changing the appearing animation to “pop”, making the marker tracks information windows changes. Each marker represents an actual item on the map and is associated to the information of the item.

The marker configuration code is as below for each item:

```
let marker = GMSMarker()
marker.position = loc
marker.appearAnimation = kGMSMarkerAnimationPop
marker.icon = UIImage(named: "map_marker")
marker.map = self.mapView
marker.tracksInfoWindowChanges = true
marker.userData = item
```

### 8.4. Info Window

An info window is used to display information to the users when they tap on a marker. If a user tap on another marker while current info window is showing, the current info window will be closed and then new info window will appear. That means only one info window can be displayed at a time.



**Figure 15:** Customized info window

The default info window contains only a title and a description. To make the design consistent with other elements and also to serve more intuitive information, we also re-designed the info window.

The customized info window consists of a title, creator name, a thumbnail image and a “Go There” button. This info window gives users direct but simple information about the item represented by that marker.

## 8.5. Google Maps Directions API

In order to provide navigation service for a user to go to a particular item, we integrated Google Maps Directions API [22] in our application.

### 8.5.1. Request format

This API is using an HTTP request for calculating directions between two locations. The request takes the following form:

```
https://maps.googleapis.com/maps/api/directions/output?parameters
```

The “output” in the request is for specifying the format of the return object. It can be either JSON or XML. In our application, we chose JSON as the return format.

The “parameters” part is for setting parameters for the calculation. Some of the parameters are required and some are optional depending on the needs of the application. The parameters being used in Eksibition are as follows:

- origin - the user’s current location in format of [latitude], [longitude].
- destination - the destination’s location in format of [latitude], [longitude].

- key - the access key generated in Google developer console. This key is required for using the API.
- mode - the mode of transport to use when calculating directions. In this app, we specify the mode as “walking” always because it’s for navigation inside campus.

With that being explained, an example of a full request sent to Google server would be like this:

```
https://maps.googleapis.com/maps/api/directions/json?
origin=22.4180729,114.2085763&mode=walking&destination= 22.419873,
114.204381&key=MY_API_KEY
```

### 8.5.2. Directions Responses

As specified in the request, the response will be in JSON format. If all of the parameters sent to Google Maps Server are valid, the response data will contain a direction information in detail.

Firstly, there is a “status” key in the JSON indicating whether the calculation is successful. A detailed transport steps are in “routes” property. There may be more than one route being returned as alternative routes to the fastest one.

In a single route, there are information including map bounds, starting address, ending address, etc. The most important information are steps under property “legs”. “legs” contains the accurate traveling pathways. The information we needed here are distance and estimated traveling duration.

A complete response example JSON is as follows:

```

{
  "geocoded_waypoints": [
    {
      "geocoder_status": "OK",
      "place_id": "ChIJlWtGnZ4IBDQRgDJB8rsq1k0",
      "types": [
        "route"
      ]
    },
    {
      "geocoder_status": "OK",
      "place_id": "ChIJH4l6uZwIBDQRZfqTXBkEOQY",
      "types": [
        "route"
      ]
    }
  ],
  "routes": [
    {
      "bounds": {
        "northeast": {
          "lat": 22.4198731,
          "lng": 114.2085304
        },
        "southwest": {
          "lat": 22.417021,
          "lng": 114.2043444
        }
      },
      "copyrights": "Map data ©2016 Google",
      "legs": [
        {
          "distance": {
            "text": "0.8 km",
            "value": 768
          },
          "duration": {
            "text": "14 mins",
            "value": 868
          },
          "end_address": "Library Blvd, Ma Liu Shui, Hong Kong",
          "end_location": {
            "lat": 22.4198731,
            "lng": 114.2043779
          },
          "start_address": "Nursery Path, Ma Liu Shui, Hong Kong",
          "start_location": {
            "lat": 22.4182063,
            "lng": 114.2085304
          },
          "steps": [...],
          "via_waypoint": []
        }
      ],
      "overview_polyline": {
        "points": "ypygCijaxTPv@JLZD`@VN?ZSh@CCNBJf@h@N^B\

```

```

    },
    "summary": "University Ave",
    "warnings": [
        "Walking directions are in beta. Use caution – This route may be missing sidewalks or pedestrian paths."
    ],
    "waypoint_order": []
  }
],
"status": "OK"
}

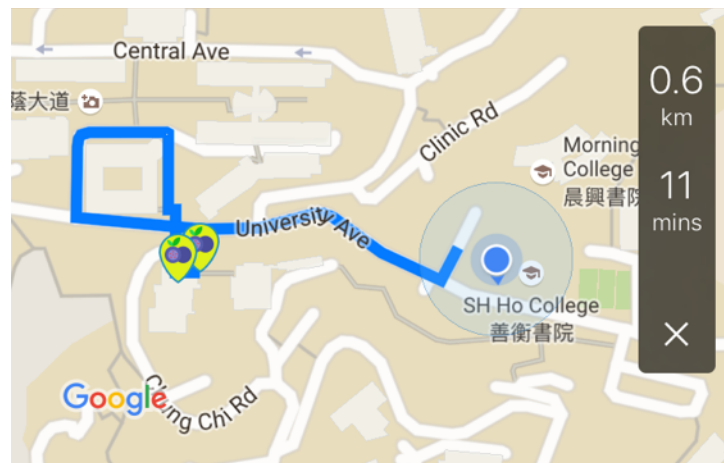
```

To draw a path on the map, we need the “overview\_polyline” property because it is an encoded overview path from origin to destination. Google Maps SDK for iOS already contains method for creating path directly from an encoded path string. We only need to create an path object using the encoded path returned by Google Maps API, and then set the path’s map property to the current map view:

```

self.mapPolyLine.path = GMSPath(fromEncodedPath: encodedPath)
self.mapPolyLine.map = self.mapView

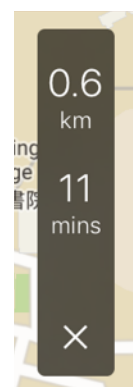
```



**Figure 16:** Displaying directions

## 8.6. Navigation Information Panel

When navigation information are shown, the navigation information panel will appear on the right side of the map taking 50 pt space. The purpose of this information panel is to intuitively let users know the distance between current location and destination location, and the estimated walking duration.



A close button is at the bottom. When tapped, the map will quit navigation mode and the navigation information panel will be hidden.

## 9. Apple Pay

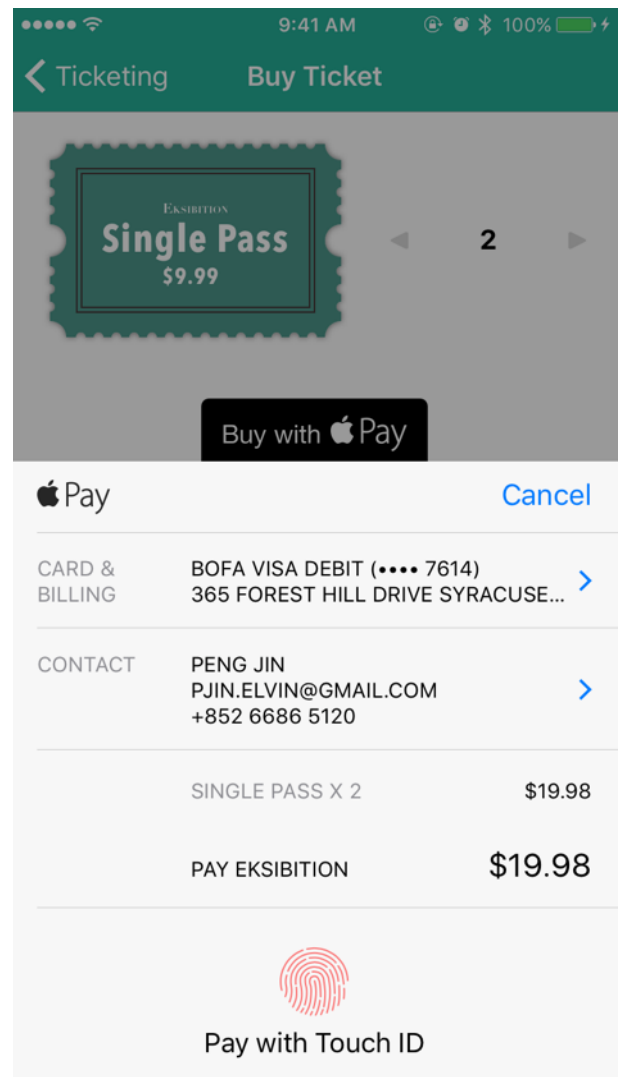
Apple first introduced Apple Pay in its conference in September 2014. Tim Cook, the CEO of Apple, illustrated that the tradition payment system like loading a card with a magnetic stripe and verifying a credit card by a combination of several numbers are not that safe. Apple Pay will be the new generation's payment system that can change the whole payment system. Basically, if a user want to use Apple Pay, he or she has to first add his card into his Apple Wallet. After that, the user can authorize a payment by just verifying his Touch ID through Apple Pay in all the retail stores and online stores supporting Apple Pay.

For developers who want to integrate Apple Pay with their applications, there are following prerequisites:

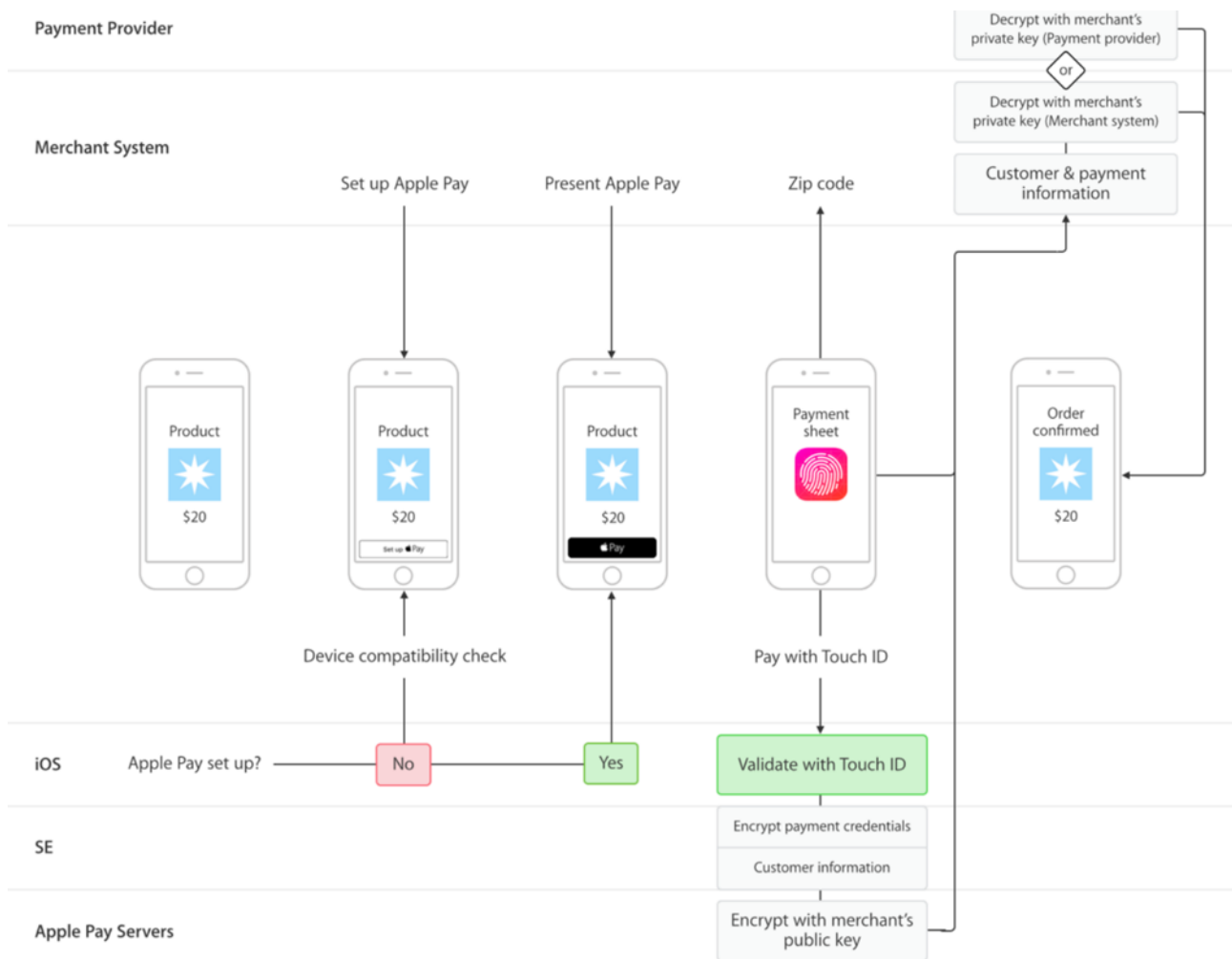
- A developer account with payment processor.
- A valid Merchant Identifier.
- A public key and a private key for encryption and decryption.
- Apple Pay entitlement being included in the application.

### 9.1. Workflow of Apple Pay

Below is a diagram showing the workflow of Apple Pay in one purchase.



**Figure 17:** Apple Pay



**Figure 18:** Apple Pay workflow [11]

## 9.2. Client-side

What we did in the client-side:

- Setting up the project to support Apple Pay by importing Apple Pay developer kit.
- Getting the information of billing, cost, shipping address and others.
- Generating an encrypted token representing this transaction.
- Sending the token to the server and handle the callback.

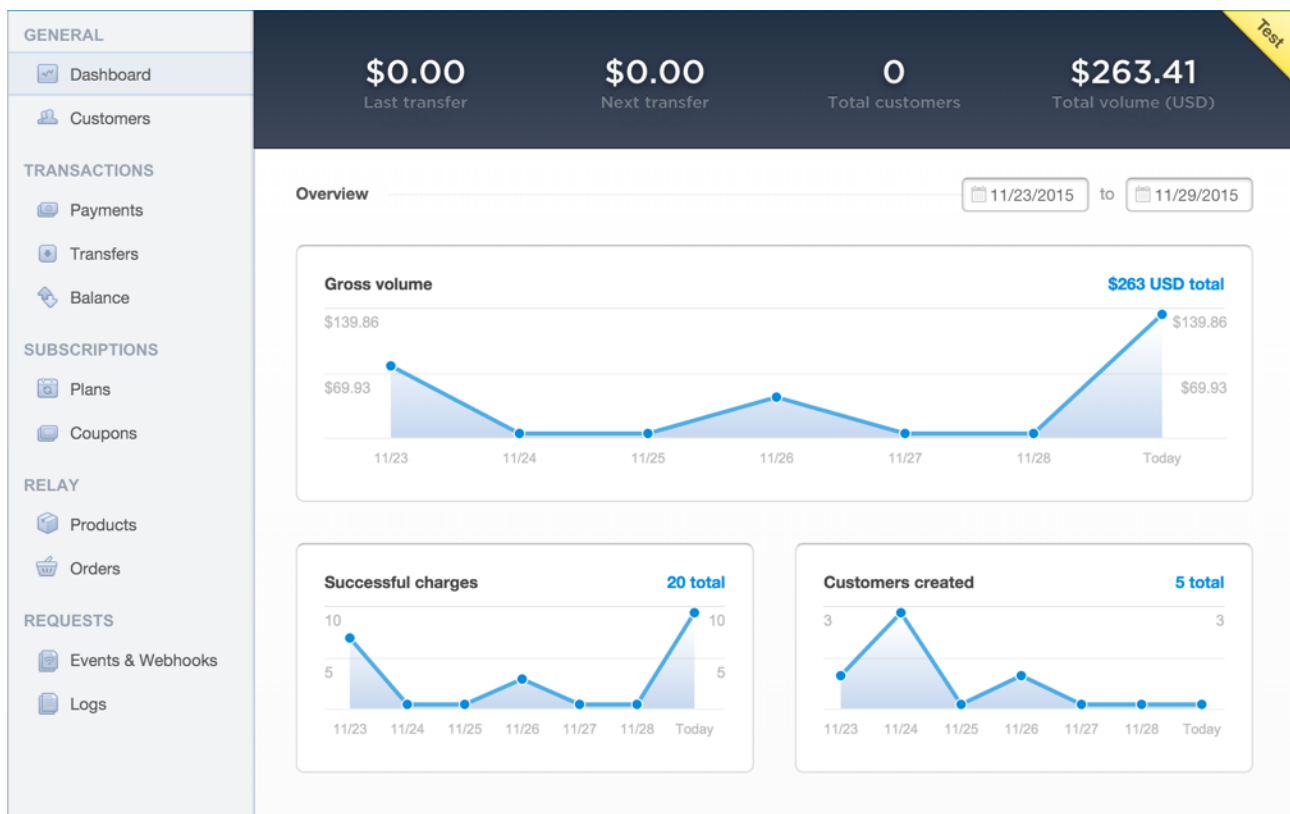
## 9.3. Server-side

For the server-side, we did the following things:

- Creating an APIs to handle the request from client, which carries the token and other transaction information.
- Decrypting the token and getting the payment credentials.
- Making the charge and send the result of it to the client.

## 9.4. Stripe

Stripe is a third party payment gateway service provider, who provides a easy platform for business owners to charge clients more easily. By integrating Stripe SDK, implementing the payment process with Apple Pay becomes much easier. Here is the steps we had gone through when integrating Stripe:



**Figure 19:** Stripe dashboard

- Install the Stripe library on both client-side and server-side.
- Get the public API key from Stripe and configure it on both client-side and server-side.
- Get the information of the purchaser from Apple Pay.

After these 3 steps, we can use methods in Stripe library to finish transaction. Here, Stripe speed up the whole development process by handling all encryption, decryption and charging with the banks for the developers.

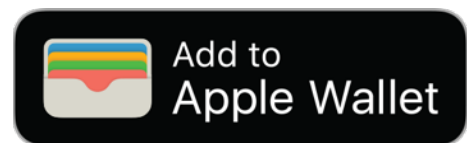
Balance history				
\$28.80 USD \$1.17	Charge	<a href="#">Charges from elvinjin8@gmail.c...</a>	— \$29.97 -	2015/11/29 18:21:12 >
\$9.40 USD	Charge	<a href="#">Charges from elvinjin8@gmail.c...</a>	— \$9.99 - \$0.59	2015/11/29 17:36:04 >
\$9.40 USD	Charge	<a href="#">Charges from elvinjin8@gmail.c...</a>	— \$9.99 - \$0.59	2015/11/29 15:22:40 >
\$9.40 USD	Charge	<a href="#">Charges from elvinjin8@gmail.c...</a>	— \$9.99 - \$0.59	2015/11/29 14:29:31 >
\$28.80 USD \$1.17	Charge	<a href="#">Charges from elvinjin8@gmail.c...</a>	— \$29.97 -	2015/11/29 13:50:33 >
View balance history				>

**Figure 20:** Stripe transaction history

## 10. Apple Wallet

Apple Wallet is an application introduced in iOS 8.1. Before iOS 8.1, it is called Passbook, which allows user to store their passes, like coupons, tickets and boarding passes. Apple introduced a new generation of Passbook named Apple Wallet together with Apple Pay in this year's conference. Users can add his own credit cards, debit cards and loyalty cards into his Apple Wallet.

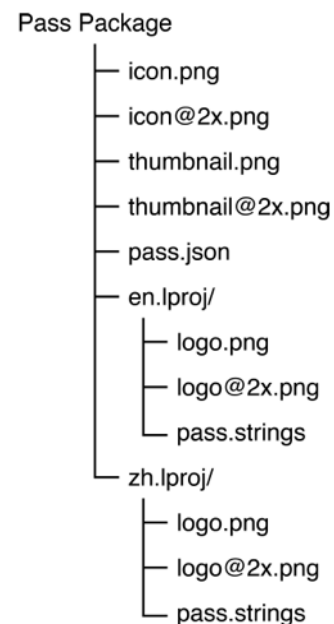
Since we have finished integrating Apple Pay in our application, users can buy tickets right within our application using Apple Pay. What we are going to do now is to enable users to add purchased tickets into Apple Wallet so that users can redeem their tickets by just showing the passes in their Apple Wallet.



**Figure 21:** "Add to Apple Wallet" badge



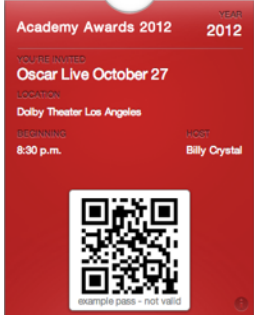
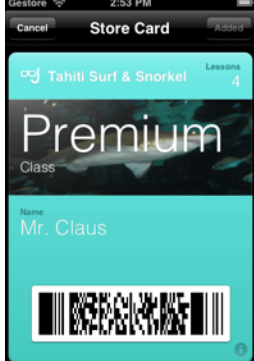

### 10.1.Turning a Ticket into a Pass

Before trying to add a ticket into Apple Wallet as a pass, we have to learn what exactly a pass is. From a user's perspective, a pass is a collection of all the relevant information of an event. From a developer's perspective, a pass is a package containing a JSON file as the template, which stores all the information and all the image resources that are required to build the pass.



**Figure 22:** Pass structure

There are 5 different styles of passes:

Pass Style	Description	Example
<b>Boarding Pass</b>	Representing a ticket to get discount when buying goods in the certain store.	 A boarding pass for Blue Airways, flight BX 651 from RIX to LGW. It includes passenger name Jane Smith, membership Gold, and a QR code.
<b>Coupon</b>	Representing a ticket to get discount when buying goods in the certain store.	 A Subway coupon for a free can of soda with any 6" or 12" sandwich. It includes an expiration date of 20 Dec, 2012 and a QR code.
<b>Event Ticket</b>	Represent a ticket to get access to an event at a particular time and venue.	 An event ticket for the Academy Awards 2012, Oscar Live October 27 at the Dolby Theater Los Angeles. It includes the start time 8:30 p.m. and host Billy Crystal, along with a QR code.
<b>Store Card</b>	Representing the user's account in a store or club.	 A store card for Tahiti Surf & Snorkel, Premium Class. It includes the name Mr. Claus and a QR code.
	If the purpose of the pass is not among the above 4 categories.	 A PassKit membership card for Percy PassKit. It includes a photo, membership number 12345678, points 888, status VIP, and a QR code.

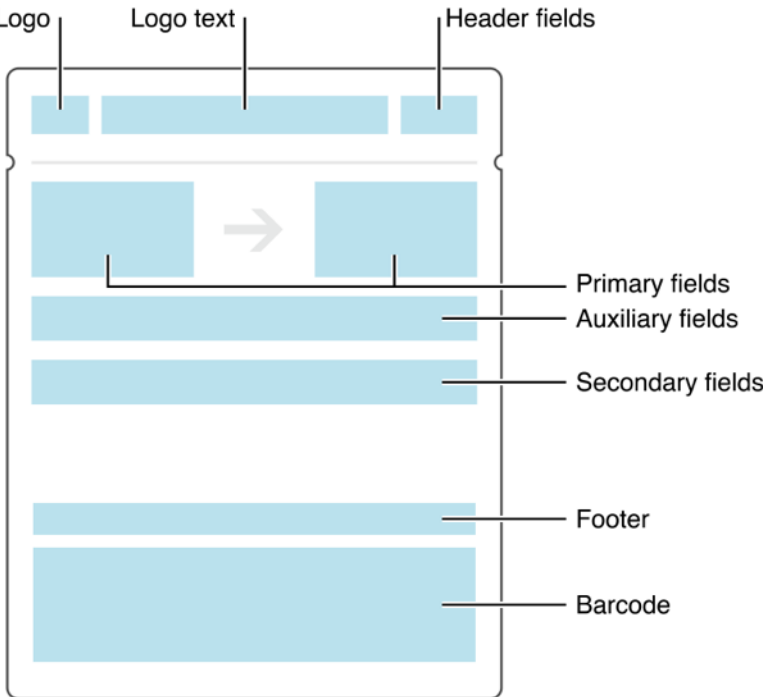
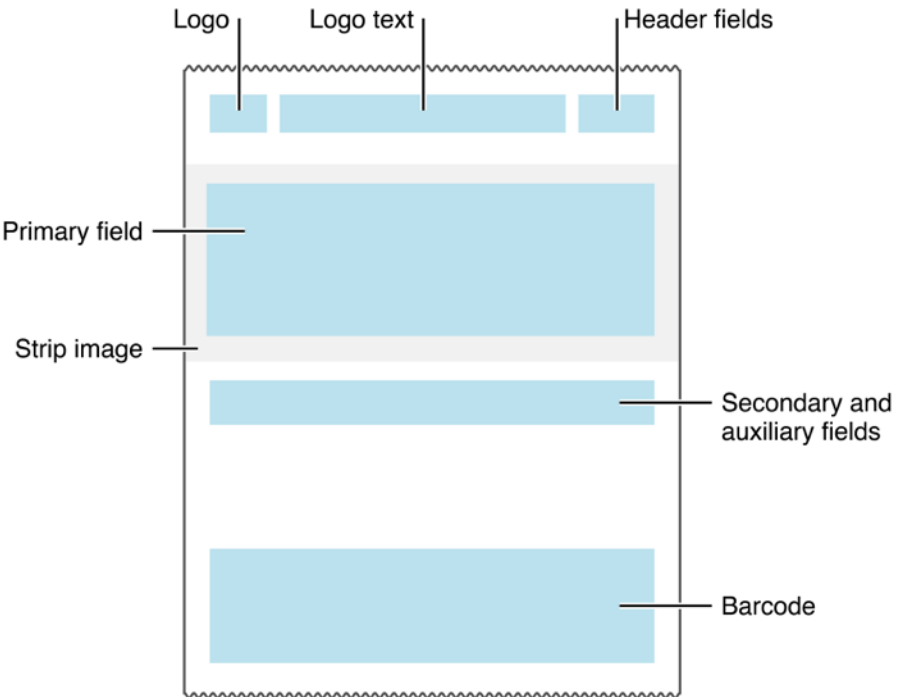
The JSON file in the pass package is for specifying the information and the style of the pass which is in a format like this:

```
{
  "formatVersion" : 1,
  "passTypeIdentifier" : "xxx",
  "serialNumber" : "xxx",
  "teamIdentifier" : "HP634A2CZK",
  "organizationName" : "Eksibition Team",
  "description" : "CUHK Museum 1-Day Pass",
  "foregroundColor" : "rgb(255, 255, 255)",
  "backgroundColor" : "rgb(60, 65, 76)",
  "labelColor" : "rgb(177, 189, 213)",
  "eventTicket" : {...}
}
```

Taking the above code as an example, the field “eventTicket” will contain all the relevant information shows on the front page of the pass. Below is a sample of a event ticket.

```
"eventTicket" : {
  "headerFields" : [{
    "dateStyle" : "PKDateStyleMedium",
    "key" : "valid_from",
    "label" : "Valid From",
    "value" : validFrom.format("YYYY-MM-DDThh:mmZ")
  }],
  "primaryFields" : [{
    "key" : "event",
    "label" : "EVENT",
    "value" : "Paintings by CUHK Students"
  }],
  "secondaryFields" : [{
    "key" : "loc",
    "label" : "LOCATION",
    "value" : "SHB 611, CUHK"
  }],
  {
    "key" : "ticket_type",
    "label" : "Ticket Type",
    "value" : "1-Day Pass"
  },
  {
    "isRelative" : true,
    "key" : "doors-open",
    "label" : "Doors Open",
    "timeStyle" : "PKDateStyleShort",
    "value" : "2015-12-10T09:30+08:00"
  }
]
```

For different styles of pass, the layout of this field is different [16].

Pass Style	Layout
Boarding Pass	 <p>The diagram illustrates the layout of a Boarding Pass. At the top, there are three header fields: 'Logo', 'Logo text', and 'Header fields'. Below these, the layout is divided into several sections: 'Primary fields' (containing two large blue rectangles with a right-pointing arrow between them), 'Auxiliary fields' (a single blue rectangle), 'Secondary fields' (a single blue rectangle), 'Footer' (a single blue rectangle), and 'Barcode' (a large blue rectangle at the bottom). Brackets and lines connect the labels to their respective fields.</p>
Coupon	 <p>The diagram illustrates the layout of a Coupon. At the top, there are three header fields: 'Logo', 'Logo text', and 'Header fields'. Below these, the layout is divided into several sections: 'Primary field' (a large blue rectangle), 'Strip image' (a horizontal line with a wavy pattern), 'Secondary and auxiliary fields' (a single blue rectangle), and 'Barcode' (a large blue rectangle at the bottom). Brackets and lines connect the labels to their respective fields.</p>

Pass Style	Layout
<b>Event Ticket</b>	<p><b>With background image</b></p> <p>Logo, Logo text, Header fields, Thumbnail, Primary field, Strip image, Secondary fields, Auxiliary fields, Background image, Barcode</p> <p><b>With strip image</b></p> <p>Logo, Logo text, Header fields, Thumbnail, Primary field, Strip image, Secondary fields, Auxiliary fields, Barcode</p>
<b>Store Card</b>	<p>Logo, Logo text, Header fields, Primary field, Strip, Secondary and auxiliary fields, Barcode</p>
<b>Generic</b>	<p><b>With rectangular barcode</b></p> <p>Logo, Logo text, Header fields, Primary field, Thumbnail, Secondary and auxiliary fields, Secondary fields, Auxiliary fields, Square barcode, Rectangular barcode</p> <p><b>With square barcode</b></p> <p>Logo, Logo text, Header fields, Primary field, Thumbnail, Secondary and auxiliary fields, Square barcode</p>

After specifying all these fields in the JSON file and adding all the relevant images in the package, we still have to sign our pass package using the registered Apple Developer private key. We have go through the following steps to be able to sign a pass [13]:

- Go to iOS Developer Portal and add a pass into the pass list of our account.
- Download the certificate from the configuration of this pass and then add the certificate to our Keychain.
- Download the Apple Worldwide Developer Relations Certification Authority (WWDR) certificate and add it to the Keychain.
- Sign the pass package with these two certificates.

## 10.2.Sign and Distribute Passes

Since the key is private, we can't just have a static pass embedded in the application. We have to sign every single pass on our server whenever the client side request a pass. So we improved our server by adding a new API for signing and distributing passes according to the requests from the client-side.

We use a module called "passbook" in npm to help us do this. The module passbook requires the two certificates in format .pem. So that we have to convert the certificated to the right format by using the command provided by passbook.

```
node-passbook prepare-keys -p keys
```

After this, we went through the following steps to sign and distribute a pass of ticket from server according to the request from the client-side.

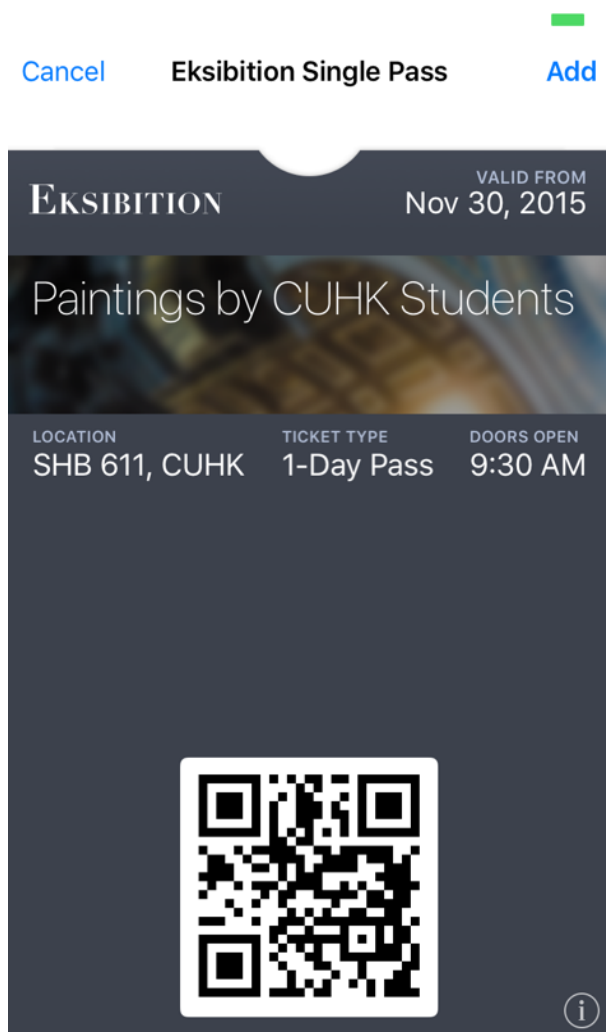
- Get the ticket number from the parameters in the request and get the related information about this ticket from the database providing the ticket number.
- Create a template for this ticket and update this template according to the information got from the database.
- Create a pass package with the template we got and sign this pass package with the two certificates.
- Send the package to the client and delete the local file of it.

## 11. Redeem System

After implemented the function that a user can add his or her tickets into Apple Wallet, we still need a redeem system for the museum organizer to check if the ticket provided by the user is valid or not. So we designed a redeem system and built another application for museum organizers to check the tickets.

As we can see from Figure 18, each pass contains a QR code at the bottom. The museum can use our redeemer application to scan this QR code to check the validity of the ticket.

The redeemer application will get a ticket number from the QR code and send it to the server. The server will check if this tick number is in the right format and if it is still valid. The server will give the redeemer application a response indicating the validity of the ticket number.



**Figure 23:** Sample Pass



Welcome Peng Jin

**Figure 24:** Redeem a ticket

## 12. News System

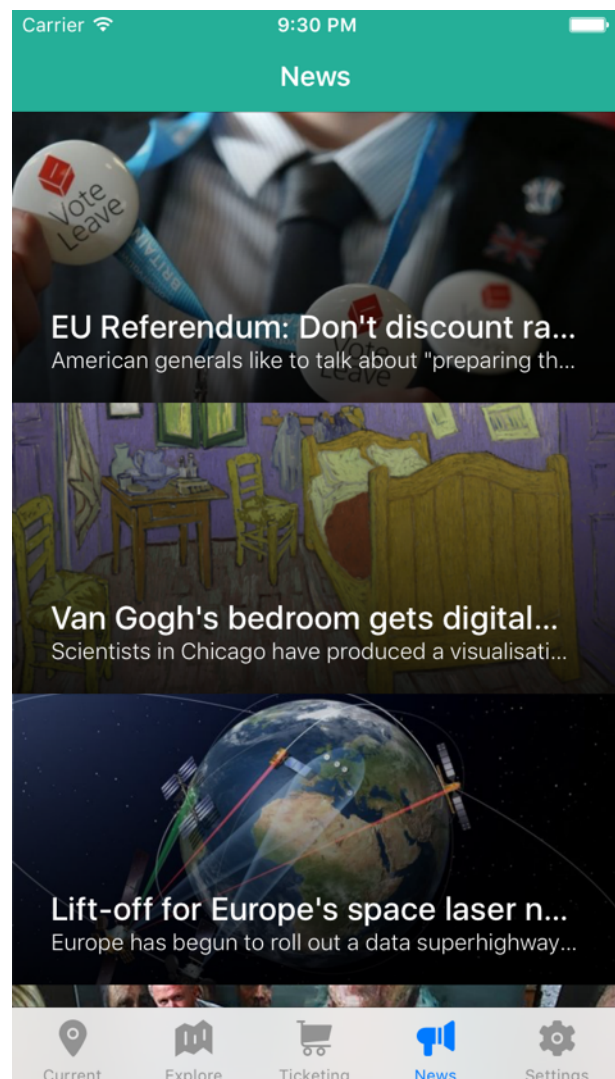
Event organizers often want to promote activities to attract more visitors, so we designed a News center just for that purpose. In the 4th tab of our app, we added a news center. Users can access all of the promotions posted by organizers here. It also takes advantage of iOS Core Data to allow offline usage. When the app is in the iOS background or the phone is locked, event organizers can send push notifications to these devices and the users will never miss a thing if they enable this feature.

### 12.1.UI Design

#### 12.1.1.News List

The basic element of the News tab is a news cell. A news cell represents a single news with titles and content preview.

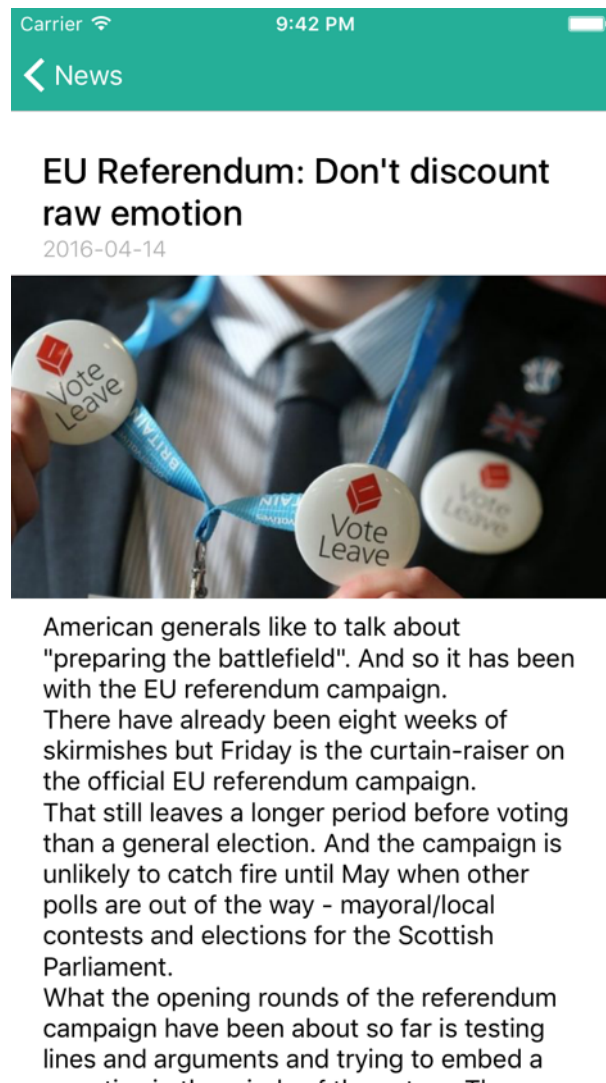
There are 4 layers of the cell view. First layer is the UITableViewCell layer, which is the container of the cell. The 2nd layer is the news head image. On top of the image is a half transparent gradient layer with black tint color. The usage of this layer is for the ease of reading titles and content previews so that no matter what color the image is, the text are always clear. The top most layer is the text layer, which gives a quick peek of the news content.



**Figure 25:** News Tab

## 12.1.2. News View

The detail view of a single news consists of a title, post time, a head image and content text. When creating new news on our web administration platform, organizer has to follow this content format to successfully create news.



**Figure 26:** News detail view

## 12.2. Study of Apple Push Notification Services (APNs)

Apple Push Notification Services (APNs) is the core of Apple's remote notification features. It is an efficient and robust service for propagating information to Apple devices. Each of those devices can have an authorized and encrypted IP connection with APNs and can receive notifications through this constant connection. If one device receives a notification while the target application is not running, the device will have an alert telling the user that there is some data waiting for the application to receive [17].

We need use our own server to create the remote notifications to our users. In the process of sending the notification, our server is called the provider, which decide when to sent a notification and what data to be sent together with the notification. When we wants to send a notification, we have to generate a payload of the notification and send it to the APNs through a persistent and secure channel using HTTP/2 protocol. APNs will handle the rest of the procedures of delivering the notification to the devices.

### 12.2.1.The Path of a Remote Notification

Every time the application is opened on a device, the server will receive the latest device token. This is an unique string generated by Apple which is used to identify this device during a certain period. When the provider want to sent a notification, it has to send the target device tokens together with the notification payload.

The notification payload is just a simple JSON dictionary which contains the data we want to send to our users, the information about the way to notify the users and some other customized data. Below is a table showing the usage of the keys in the payload JSON dictionary: [18]

Dictionary	Key	Value Type	Remark
aps	alert	string/dictionary	The system will display an user configured alert or a banner is this property is included. If this aps is a dictionary, the description of the keys may refer to the next table.
	badge	number	The red number showed on the left-upper corner of the icon. If this property is not included, the red number will not change. If this property is set to 0, the red number will be removed.
	sound	string	The name of the sound file you want to be played when users receive the notification. If 'default' is set to this property, the default sound will be played.

Dictionary	Key	Value Type	Remark
	content-available	number	Assigning 1 to this property indicating that there is some new content available for the application.
	category	string	This is a property identifying custom action defined in client side.
alert	title	string	
	body	string	
	*-key, *-action	string, array, null...	Some other keys that can be customized for other usages.

Here is an example of the payload:

```
{
  "aps" : {
    "category" : "NEW_MESSAGE_CATEGORY"
    "alert" : {
      "body" : "This is the body of the alert.",
      "action-loc-key" : "VIEW"
    },
    "badge" : 3,
  },
  "acme-account" : "xxx.alert@gmail.com",
  "acme-message" : "message123456"
}
```

Here are two figures showing the path of a remote notification with the participation of the remote devices, APNs and maybe more than one providers.

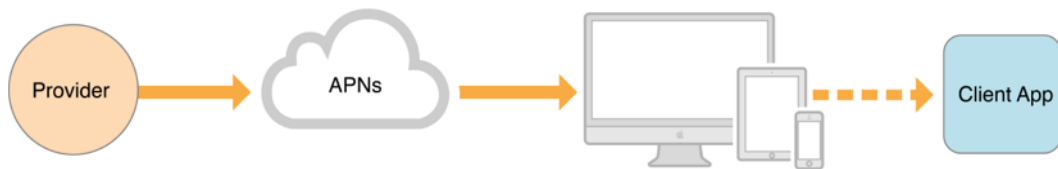


Figure 27: Path with one provider

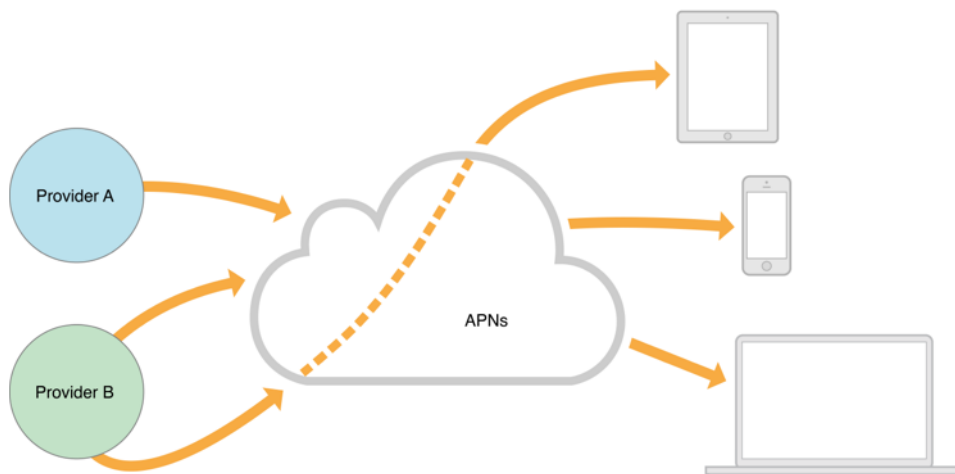


Figure 28: Path with multiple

### 12.2.2.Security Architecture

To provide the communication in high level of security, APNs adjusts the entry point between the providers and devices with two different level of trust: connection trust and token trust [17].

Connection trust makes sure that APNs is connected to a provider who is authorized to deliver notifications. The connection between APNs and devices are also ensured by connection trust which requires the legitimacy of the devices.

Here are two figures showing the mechanism of these two kinds of connection trust:

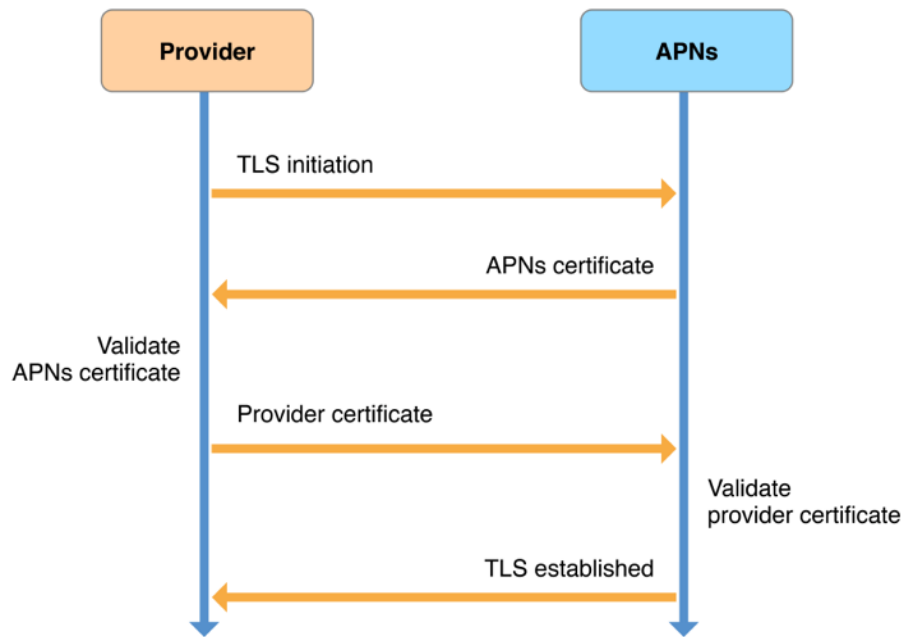


Figure 29: Connection trust between providers and APNs

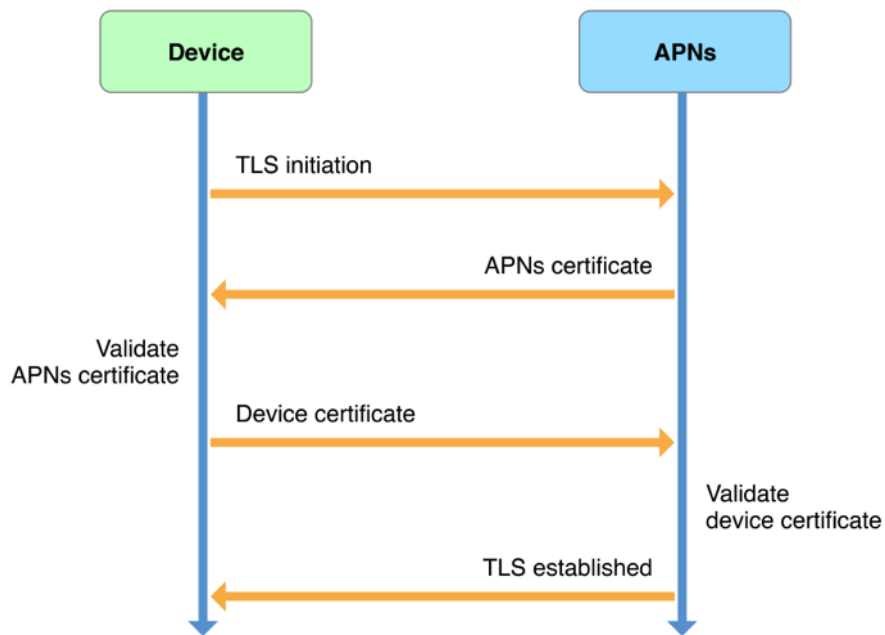


Figure 30: Connection trust between devices and APNs

## 12.3.Pushing Notifications with Node.js

The connection between the provider (our server) and APNs can be separated into two components. We have to implement both of them in order to deliver a productive service [19].

- **Gateway Component:** The gateway component is the TLS connection showed in the previous chapter. As a provider, our server has to construct a connection to APNs to send message and let APNs process the message to users. Apple recommends a constant connection to be established between a provider and APNs, which is always on, no matter if there is any message to be sent.
- **Feedback Component:** The feedback mechanism make it possible for providers to occasionally get a list of devices that are not acceptable of receiving the notifications for a certain application. We also have to implement a feedback workflow before going to production since Apple is monitoring all providers' usage of APNs in order to make sure providers are not wasting resources on those devices which can not receive notifications.

### 12.3.1.Node.js Module: APN Agent

APN Agent is a Node.js module developed to facilitate the procedure of sending remote notifications with APNs. It is integrated with many features that makes it easier for developers to implement a provider for pushing notifications. The main features we use are:

- Maintaining a persistent connection between the server and APNs gateway.
- Configuring the connection to be auto-reconnect when error occurs.
- Using the chain-able message builder to customize messages.
- Using the feedback service to get a list of those devices which can not receive notifications any more.

Installation: `$ npm install apnagent`

Set up with certificate:

```
var pfx = join(__dirname, './sslcert/aps_development_key.p12');
agent.set('pfx file', pfx).set('passphrase',
'passphrase').enable('sandbox');
```

Establish the connection:

```
agent.connect(function (err) {
    if (err && err.name === 'GatewayAuthorizationError') {
        console.log('Authentication Error: %s', err.message);
        process.exit(1);
    }
    var env = agent.enabled('sandbox') ? 'sandbox'
        : 'production';

    console.log('apnagent [%s] gateway connected', env);
});
```

Push notifications:

```
agent.createMessage()
    .device(deviceToken)
    .alert(news.title)
    .set('newsId', news.newsId)
    .set('tab', 3)
    .badge(1)
    .send();
```

Deal with Feedback:

```
feedback = new apnagent.MockFeedback();
feedback.set('interval', '30s').connect();
feedback.use(function (device, timestamp, done){
    .....
});
```

## 12.4.Receiving and Processing News Notification in iOS

iOS push notification are handled by the OS itself and then are passed to the application for further processing. When receiving a push notification from our server, such as a new activity promoting, there are 3 states the application can be, and need to be handled accordingly:

- Completely not started:

The app will start and do its initialization, check permission and register notification service. Then a delegate method will be called to check the remote notifications.

- Background mode:

The app will become active and check notifications in the delegate method.

- Active mode:

The app will directly run the code inside the delegate callback function without showing notification bar on the top.

## 12.5. Core Data

Core Data [23] is an object graph and persistence framework provided by Apple in the OS X and iOS operating systems. Core Data are used to store fetched news in local space so that when there is no internet connection, users can still check old news. In our application, we store fetched news in SQLite and fetched images in Data Container in the app's sandbox.

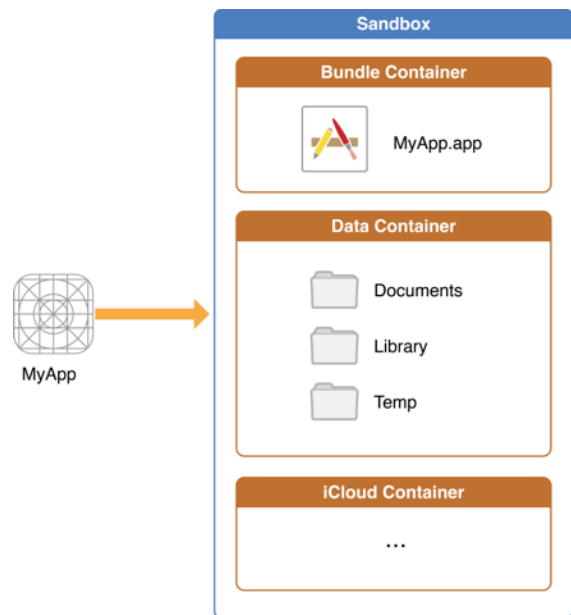


Figure 31: iOS app storage

## 13. Multiple Language Support

### 13.1.Client Side Design

For the need of change the language on the fly, there is a global variable indicating current language code stored in the persistent storage. The code is either “en” for English or “cn” for Chinese. When the user interface needs to configure text, it first get the unique key of the text, then go to corresponding language file to fetch the text for current language. After implementing the mechanism, when adding new text, all needed to do is only translating the text in different language file.

### 13.2.Server Side Support

In order to support the multiple language features, we have to do some modifications on our database and also change the way we generate response data from server.

#### 13.2.1.Database

The first thing we have to improve in database is the schema of some tables which are storing the data to be displayed on the client side.Taking the table *item* as an example: *title* and *description* are the two columns that stores the data to be displayed on client side. So we have to add two more columns to this table for a new language.

Here are the original schema and the updated schema for *item* table:

```
beaconUUID: String,  
beaconMajor: String,  
beaconMinor: String,  
title: String,  
coverImage: String,  
author: String,  
country: String,  
description: String,  
soundtrack: String,  
images: Array,  
lat: Number,  
lng: Number,  
introduceTime: Date,  
inExhibition: Boolean,  
views: [String],  
likes: [String],  
shareCount: Number
```

```
beaconUUID: String,  
beaconMajor: String,  
beaconMinor: String,  
title: String,  
coverImage: String,  
author: String,  
country: String,  
description: String,  
soundtrack: String,  
images: Array,  
lat: Number,  
lng: Number,  
introduceTime: Date,  
inExhibition: Boolean,  
views: [String],  
likes: [String],  
shareCount: Number,  
title-zh: String,  
description-zh: String
```

## 13.2.2.Format of Response Data

In order to keep the consistency of the API design, we decided not to add new APIs for multiple languages. In consequence, we decided to change the format of the response data of some API to support multiple languages.

Take API *item/:uuid/:major/:minor* as an example. This API is to get all the information of one certain item and the response data is a JSON object. We changed some properties of this JSON object from *string* type to *array* to support multiple language.

Here are the updated format and an example for the response data.

```
beaconUUID: String,  
beaconMajor: String,  
beaconMinor: String,  
title: {  
  languages: {  
    en: 0,  
    zh: 1  
  },  
  content: Array  
,  
coverImage: String,  
author: String,  
country: String,  
description: {  
  languages: Array,  
  content: Array  
,  
soundtrack: String,  
.....  
views: [String],  
likes: [String],  
shareCount: Number
```

```
...  
title: {  
  languages: {  
    en: 0,  
    zh: 1  
  },  
  content: {  
    "English  
content.",  
    "Chinese  
content."  
  }  
,  
...  
description: {  
  languages: {  
    en: 0,  
    zh: 1  
  },  
  content: {  
    "English  
content.",  
    "Chinese  
content."  
  }  
,
```

# 14. Account Managing System

## 14.1.Client Side Design

With account system, users can bind purchased tickets to their account so that they can access their tickets from different devices. In the front-end, we implemented a sign in / sign up page for using personal accounts. Facebook login is also supported to make the whole process more convenient.

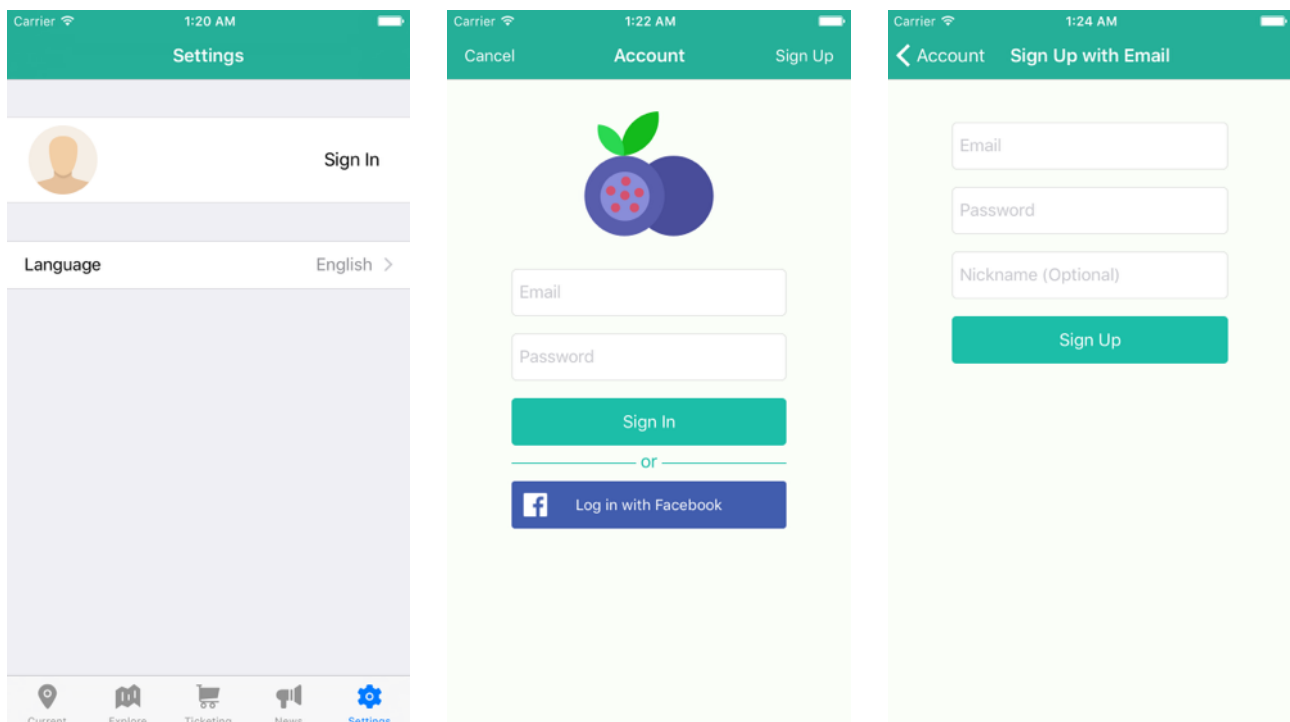


Figure 32: Account views

In Settings tab, when user tap the account cell, sign in page will show up. When using it the first time, user needs to sign up first by tapping “Sign Up” button on the navigation bar and the sign up page will appear if the user wants to sign up using emails.

If a user wants to just use Facebook account to sign in, he/she doesn’t need to do extra register step. Our backend will handle its first sign in to register a new account.

### 14.1.1.Account Manager

To manage the status of current user, we implement a singleton called Account Manager.

Whenever we need to access users sign-in status, we only need to contact the Account Manager singleton object to do corresponding operations.

The operation are listed as follows:

Method	Availability	Description
<b>isLoggedIn() -&gt; Bool</b>	Public	Check if user has logged in
<b>logout()</b>	Public	Log out current user
<b>login(withAccessToken token: String, userId: String, nickname: String?, imagePath: String?)</b>	Public	Call api to log in using credentials input by user
<b>updateAccessToken(token: String?)</b>	Private	To be used in login() to configure access token after successfully log in
<b>accessToken() -&gt; String?</b>	Public	Get the access token, which is needed when making account related API call
<b>updateUserId(userId: String?)</b>	Private	To be used in login() to configure user ID after successfully log in
<b>userId() -&gt; String?</b>	Public	Get user ID of current user
<b>updateNickname(userId: String?)</b>	Private	To be used in login() to configure nickname of user after successfully log in
<b>nickname() -&gt; String?</b>	Public	Get user's nickname
<b>updateImagePath(userId: String?)</b>	Private	To be used in login() to configure image path after successfully log in
<b>imagePath() -&gt; String?</b>	Public	Get profile image path

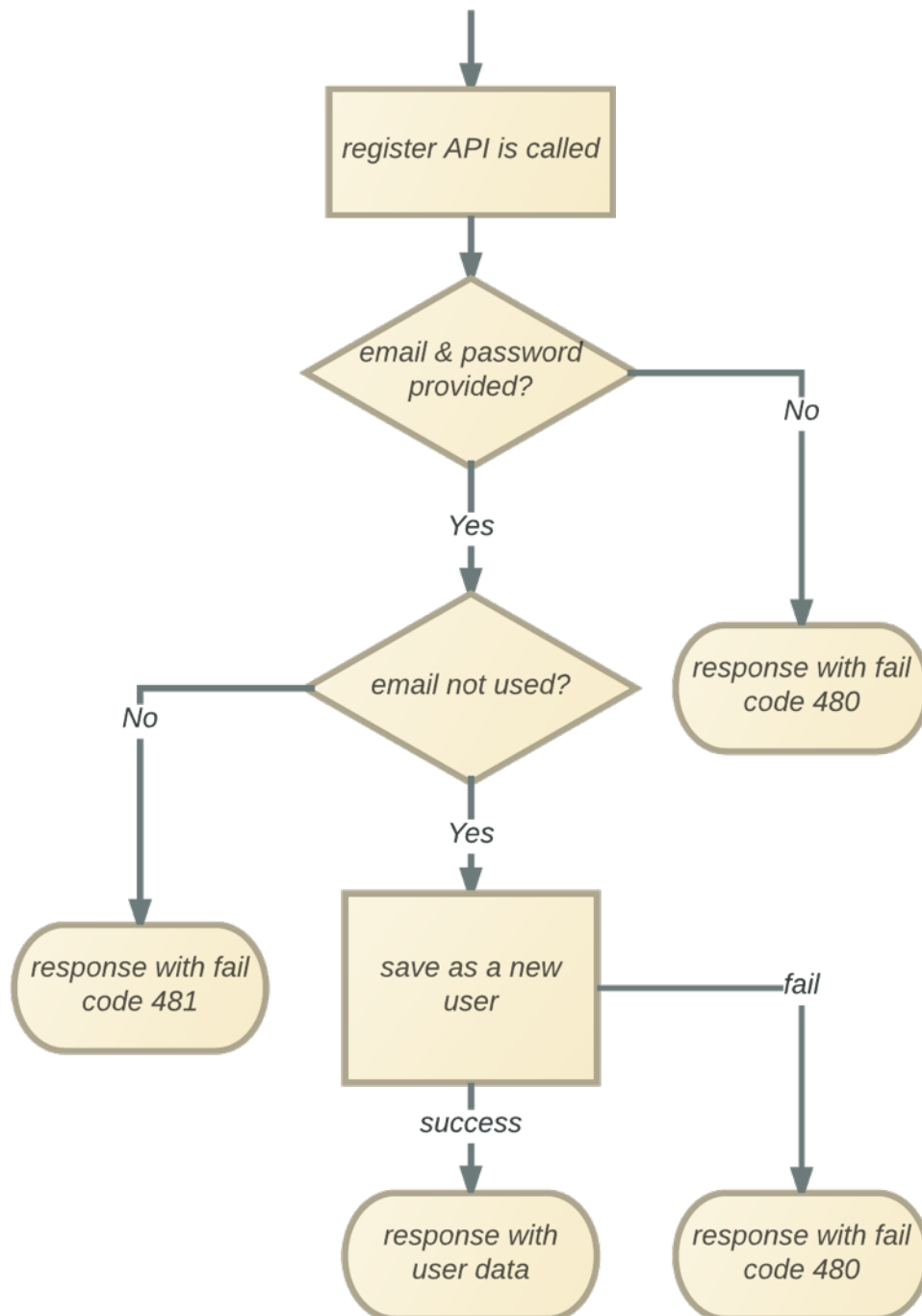
## 14.2.Server Side Implementation

We have implemented all the APIs related to users' actions under the router api/users. Below is a table illustrating the usage of these APIs:

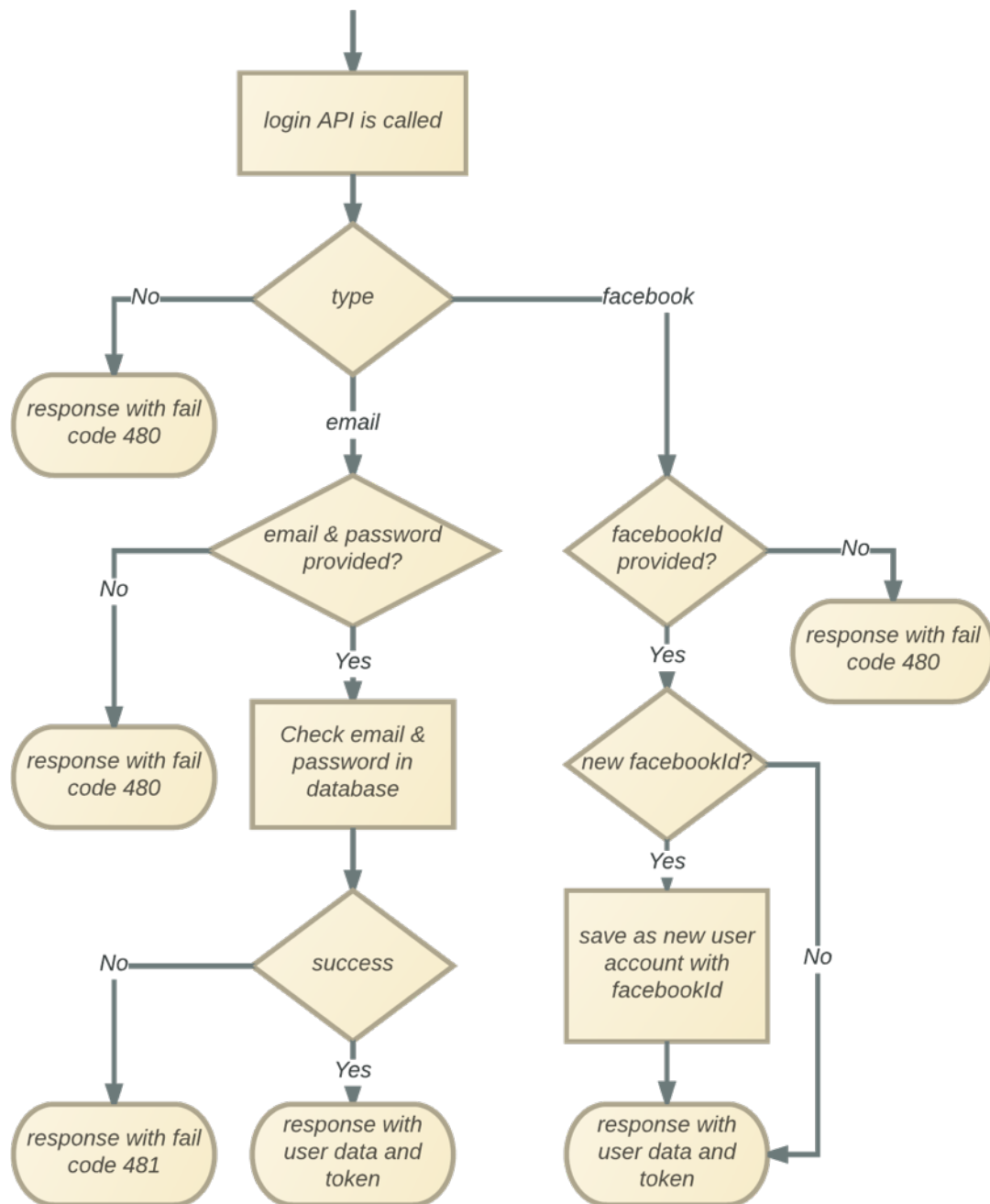
API	Method	Permission	Arguments	Usage
<b>/register</b>	post	public	email: string password: string name: string	Create a new user regarding to the information provided. If the email is already used for another user, our server will response with error code 481. If success, our server will response with a JSON object containing user's data.

API	Method	Permission	Arguments	Usage
<b>/login</b>	post	public	type: string email: string password: string facebookId: string	There are two login type: "email" and "facebook". If the user is trying to login with email, he has to provide email and password. If the user is trying to login with Facebook , he has to finish the authorization procedure in client side and then our application will send his facebookId to the server.
<b>/bind</b>	post	user	token: string facebookId: string	This API is for an logged in user to bind his email account with his Facebook account. So that this API need token to be provided indicating the login status of the user. Our server will find out the corresponding user account according to the token and bind a Facebook account to this existing user account.
<b>/logout</b>	post	user	token: string	Log out a user from our application.
<b>/all</b>	post	admin	No	Get a list of all users from our database. This API requires admin permission to process.

Below is a diagram showing the workflow of registering new users:

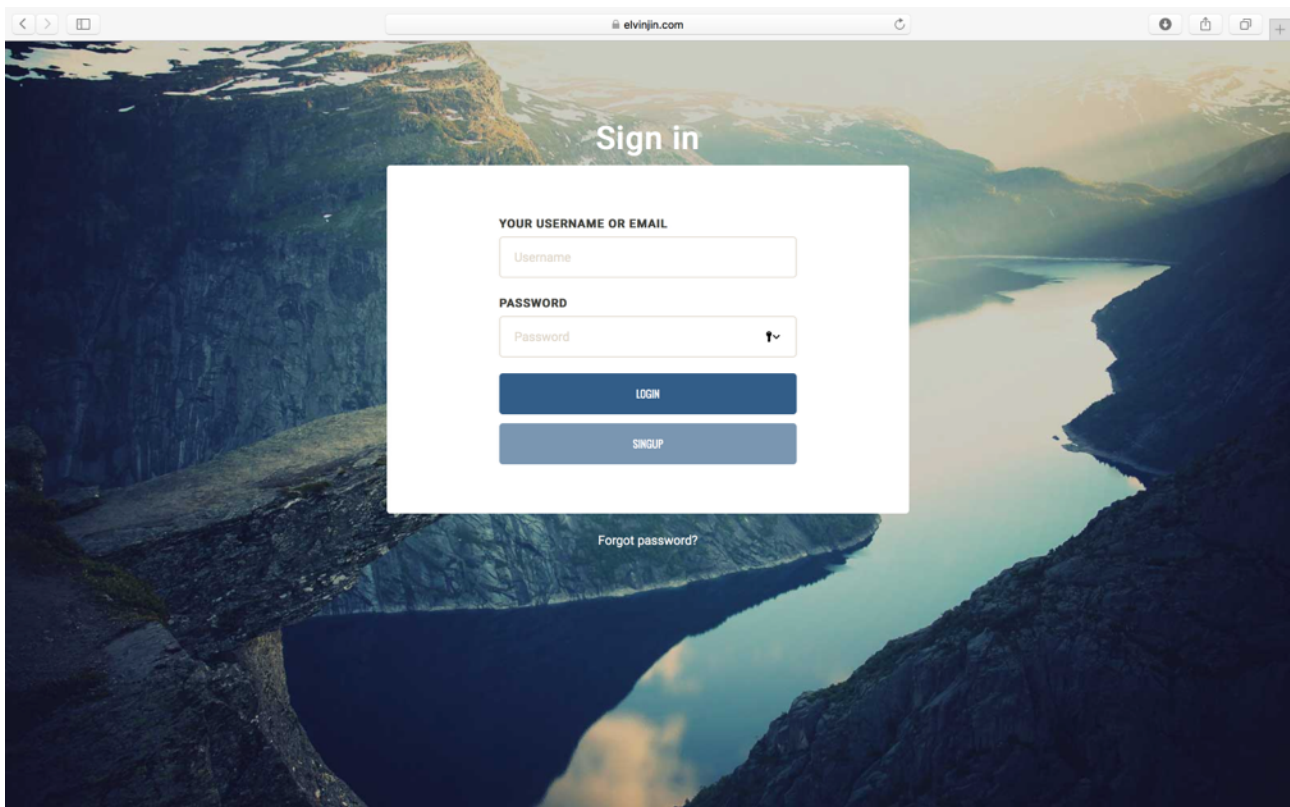


Below is a diagram showing the workflow of logging in:



## 15. Web Admin Panel

We have built a website as an admin panel for the administrators of our application so that they can have a user interface to manipulate and update data more conveniently. This website requires admin account to log in. The URL of this website is <https://elvinjin.com:8081/panel/login>.



**Figure 33:** Admin panel - log in

After the administrator logged in the panel, he/she can see all the data in detail at the homepage. In addition to the total number of the items, users, orders and news, the administrator can also see a sales report showing the number of sold tickets and redeemed tickets in recent 5 months. A list of the latest 5 orders will also be showed on the right side.

Instead of just having a view on the data, administrators can also do modification in this website. Following is a table showing all the functions of each page:

URL	title	Usage
<b>/panel/home</b>	Dashboard	Administrators can see a summary of all kind of data here.
<b>/panel/item</b>	Items	Administrators can see all the items stored in the database displayed in a google map. Each marker on the google map corresponds to one item. Administrators can go to another page to modify the information of one item by just click the corresponding marker. If the administrator wants to add a new item, he can click the button “Add New Item” to do so.
<b>/panel/addItem</b>	New Item	In this page, the administrator can add a new record of item into the database by type in all the required information. Once the form is submitted, the browser will jump back the previous page and the administrator can see the new item he just added.
<b>/panel/editItem</b>	Edit Item	In this page, the administrator can do any modification he wants on the taggert item, including change title, author, description, beacon information and even related images. Once the form is submitted, the browser will also jump back.
<b>/panel/user</b>	Users	A list of all registered users’ information will be showed in this page.
<b>/panel/order</b>	Orders	A list of all registered orders’ information will be showed in this page.
<b>/panel/newsList</b>	News List	The administrator can see a record of all pushed news and he can go to the detail of each piece news by clicking on it. He can also choose to push a new piece of news by clicking on the button “Create News”.
<b>/panel/newsDetail</b>	News Detail	A page shows the detail content of a certain piece of news, which can be pushed again.
<b>/panel/addNews</b>	Push News	In this page, the administrator can create a new piece of news by fill in the form and submit it.

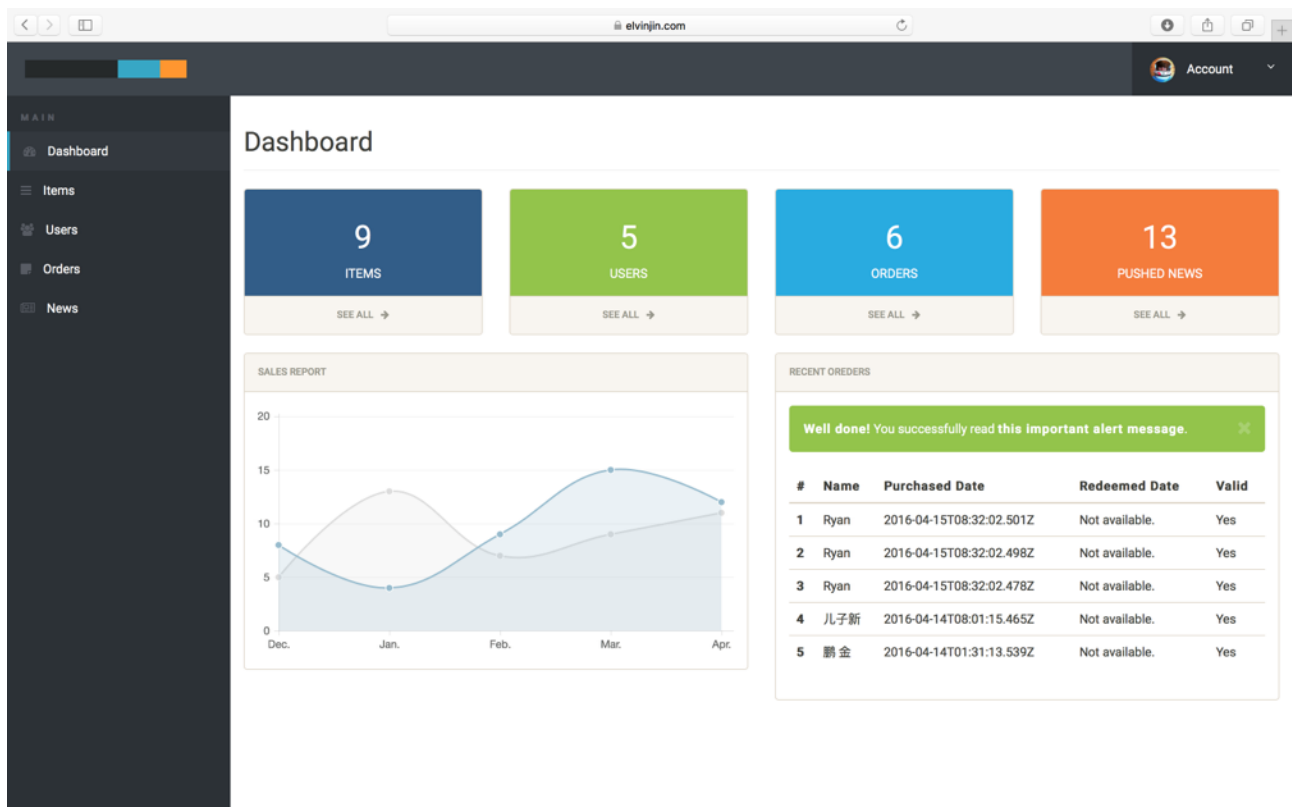


Figure 34: Dashboard

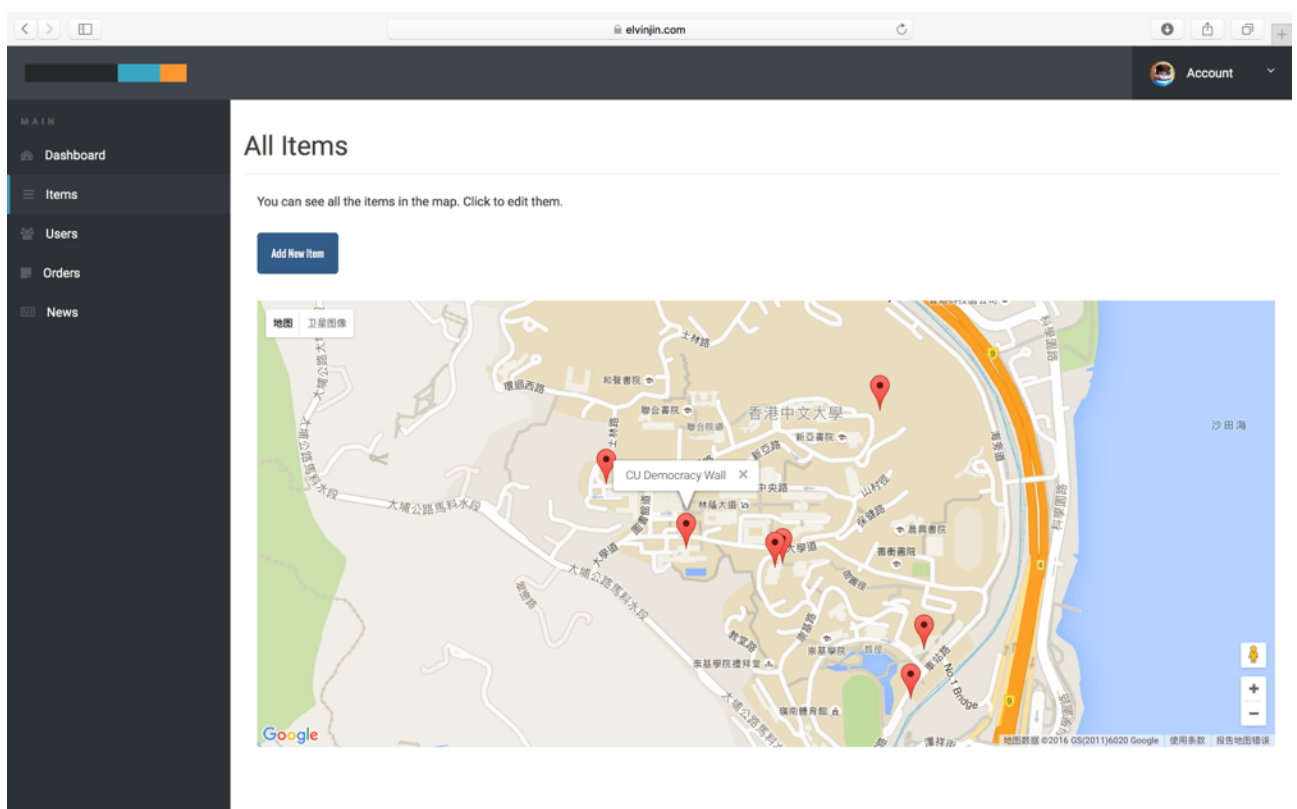


Figure 35: Item list

elvinjin.com

Account

MAIN

Dashboard

Items

Users

Orders

News

## Add New Items

Title:

Author (if any):

Cover Image:

选取文件 未选择文件

More Images:

选取文件 未选择文件

Soundtrack:

选取文件 未选择文件

Position:

Drag the marker to the target position

lat:

22.419066

lng:

114.207126

Map View:

地图 卫星图像

MAIN

Dashboard

Items

Users

Orders

News

Users

ALL USERS

#	Name	Email	User id.
1	Ryan HAN	Not Provided.	146018342931544oL2HH
2	Ryan HAN	Not Provided.	1460183554283GuHymaB
3	Peng Jin	Not Provided.	1460195766909LVZwXfv
4	Elvin	elvinjin8@gmail.com	14601957843720R8dTFk
5	Ryan Han	Not Provided.	1460620862419d1oDZei

#	Name	Email	Date	Ticket No.
1	Ryan	hanzuohao123@gmail.com	2016-04-15	1460709122501UGxJ8N
2	Ryan	hanzuohao123@gmail.com	2016-04-15	1460709122498qzhO4w
3	Ryan	hanzuohao123@gmail.com	2016-04-15	1460709122478bhAm5n
4	儿子新	hanzuohao123@gmail.com	2016-04-14	1460620875465LJFIP7
5	鹏金	elvinjin8@gmail.com	2016-04-14	1460597473539la7Vhw
6	Elvin Jin	John-Appleseed@mac.com	2016-04-09	1460199956329NGIF2S

Figure 38: Orders

ID	Timestamp	Headline
ID: 1460598440427R8ZZYX	TIME: THU APR 14 2016 09:47:20 GMT+0800 (CST)	EU Referendum: Don't discount raw emotion
ID: 1455520141129WM8VUZ	TIME: MON FEB 15 2016 15:09:01 GMT+0800 (CST)	Van Gogh's bedroom gets digital makeover
ID: 14543186296418ZNSVS	TIME: MON FEB 01 2016 17:23:49 GMT+0800 (CST)	Lift-off for Europe's space laser network

Figure 39: News list

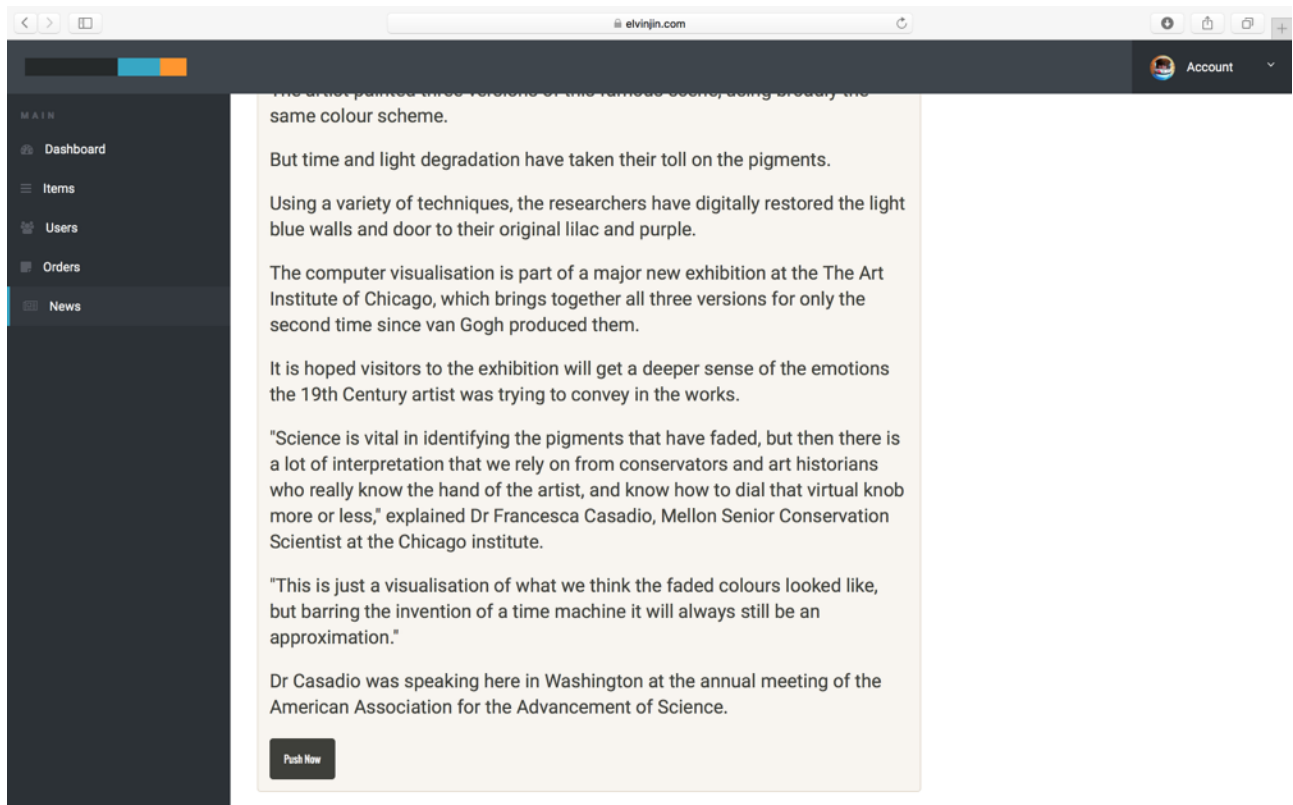


Figure 40: News detail

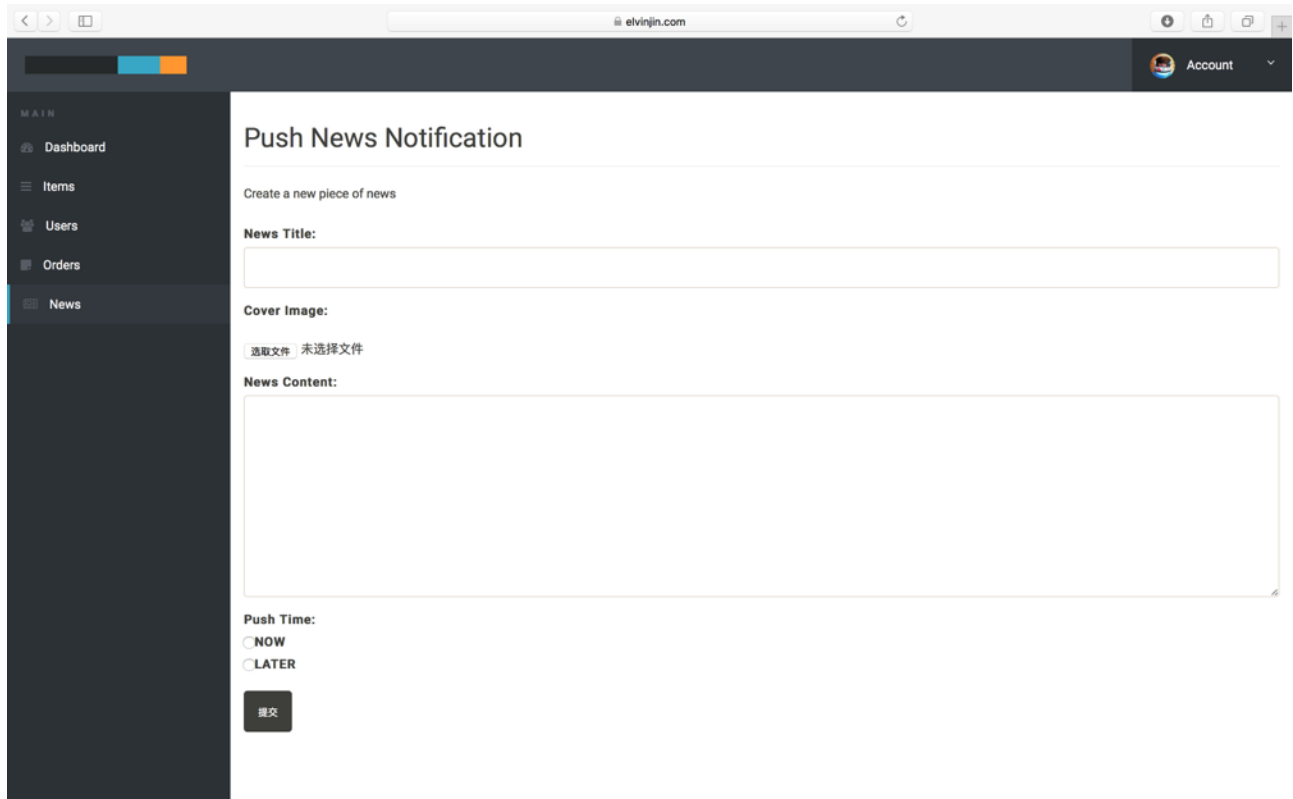


Figure 41: Create news

## 16. Backend Code Revision

As the number of features grows, we added more APIs in the backend and the total length of the code is becoming larger. We did a revision on the backend implementation and modularized some part of the code in order to make the whole structure clearer and more scalable.

### 16.1. Standardize HTTP Response

In the former version of our backend code, we were dealing with each response separately and with no standard to follow. After we have implemented more and more APIs, we feel obliged to standardize the way we modifying the HTTP response.

We customized two HTTP status code to indicate what kind of error is happening.

- **480:** In our design, code 480 will be received in two situation. The first one is logic error. For example, one API requires 3 arguments in HTTP request, while the client only provided 2 of them, or the name of the parameter is different from what the server is expecting. The second one is database internal error. For example, the server can not get/save a record from/into the database.
- **481:** Code 481 will be received when the provided data is invalid. For example, wrong user name, wrong email, wrong new ID and other data error.

We implemented an appHelper module to deal with all these HTTP responses and other things that can be modularized out from the router file like exchanging data from database.

Below is how we deal with HTTP responses in router file last semester:

```
res.json({
  success: true,
  message: "User registered."
});
```

```
res.status(481).send({
  success: false,
  message: "No matching item."
});
```

Below is how we deal with HTTP response now:

```
appHelper.errorResponse(res, 481, 'error', 'Cannot get item.');
```

```
var resData = appHelper.userInfoWithToken(user, token);  
res.status(200).send(resData);
```

In appHelper.js:

```
exports.errorResponse = function(res, code, title, message){  
    res.writeHead(code, title);  
    res.write(message);  
    res.end();  
}  
exports.userInfoWithToken = function(user, token){  
    var ret = {  
        userId: user.id,  
        name: user.name,  
        email: user.email,  
        facebookId: user.facebookId,  
        imageURL: user.imageURL,  
        type: user.type,  
        token: token  
    };  
    return ret;  
}
```

## 16.2.Protecting APIs with Middleware

Middleware is functions that will be involved by the Express.js routing layer before we actually handle the request, so that it is placed between the raw request and the final intended handler. We usually see these functions as middleware stack because they are always invoked according to the order they are placed [20].

Below is an example of a middleware and how this middleware is used:

```

exports.userMiddleWare = function(req, res, next) {
  // check header or url parameters or post parameters for token
  var token = req.body.accessToken || req.get('accessToken');
  var id = req.body.userId || req.get('userId');
  if(!token || !id){
    return exports.errorResponse(res, 480, 'error', 'message');
  }else{
    JWTUserId.findOne({token:token, userId:id},function(err, result){
      if(err) throw err;
      if(result){
        next();
      }else{
        return exports.errorResponse(res, 481, 'error', '');
      }
    });
  }
}
}

```

We can use some certain middleware to protect our APIs. We have two middleware functions to check if the request has the permission to access its target API and both of them are implemented in appHelper.js. The example showed above is the middleware checking if the request is sent from an user-logged-in client.

Below is the middleware checking if the request is sent from a admin-logged-in client.

```
exports.adminMiddleWare = function(req, res, next) {
  var token = req.body.accessToken || req.get('accessToken');
  var id = req.body.userId || req.get('userId');
  if(!token || !id){
    return exports.errorResponse(res, 480, 'error.', 'message.');
```

```
  }else{
    JWTAdminUserId.findOne({...},function(err, result){
      if(err) throw err;
      if(result){
        next();
      }else{
        return exports.errorResponse(res, 481, 'error', '');
      }
    });
  }
}
```

```
}
```

## 17. Difficulties in the Project

During doing this project, we have faced the following difficulties which took us a relative long time to solve.

- When doing the research on the indoor positioning and studying on our first approach, we read some paper on this topic and found a solution that are relatively accurate. In order to understand the theory behind that approach, we did research on the features of the bluetooth energy in order to learn the way it broadcast and how its signals are interfered. At the end we got an algorithm and a plan on how to calculate the environment of one room and map it into a parameter for position analyzing, but we gave up this approach because it is nearly impossible to calculate the parameter, which represents the environment of a big and complex exhibition hall.
- Designing the indoor map also cost us a lot of time. As what we have mentioned at the section “Study of Custom Indoor Map Design”, we spend much time on trying to use Mapbox as a tool to create the custom indoor map, in which most of the time were spent on how to draw the map as a .geojson file. At that time, we even developed a quite complicated method to parse the .geojson file and draw it out by ourselves. But at the end, we gave up Mapbox for the reasons mentioned early.
- When building the server for signing and distributing the passes, we kept failing on the final point, which means we could not know what client side get from the server or whether the data generated by the server is correct. Since the error occurred at the final step, we don’t know the reason why the file failed to be downloaded and opened. After a long time fixing, we replaced the certificate and regenerate the .pem file using the command provided by passbook module and change the way we used to update the template of the pass. Finally, we get out pass and can successfully add it to our passbook.
- When we were implementing the pushing notification, we were unable to push the notifications for a long time but we can not find out the reason. The connection between our server and APNs was working well and there were no error thrown from the console. We finished procedure of sending the notification but had no response. After reviewing the document of the APN agent (the node.js module we used for APNs) , we found that one of the object named agent was used in a wrong way. Instead of creating another agent

in the router layer, we have to use the same instance of it to send notifications after the connection was established. We solve that problem by moving the router of pushing to where this agent is initialized. Finally, we pushed our first notification successfully to our application.

- When developing our admin panel, we spent a lot of time thinking how to make our admin website safer. Because all the operations in this admin panel can change the data we store in the data base directly, we must not let other non-admin people get the access to this website. It seems just a problem of session storage, but how we arrange our html, javascript and CSS resources is also very important. We set all the javascript and CSS files as static files in order to let browsers have the access to them when loading certain html pages. We did not set any html pages as static files to avoid other people visit this website through file name. Instead, we build some APIs with relative path which will read the local html files in the server and send response with html data. Once the admin has successfully logged in the website, we will set a pair of cookies in the client side to keep the admin logged in. Each request sent to admin APIs will be checked by an admin middleware first so that unapproved actions will be rejected. At last, we think our admin panel is safe enough.
- When implementing the news tab in our iOS app, a very important part of the implementation is to store the received news locally. The first solution we came out was to store all the information in the form of SQLite inside the persistent memory of the application. But SQLite is only designated for light storage, while we may have some large image to store. Finally, we decided to store the image separately in the data store of the sandbox and then store the path of this image together with other information as SQLite. This solution was proved feasible and very efficient.

# 18. Contribution and Reflection

## 18.1. Contribution

### 18.1.1. Fall 2015

This project kick off in fall semester of 2015. In the first few weeks, my main tasks are studying Apple's newest technologies and trying to think what we can use in this project. In this period, I gained deeper understanding of how the new technologies works and how they help people achieve more convenient life.

The project itself was also in the designing phase. Because I have studied basic design theories via online course before, I took the responsibility of designing the UI interface. Design is always my interest at spare time. As a engineer, we sometimes need to think as a designer to have better understanding of how users interact with the app. So I took my time designing the app, wanting to make every pixel to its best design.

When the design phase is finished, my partner Ryan is responsible for backend API development and I'm responsible for iOS front-end development. There are 5 tabs in our application, a.k.a. Current, Explore, Ticketing, News and Settings. I started to implement each tab one by one.

By the end of this term, I finished Current tab, Explore tab and Ticketing tab.

### 18.1.2. Spring 2016

In this semester, we no longer limit ourself to do the indoor navigation. Instead, we took advices from Professor Lyu and Dr. Edward, which is to scale the application to CUHK guide. After the main goal revised, I modified my design and then continued the work left since last semester.

When doing push notification feature, I work with Ryan closely because the backend also needs support to finish this task. With prior experience of manage Apple provisioning files and certificates, I helped backend generate the push certificates and successfully implement the push notification feature.

## 18.2.Reflection

During the development, I gained knowledge and practice of more cutting edge technologies such as node.js, swift, 3D touch, Apple Pay, Apple Wallet, APNs, Facebook SDK, Google Maps SDK, Core Data, etc. I think this was really a great experience for me because besides learning hard skills, I also learnt how to work with people with different backgrounds and learn from them. I think this would be valuable for my future self development.

## 19. Summary

After developing our application for 2 semesters, we finally implemented all the features planned. During the entire period of development, user experience is always of the first priority. When we were implementing one feature, we wanted to implement it in a robust way. We were not just developing it for demonstration purpose, but we see the goal of our project as developing a product which has complete functions and can be put into the market.

By the end of this semester, our application is equipped with its user account system, map system, ticket system, news system and other basic functions.

In application side, besides those traditional user interactions, we also took advantage of the latest iOS features such as 3D Touch, Apple Pay and Apple Wallet. In server side, we also have an admin panel website for administrators to manage the data.

We kept learning new technologies along the way of developing. We are getting much more familiar with terms like BLE, beacons, RSSI, APNs, Core Data and RESTful and tools like Node.js, Stripe and MongoDB. Now we can make great use of these tools to develop better applications.

At the end of this report, we would like to give our special thanks to Prof. Michael R. Lyu and Mr. Edward Yau, who are willing to take their time to meet with us and offering constructive comments on our project continuously. Without them we wouldn't have tackled this much difficulties.

# Reference

- [1] "iOS: Understanding iBeacon". Apple Inc. February 2015, <https://support.apple.com/en-gb/HT202880>
- [2] "Beacons: Everything you need to know.". Pointrlabs.com. 18 January 2015. <http://www.pointrlabs.com/blog/beacons-everything-you-need-to-know/>
- [3] Bluetooth.com, (2015). Bluetooth Low Energy | Bluetooth Technology Website. [online] Available at: <http://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-technology-basics/low-energy> [Accessed 28 Nov. 2015].
- [4] Warski, B. (2014). How do iBeacons work?. [online] Blog of Adam Warski. Available at: <http://www.warski.org/blog/2014/01/how-iBeacons-work/> [Accessed 28 Nov. 2015].
- [5] iBeacon.com Insider, (2014). What is iBeacon? A Guide to iBeacons. [online] Available at: <http://www.iBeacon.com/what-is-iBeacon-a-guide-to-beacons/> [Accessed 28 Nov. 2015].
- [6] Developer.estimote.com, (2015). [online] Available at: <http://developer.estimote.com/> [Accessed 28 Nov. 2015].
- [7] Estimote.com, (2015). Estimote. [online] Available at: <http://estimote.com/> [Accessed 28 Nov. 2015].
- [8] Developer.estimote.com, (2015). What is Estimote Indoor Location SDK. [online] Available at: <http://developer.estimote.com/indoor/> [Accessed 28 Nov. 2015].
- [9] Developer.estimote.com, (2015). Build an App With Indoor SDK. [online] Available at: <http://developer.estimote.com/indoor/build-an-app/> [Accessed 28 Nov. 2015].
- [10] Developer.apple.com, (2015). Model-View-Controller. [online] Available at: <https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html> [Accessed 30 Nov. 2015].
- [11] Apple Pay, (2015). Getting Started With Apple Pay. [online] Available at: <https://developer.apple.com/apple-pay/Getting-Started-with-Apple-Pay.pdf> [Accessed 29 Nov. 2015].
- [12] Wikipedia, (2015). Log-distance path loss model. [online] Available at: [https://en.wikipedia.org/wiki/Log-distance\\_path\\_loss\\_model](https://en.wikipedia.org/wiki/Log-distance_path_loss_model) [Accessed 1 Dec. 2015].

- [13] White, T. and White, V. (2013). 10 Must Have Apps That Support Apple's Passbook | BestAppSite. [online] BestAppSite. Available at: <http://www.bestappsite.com/10-must-have-apps-that-support-apples-passbook/> [Accessed 30 Nov. 2015].
- [15] Mics.org, (2015). NCCR - MICS - Project IP6 abstract. [online] Available at: <http://www.mics.org/micsProjects.php?groupName=IP6&action=abstract> [Accessed 1 Dec. 2015].
- [16] Developer.apple.com, (2015). Wallet Developer Guide: Pass Design and Creation. [online] Available at: [https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/PassKit\\_PG/Creating.html#//apple\\_ref/doc/uid/TP40012195-CH4-SW1](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/PassKit_PG/Creating.html#//apple_ref/doc/uid/TP40012195-CH4-SW1) [Accessed 1 Dec. 2015].
- [17] Developer.apple.com. (2016). Apple Push Notification Service. [online] Available at: <https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html> [Accessed 19 Apr. 2016].
- [18] Developer.apple.com. (2016). The Remote Notification Payload. [online] Available at: <https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/TheNotificationPayload.html> [Accessed 19 Apr. 2016].
- [19] Luer, J. (2016). Delivering iOS Push Notifications with Node.js. [online] Engine Yard. Available at: <https://blog.engineyard.com/2013/developing-ios-push-notifications-nodejs> [Accessed 19 Apr. 2016].
- [20] Safaribooksonline.com. (2014). Express.js Middleware Demystified - Safari Blog. [online] Available at: <https://www.safaribooksonline.com/blog/2014/03/10/express-js-middleware-demystified/> [Accessed 19 Apr. 2016].
- [21] "Google Maps SDK for iOS". Google Developers. Web. 19 Apr. 2016.
- [22] "Google Maps Directions API". Google Developers. N.p., 2016. Web. 19 Apr. 2016.
- [23] "Core Data Programming Guide: Connecting The Model To Views". Developer.apple.com. N.p., 2016. Web. 19 Apr. 2016.