

Campus Conqueror

A Design Project Report

Presented to the School of Electrical and Computer Engineering of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering, Electrical and Computer Engineering

Submitted by

Xiaoyu Guo (xg229), Yuxin Cao (yc2263)

MEng Field Advisor: Joe Skovira

MEng Outside Advisor: Aija Leiponen

Degree Date: January 2017

Abstract

Master of Engineering Program

School of Electrical and Computer Engineering

Cornell University

Design Project Report

Project Title: Campus Conqueror

Author: Xiaoyu Guo, Yuxin Cao

Abstract: This MEng project is to implement and extend a business model proposed by four students¹ in Digital Businesses Strategy class from Johnson School. It is an events finder targeting specifically on all the events in Cornell and college town. It contains a website, an iOS application and a ticker. The skill needed for the project are web full stack development, iOS application development, cross platform database management and embedded system development. A website and an iOS application which support events displaying, searching, posting and cross platform data synchronization has launched. A Raspbeery Pi based arcade machine which can show upcoming events and allow people to play game with the joystick and buttons on it is also built. Compared with similar events finders, this event finder could overtake them because it is more informative, highly localized and cross-platform.

Keywords: Cross-platform, Events Finder, Website, iOS App, Arcade Machine.

¹ Team 5 of AEM 3220 - Digital Business Strategy Fall 2015: Nihar Suthar, Alejandro Mankin, Sungwon Yun, Brandon Greer

Xiaoyu Guo's Executive Summary

In the first semester, my contribution to this project is building an iOS app which meets the requirements from digital business group and work with my partner who has been working on the web development to setup a Firebase Database to store and share both the data from app and website.

In the second semester, my contribution to this project is optimizing the iOS app and working with my partner on writing python program on Raspberry Pi for events displaying based on the arcade machine, and also circuit soldering as well.

The iOS app is built using Xcode 7.3 for both the UI design and attributes realization. The online database used to store data is Firebase, and I also use Cloudinary to save the image data uploaded by users.

Acknowledgement

- How to create Side Menu: <https://github.com/romaonthego/RESideMenu>
- How to hide or show the Tab Bar: <http://blog.csdn.net/riveram/article/details/7345577>
- How to get the current location and show on the map: <http://stackoverflow.com/questions/24717547/ios-8-map-kit-obj-c-cannot-get-users-location>
- How to implement Calendar View in iOS: <http://www.code4app.net/ios/Weekly-calendar-view-for-iphone,-scrollable/54c5daefe24741cf0dc54249>
- How to create an awesome Refresh Control: <https://github.com/coolbeet/CBStoreHouseRefreshControl>
- How to implement Autocomplete Places for map: <https://github.com/mrugrajsinh/MVAutocompletePlaceSearchTextField>
- How to setup Raspberry Pi: <https://www.raspberrypi.org>
- How to use pygame to display: <http://www.pygame.org/hifi.html>

Implementation Details

- Login interface: new user can sign up first and use the username and pass word he or she just signed up for signing in, when the data loading is finished, the Home view will show up.
- Home view: this view is supposed to load all of the events from database and list them in time order, users are allowed to search in the search bar above the view for events according to event title or introduction.

- Map view: this view is designed to show the current events happening around on campus, allowing users to quickly get the information of the events by click the marker in the map and tap on the info domain of the marker.
- Categories view: this view is implemented to let users filter events by tags, offering a better way for users to quickly look for events they are interested in.
- Calendar view: this view provides a table for looking up events in certain date, which can also save time for users to look for events happened in previous date or happening in a few days.
- Side menu: this attribute is inspired by most of the current popular social apps like Facebook and Twitter. The side menu saves a lot of space for the main view and can provide more attributes and functions for users. In this app, the side menu offers “Home”, “Profile”, “New Post”, “My Posts”, “My Attendance”, “Settings” and “Log Out” functions for users, providing convenience for switching from one view to another view.
- “Profile”: users are allowed to make some changes to their nickname, age, gender, interests, etc.
- “New Post”: in this view, some special users with posting permit are able to post a new event to all of the users of the app. For the attributes of the event, there are event name, starting time and ending time, location, restriction, introduction, primary tag, secondary tag and images.
- “My Posts”: in this view, users can see all of the events posted by themselves, providing convenience for looking up in what they posted in previous time.
- Event detail view: this view are designed to show detail information of the event to users, and can be activated and shown by tapping on event cells in different views, like Home view, Map view, Categories view and Calendar view.

Yuxin Cao's Executive Summary

In the first semester, my contribution of the project is consulting with my teammate and digital business students to build a website to post and display events information. I also work with my teammate to setup a Firebase Database for permanent data storage and data communication with the iOS app.

In the second semester, my contribution of the project is to build a Python web crawler. It can add the events to the database by crawling Cornell event calendar website. It also provides a command line interface for us to delete all the events with one commands. I also work with my partner Xiaoyu to develop the arcade machine, including developing the python program and soldering the circuit board.

The website is built using Bootstrap framework and jQuery library for the front end, Node.js on the backend, and Firebase as the database, Cloudinary and AWS S3 as image service provider. The website is currently being hosted by heroku at <http://cueeventsfinder.herokuapp.com/>.

Acknowledgement

- I used some open source bootstrap template from <http://startbootstrap.com/>
- I used open source JQuery library <https://github.com/jdewit/bootstrap-timepicker/>
- The homepage design of my website is inspired by <https://www.eventbrite.com/>
- The map page design of my website is inspired by <https://www.airbnb.com/s/San-Francisco--CA?source=ds>

Implementation Details

- Index page is a sign in and sign up page that perform basic account lookup, account adding and storing some user information to the cookies so that it wouldn't need to be extracted again from the database.
- On homepage, user can see the top six most popular events (most viewed), a button redirecting to post event and six category images to redirect user to different categories. At the top of the homepage, there are navigation tabs for filtering events by starting time and finding out all the events happening today.
- If the user has trigger a search (by time, category or keywords), the user would be redirect to a list of search result page. Then the user could choose the event most interested in and looking at more details.
- In the page 'events happening today', all the events are displayed in a map view. When user hover the mouse on some events on the left side, a marker indicating the location of the events would be highlighted on the right hand side map.
- To post the events, the user could specify the events title, time and location, whether the events is restricted to a specific group of people and upload some images for the events.
- The python web crawler use the python Firebase API provided from <https://pypi.python.org/pypi/python-firebase/1.2>

Contents

Introduction.....	7
Website.....	8
We want a simple, clear and attractive homepage	8
Support different size of devices	10
Handlebars	11
An interactive map view	13
Simple Form to Post Event	14
One Result List Template for All.....	16
iOS App.....	17
Though small perfectly formed	17
Detailed Event Information View	19
Side Menu Design and Event Post Management.....	19
Data Management	22
Arcade Machine	23
Pygame Display Implementation.....	23
Implementation of the Python Web Crawler	25
Fetch and Parse Events with Python	25
Geocode the locations	25
Decide the Category for Fetched Events	26
Problems we encountered.....	27
Image Management Service.....	27
Joystick not Always Responsive	30
Conclusion	30
Appendix.....	31

Introduction

Studying in Cornell is challenging, so we want to make finding and hosting events easier. Currently, students get informed of events in four ways: social network, stalls or drawings at school, Cornell Events Website, and Handshake for professional events. We think there might be a way to gather all the information into one place, and allow users to search, filter and even subscribe for the event they are interested in. This cross platform system shall seamlessly combine three parts, website, iOS application and an arcade machine, sharing the same database. The website is the most powerful and efficient way to search for and filter out interested events, posting and managing events. The iOS application is built to allow real-time notification from the host of event, and seamless coordination with other mobile applications. For example, with the embedded google map API, the user can see its location and the events happening nearby. The arcade machine works as a complementary part of the other two. It would be placed in public area and display the most popular upcoming events. With joystick and buttons, we also built a small game to the arcade machine. The background of the game is the content of events. The events information in three different platforms are fetched and post to the same database (Firestore). We use AWS S3 and Cloudinary to store image, and put their URLs to Firestore. We also create a Python web crawler to fetch Cornell sponsored events information from Cornell events website. It can parse the content of the website, and classify the events into different categories based on the keyword in the events details.

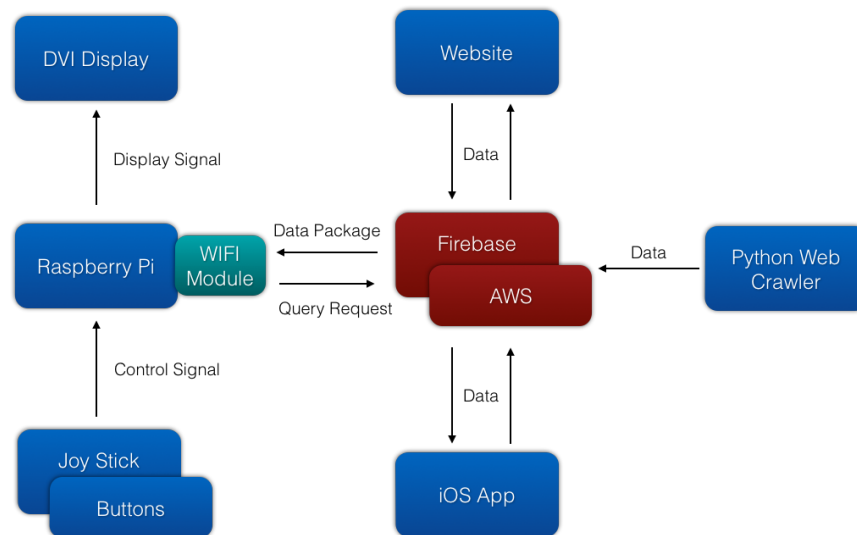


Figure 1.1 System Diagram of the Project

Website

We want a simple, clear and attractive homepage

We want to build a website that looks appealing. It should have a nice layout and smooth animation effects. We also want to build a website that is straight forward to use. It should have handy navigation tab, clear layout of different functionalities. Last but not least, we want something that we can build quick and easily. To accomplish these, we choose bootstrap framework, which has a handful of open source template built by experienced developer. So we adapted and combine several open source template from <http://startbootstrap.com/> to build our homepage.



Figure 2.1 Homepage at posting events

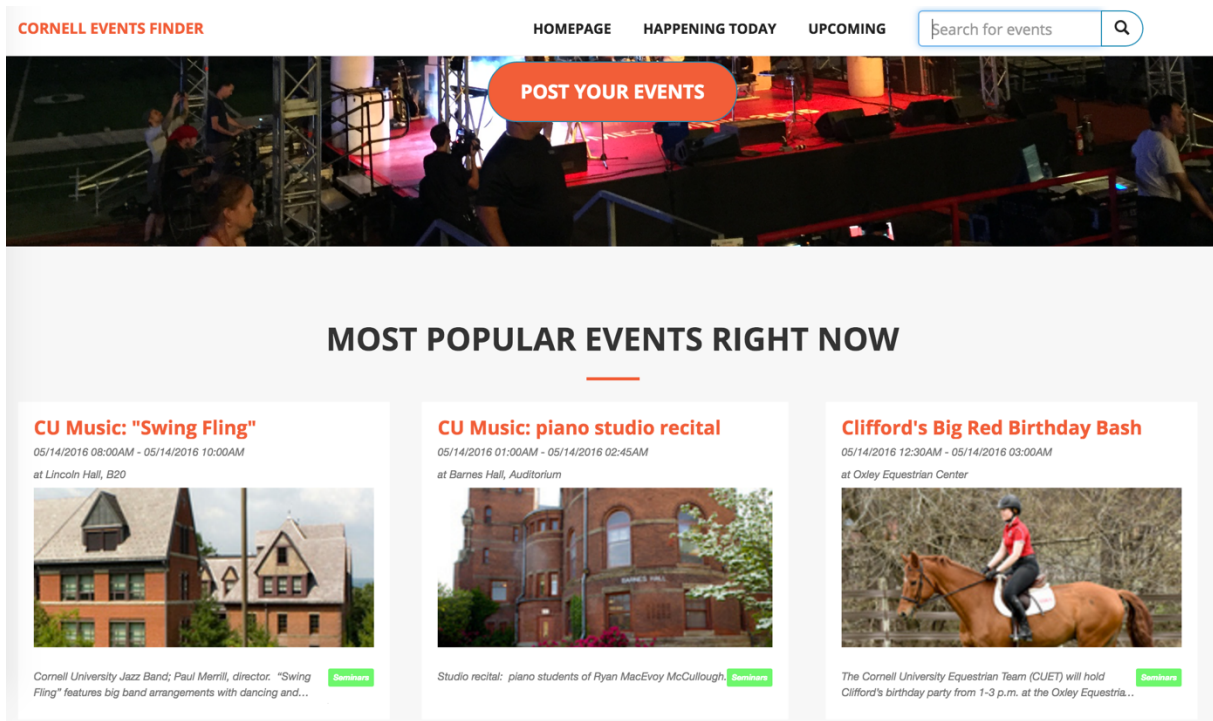


Figure 2.2 Homepage at popular events

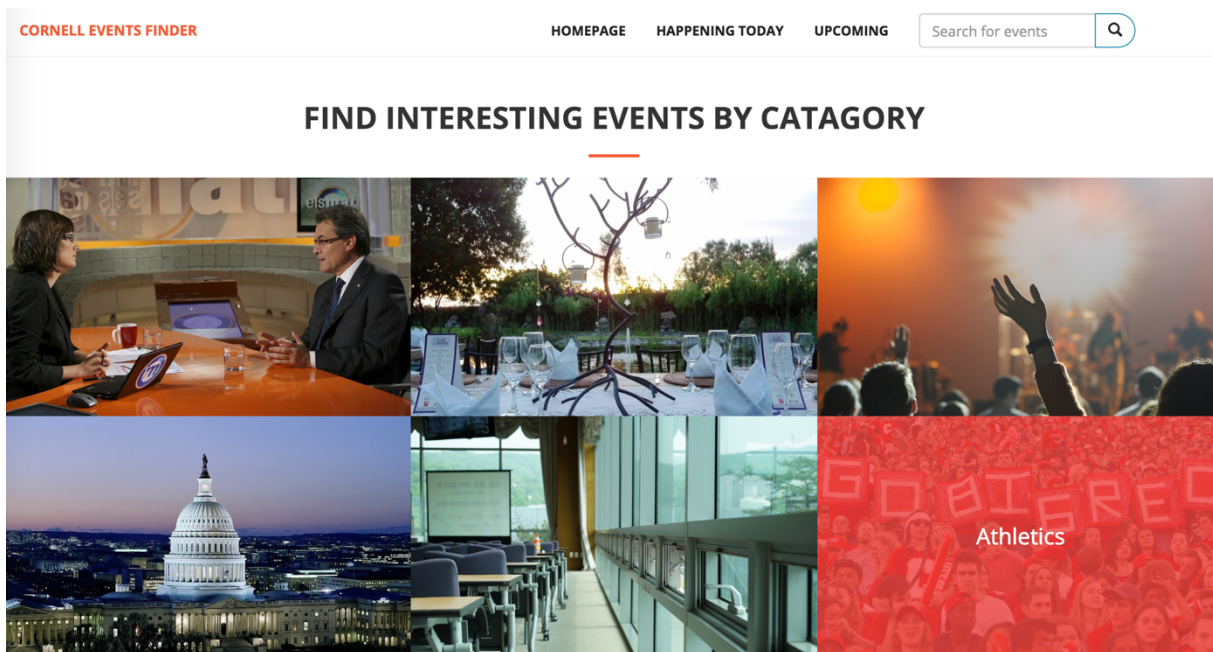


Figure 2.3 Homepage at events categories

At the very top of the homepage, we use a landscape music festival image to impress our user. It can post the events with the red button in the middle. Then if the user scroll down a little, it will find the popular events screen where we display the most viewed events. At the bottom of the homepage, we have six thumbnails image each represent a different category. The popular events tab and the category tabs has animation effects when the mouse is hovering above them. The top navigation is fixed at the top of the website, so the redirection to

“Happening Today”, “Upcoming” and the search bar will be always available no matter where the user has scroll to. This design makes the navigation bar much more accessible than letting it scroll.

Support different size of devices

As a “mobile-first front end framework” bootstrap support mobile size display natively. With some effort, the website won’t screw up when display on different size of screen even on mobile device.

- Specifying how many sections to display on different size of devices.

```
<div class="col-lg-4 col-md-4 col-sm-4 col-xs-6 popular-item">
```

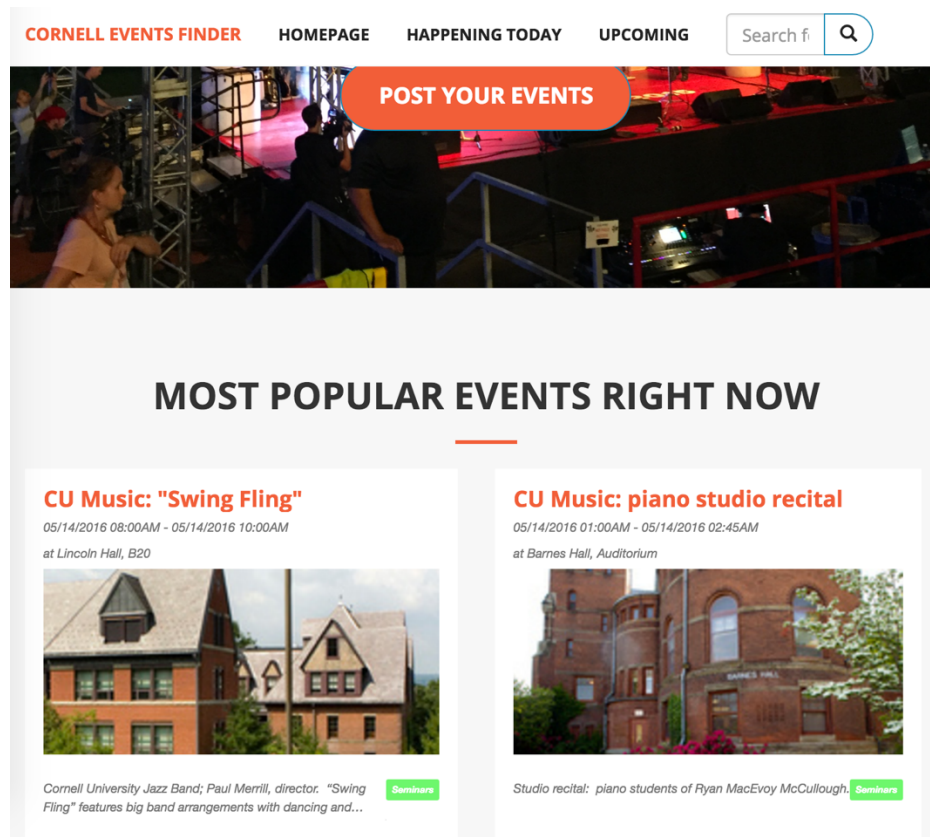


Figure 2.4 Homepage on extra small size device

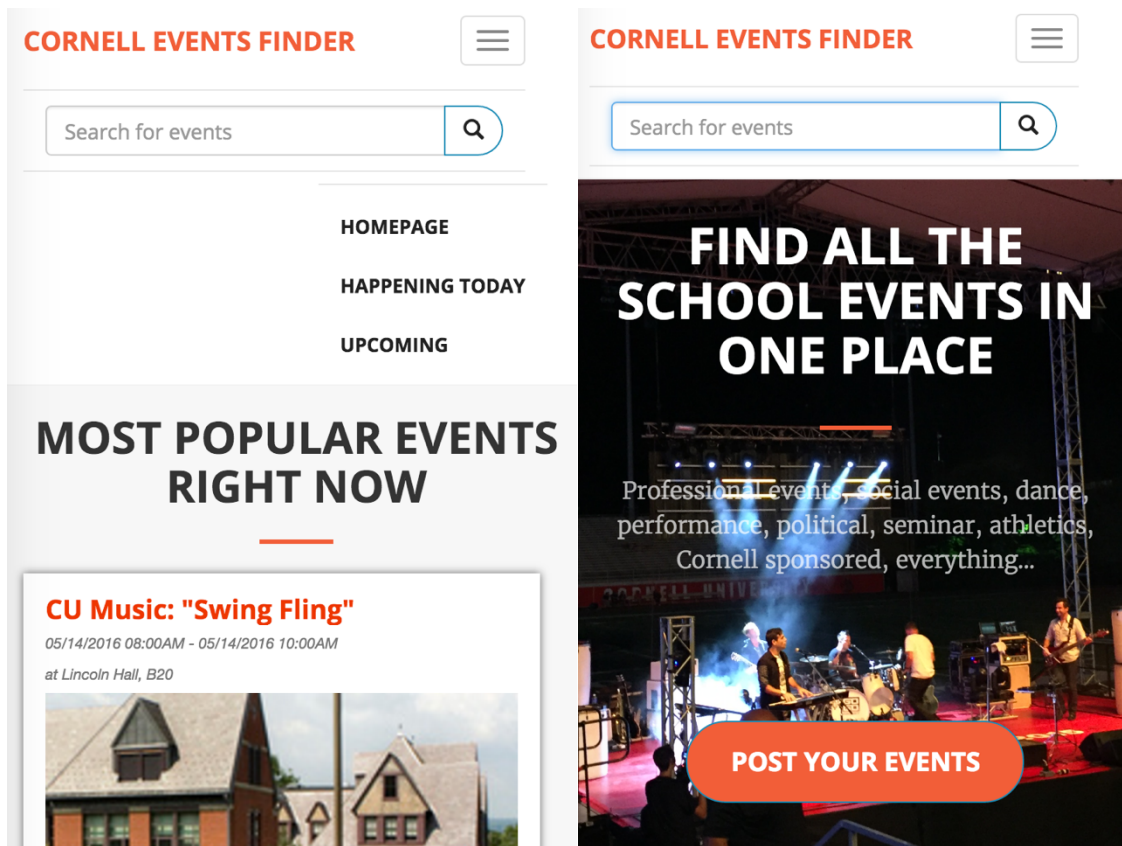


Figure 2.5-6 Homepage resized for mobile phone

There are three sections per row on the large screen devices, but there would only be two on extra small screen device, and only one section per row when displaying on the mobile phones.

The navigation bar would collapse into a drop down button if specifying it to be navbar

```
<nav id="mainNav" class="navbar navbar-default navbar-fixed-top">
```

Handlebars

Handlebars is used to feed data into a html template division. When building a list similar divisions, it would be very handy to have handlebars to help us reuse the code. It makes our code much more concise and easy to maintain.

```

<div class="row">
  {{#each popular}}
    <div class="col-lg-4 col-md-4 col-sm-4 col-xs-6 popular-item">
      <a href="#" class="event-click hvr-glow" data-internalid="{{id}}">
        <div class="popular-caption popular-link">
          <h4 class="title">{{title}}</h4>
          <p class="text-muted popular-time">{{startingTime}} - {{endingTime}} </p>
          <!-- <p class="text-muted popular-tag"> tag </p> -->
          <p class="text-muted popular-location">at {{location}}</p>
          <div class="container">
            
          </div>
          <p class="text-muted description-text"><br><span class="label label-default tag-label" style="background-color: {{categoryColor}}">{{category}}</span>{{description}}</p>
        </div>
      </a>
    </div>
  {{/each}}
</div>

```

Figure 2.7 Frontend: create a html template and specify handlebars field.

When rendering this page, we create a json object named the “popular” with corresponding fields (title, startingTime, endingTime, location, ...) and user res.render to send it with the html page to the browser. Here “category” is read from file system, and “popular” is fetched from Firebase.

```

// Get all of our popular data, category data and concat them to a single json object
// read from file.
var category = require('../data/category.json');
var Firebase = require("firebase");
var processData = require('./processData.js');
// var eventRef = new Firebase("https://event-finder.firebaseio.com/events");
var eventRef = new Firebase("https://event-finder-test.firebaseio.com/events");

// copy all the objects from o2 to o1.
function jsonConcat(o1, o2) {
  for (var key in o2) {
    o1[key] = o2[key];
  }
  return o1;
}

// fetch from firebase.
exports.view = function(req, res){
  eventRef.orderByChild("numberOfViewed").limitToLast(6).once("value", function(snapshot) {
    var popular = {
      "popular": []
    };
    snapshot.forEach(function(data) {
      var tmp = processData.processSearchListData(data);
      // put the most viewed element to the front
      popular["popular"].unshift(tmp);
      // console.log("time of View: " + data.val().numberOfViewed);
    });
    var data = {};
    data = jsonConcat(data, popular);
    data = jsonConcat(data, category);
    res.render('index', data);
  });
};

```

Figure 2.8 Backend: populate json objects and send them with the html page.

An interactive map view

Nothing can be more intuitive than a map to help user find the location of events. Inspired by the Airbnb website https://www.airbnb.com/s/New-York--NY?alsm=1&s_tag=fj7ab4vU, we built an interactive map. When the mouse is hovering above the events, the right-hand-side map would automatically change its center to the location of that event, the marker for that event would turn red, and the title and location of the event would be displayed above the marker. When clicked on a marker, all the events at that location would be displayed in a list.

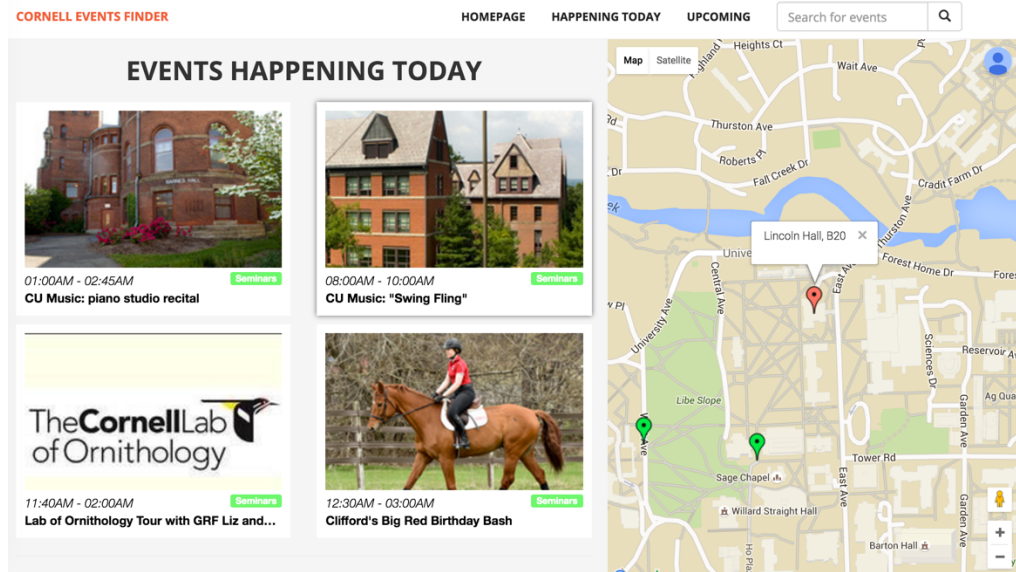


Figure 2.9 Hovering above an event.

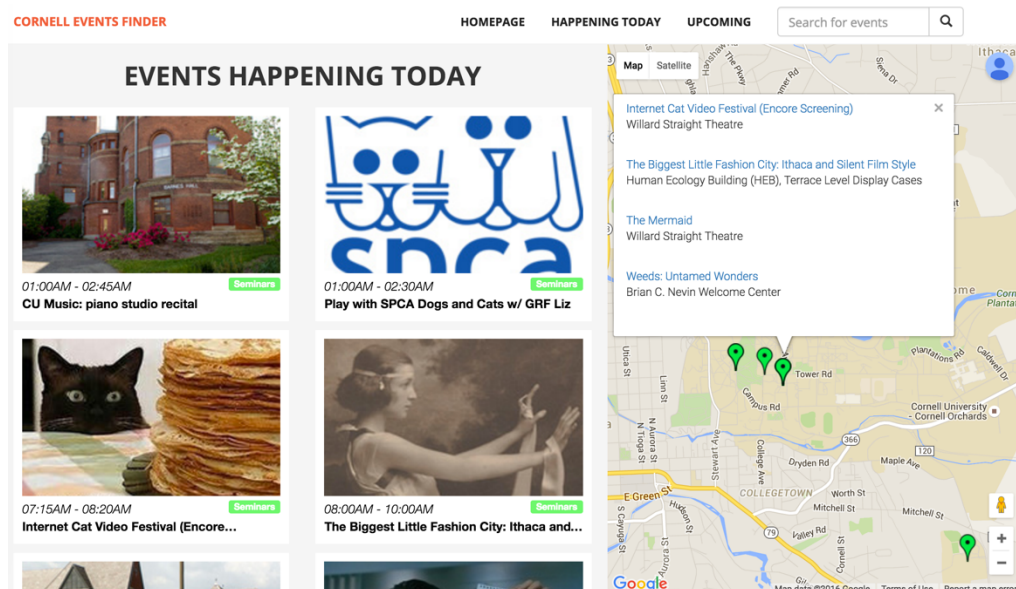


Figure 2.10 Click on a marker on map.

The hovering effects are implemented with jQuery. jQuery is a JavaScript library helpful when manipulating html elements. The right hand side map is implemented with Google Map API, it has marker, information window, and allow us to add listeners to the markers.

```
$( "#" + data.key() ).mouseenter( handlerIn ).mouseleave( handlerOut );
    }
  });
});
}

function handlerIn() {
  var id = $(this).attr('id');
  console.log("in " + id);
  var location = locations[id]
  var markerID = location.lat + ";" + location.lng;

  markers[markerID].setIcon("http://maps.google.com/mapfiles/ms/icons/red-dot.png");
  infowindows[id].open(map, markers[markerID]);
  map.panTo(locations[id]);
  for (var key in infowindowforclicks) {
    infowindowforclicks[key].close();
  }
  map.setZoom(16);
}
```

Figure 2.11 Add hovering effects with jQuery.

```
function makeInfoWindowEvent(map, infowindow, marker, markerID) {
  var identifier = google.maps.event.addListener(marker, 'click', function() {
    for (var key in infowindowforclicks) {
      infowindowforclicks[key].close();
    }
    infowindow.open(map, marker);
  });
  listenerIdentifiers[markerID] = identifier
}
```

Figure 2.12 Add events listener for the markers.

Simple Form to Post Event

The users are able to post their events after clicking the “Post” button on the homepage. When posting the events, the user could specify the events title, time and location, whether the events is restricted to a specific group of people and upload some images for the events. JavaScript would check if the input data are legal. If so, the information would be posted into the Firebase and an alert would pop up and help user to redirect to the homepage.

NEW EVENT
Add a new event that you host.

CTAS Night Market 2015

Physical Sciences Bldg, East Avenue, Ithaca, NY, United States

11/07/2015 8:00 PM

11/07/2015 10:00 PM

Social Events

Restrict to Specific Attendants? *

☐ Cornell Sponsored ☒ No Charge ☒ Free Food

Choose Files 2 files

GET READY FOR THE CORNELL TAIWANESE AMERICAN SOCIETY'S BIGGEST EVENT OF FALL SEMESTER!

Are you missing Taiwan and its lively night markets? Craving homemade Taiwanese food, dessert, and drinks? Never even heard of a Night Market but curious?

Then come out to NIGHT MARKET 2015 for a night of delicious FREE food, games, and amazing company.

Featuring dishes such as:
Minced Pork Rice
Cold Sesame Noodles
Thank You Chicken

POST MY EVENT

Figure 2.13 Post a New Event.

We use data-time-picker from <https://github.com/jdewit/bootstrap-timepicker/> to pick time and date. It includes the styling sheets and the JavaScript functions for the dropdown and selection actions of the form. The image would be uploaded to our AWS S3 bucket and a secure (with https prefix), public visible URL will be store together with other information of the events into the Firebase database.

```
function upload_file(file, signed_request, url, json, currentAccount){
  console.log("signed_request")
  console.log(signed_request)
  var xhr = new XMLHttpRequest();
  xhr.open("PUT", signed_request);
  xhr.setRequestHeader('x-amz-acl', 'public-read');
  xhr.onload = function() {
    if (xhr.status === 200) {
      // alert("uploaded file " + url);
      json["imageOfEvent"].push(url);

      // after retrieved the image URL, store the event information into Firebase.
      storeJsonFile(json, currentAccount);
    }
  };
  xhr.onerror = function() {
    alert("Could not upload file.");
  };
  xhr.send(file);
}
```

Figure 2.14 Upload image to AWS S3 and then store events to Firebase in the callback.

One Result List Template for All

We provide user several ways to filter out the events they want, including by categories, by keywords and by how soon the event will happen. We use a same template populated with searching result to display the events user what. This allow use to reuse the front-end code and make the program easy to maintain.

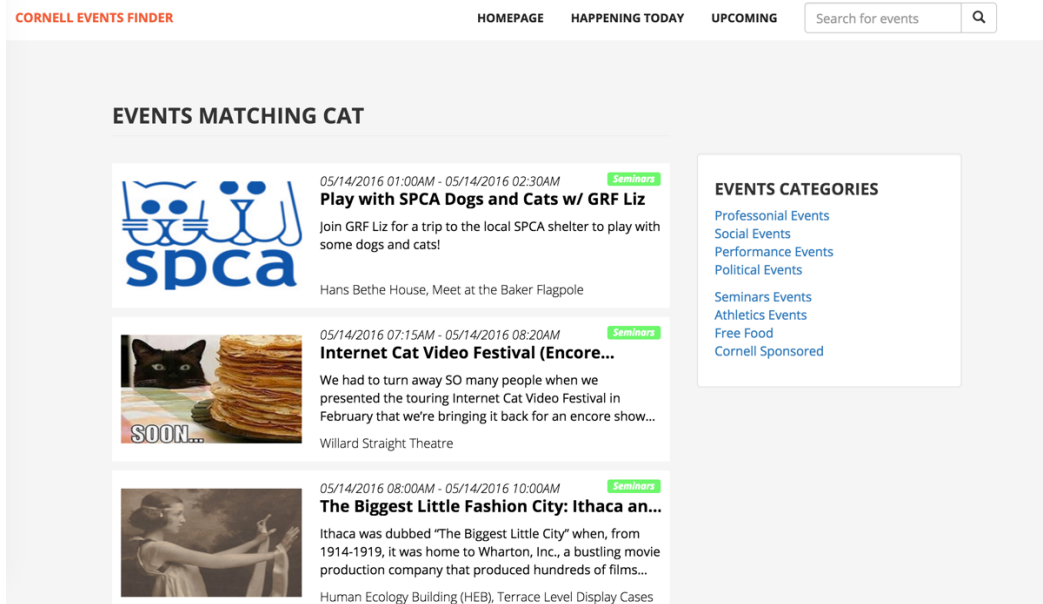


Figure 2.15 Searching for keyword “cat”.

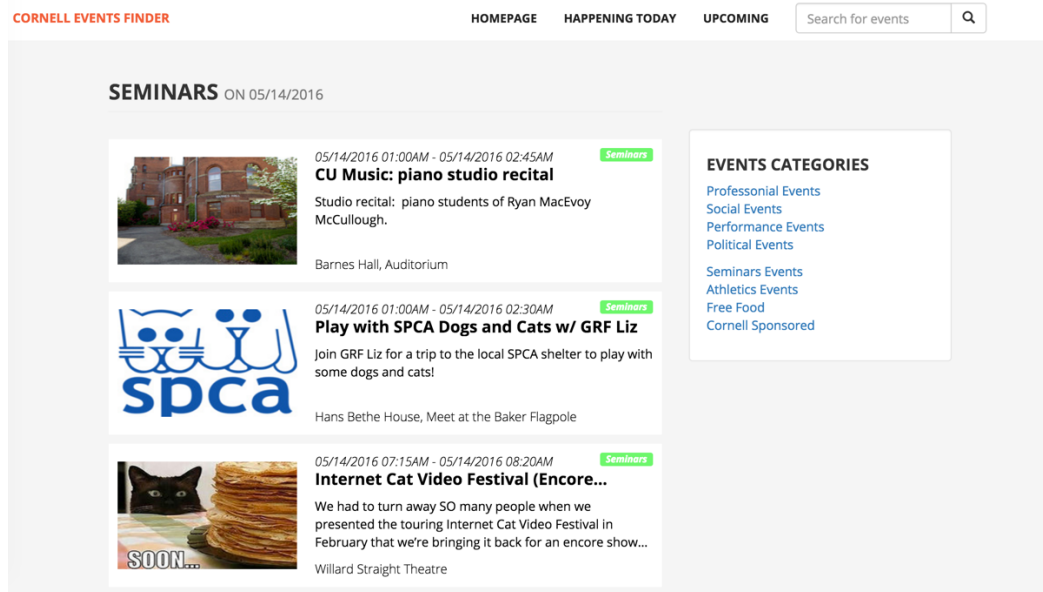


Figure 2.16 Searching for category “Seminars”

iOS App

Though small perfectly formed

Unlike having a whole big display of the website, the iOS app we build is aiming at provide the whole world on a little tiny screen of a smart phone for users. Therefore, the UX and UI is what we are concerned a lot, as well as the actual functionalities of the app.

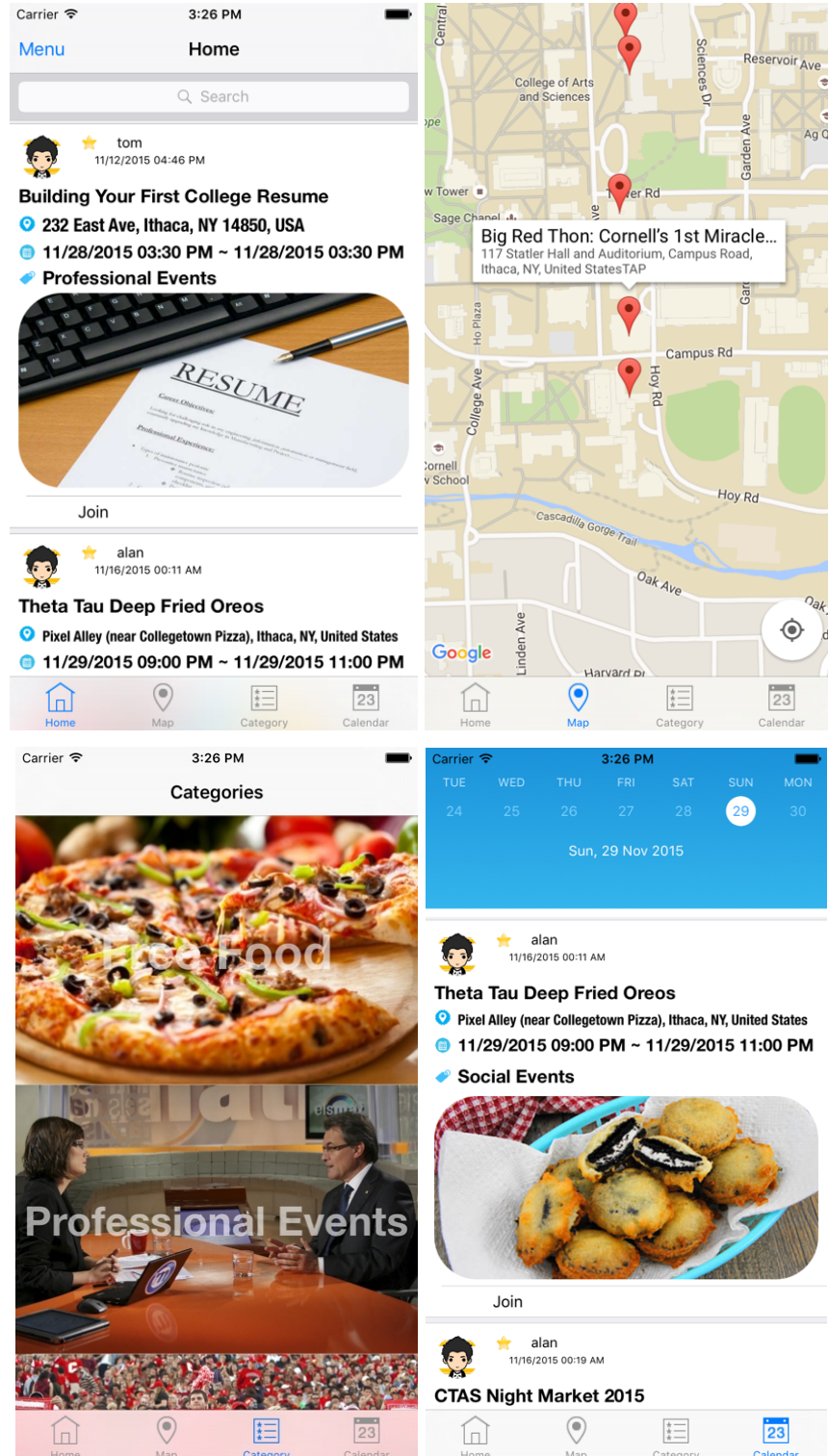


Figure 3.1 Home, Map, Categories and Calendar Views

As shown in Figure 2.17, there are four main views in the app --- Home, Map, Categories and Calendar Views. And each of these four views is built with its own functional purpose.

Home view aims to provide all of the events that currently are happening on campus, with the function that for users to search for these events by the keywords of the title or contents. Map view could provide a better view of the events with the accurate locations shown with red annotations on the map, a tiny callout view with specific information will pop up when users tapping on the little red marker. Categories view is designed for users who would like to search for events by their categories, like free food, professional, social, etc., which saves tons of time for people who want to participate in certain type of event. Furthermore, the calendar view collects all the events by their dates and show users with a clearer timeline of the events that are past, currently happening or will be held in the next few days.

To implement the searching function, we use NSPredicate embedded within Objective-C to filter out the events users want by keywords of the title or contents. Figure 2.18 shows how this function is implemented.

```
314 - (void)filterContentForSearchText:(NSString*)searchText
315 {
316     NSPredicate *resultPredicate1 = [NSPredicate predicateWithFormat:@"nameOfEvent contains[c] %@", searchText];
317     NSPredicate *resultPredicate2 = [NSPredicate predicateWithFormat:@"introOfEvent contains[c] %@", searchText];
318     NSArray *subPredicates = @[resultPredicate1, resultPredicate2];
319     NSCompoundPredicate *resultPredicate = [NSCompoundPredicate orPredicateWithSubpredicates:subPredicates];
320     searchResults = [events filteredArrayUsingPredicate:resultPredicate];
321 }
322
```

Figure 3.2 Events filtering function implementation

For the Calendar view, in order to filter the events with specific dates, we come up with the following methods, shown in Figure 2.19. By predicate the events with format like “starting time string contains ...” and “ending time string contains ...” we could easily filter out the events of interest.

```
89 - (void)filterEventsToDate: (NSString *)dateNumber{
90     NSPredicate *startingTimePredicate = [NSPredicate predicateWithFormat:@"startingTimeString CONTAINS[c] %@",
    dateNumber];
91     NSPredicate *endingTimePredicate = [NSPredicate predicateWithFormat:@"endingTimeString CONTAINS[c] %@",
    dateNumber];
92     NSArray *subPredicates = @[startingTimePredicate, endingTimePredicate];
93     NSCompoundPredicate *resultPredicate = [NSCompoundPredicate orPredicateWithSubpredicates:subPredicates];
94     dateEvents = [events filteredArrayUsingPredicate:resultPredicate];
95     [self.myTableView reloadData];
96 }
```

Figure 3.3 Methods to filter events with dates

Detailed Event Information View

To provide the users with better view of the events they would like to take part in, we also create a detailed information view for all of the events shown in the Home View, Map View, Categories View and Calendar View as well. Users can just tap on the cell on the views mentioned above to navigate to another view, which is the detailed information view of such event, shown below in Figure 2.20.

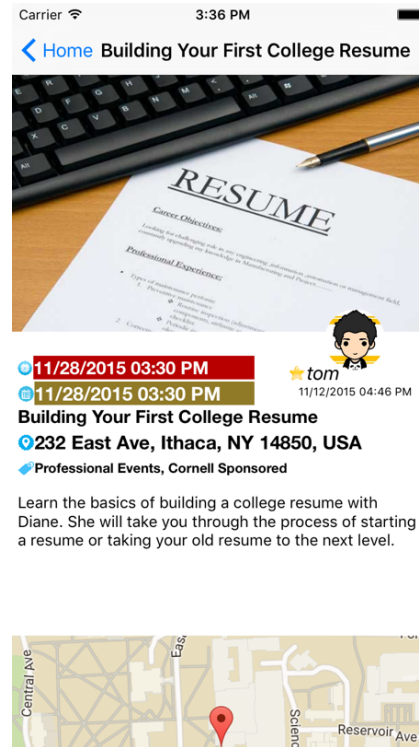


Figure 3.4 Detailed information view for event

In the detailed event information view, from top to bottom, left to right, there are image of the event, starting time, ending time, author profile image, author name, dates posted, title, location, categories, introduction and the map view. All necessary information of the event will be listed here for users to go through.

Side Menu Design and Event Post Management

Just as the side menu in Facebook and lots of other social apps, we also design and implement the side menu in our app, in order to offer better user experience and more functionalities to users. We reference a side menu template here at <https://github.com/romaonthego/RESideMenu> The side menu in our app is shown below in Figure 2.21.

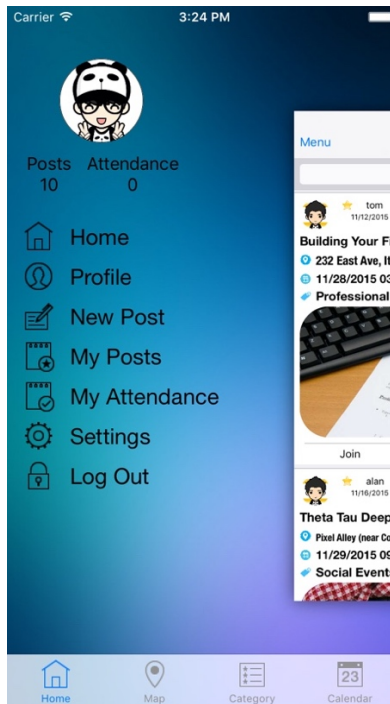


Figure 3.5 Side menu design

Within the side menu, users are allowed to edit their profile image and navigate to different views, which consists of d “Home”, “Profile”, “New Post”, “My Posts”, etc.

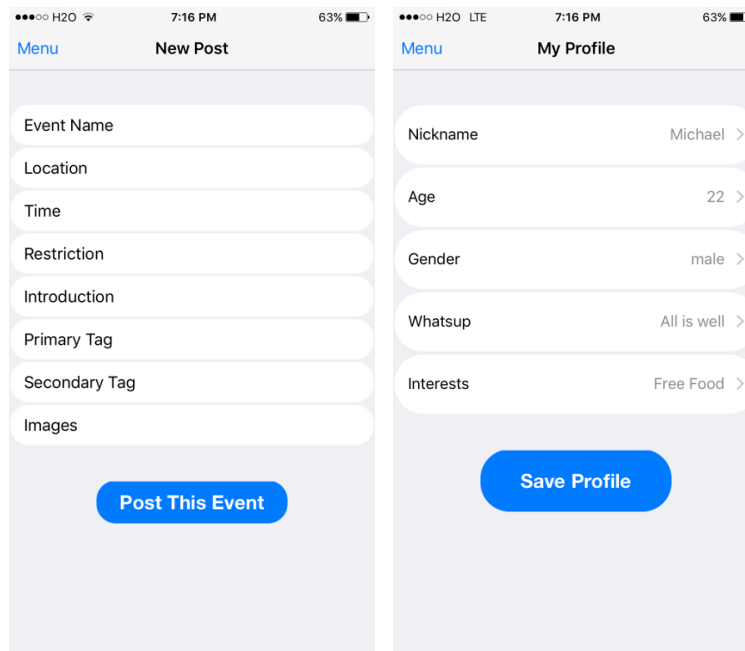


Figure 3.6 New post (left) view and profile view (right)

Figure 2.22 above shows the “New Post” view and “Profile” view that can be reached from the side menu. These information cells are well-organized in the table view in order for a clear and tidy vision from the user side. And they are clipped into ellipse and these two buttons are colored in blue for a better user interface and experience as well.

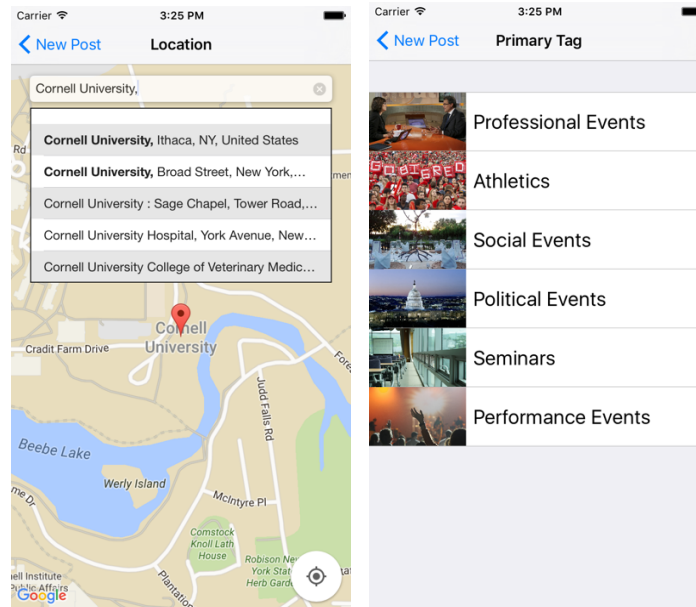


Figure 3.7 Location selection view (left) and event tag allocation view (right)

Within the “New Post” table view, there are several cells for event organizers to add information and select the location or tags for their events. Two pictures shown in Figure 2.23 above illustrate this part well. For the location selection view, we reference some source codes here at

<https://github.com/mrugrajsinh/MVAutocompletePlaceSearchTextField> in order to provide the places auto-completion function for event organizers when they try to type the location in the text field on the map view. This function we refer to will automatically complete the locations when the text field starts being edited. And we also add another function that allows the map view to show the little red marker when users tap on certain location on the pull-down list below the text field. Partial codes of the draw markers function are shown below.

```

103 -(void)drawMarkers: (GMSPlace*) place{
104     [self.mapView clear];
105     GMSMarker *marker = [[GMSMarker alloc] init];
106     marker.title = [place name];
107     marker.snippet = [place formattedAddress];
108     marker.appearAnimation = kGMSMarkerAnimationPop;
109     marker.position = [place coordinate];
110     marker.map = self.mapView;
111     longitude = place.coordinate.longitude;
112     latitude = place.coordinate.latitude;
113     locationOfEvent = place.name;
114     [self.mapView animateToLocation:place.coordinate];
115 }

```

Figure 3.8 Draw markers on the map view

For the event tag selection view on the right of Figure 2.23, event organizer can choose the primary tag, and secondary tag as well (the view for choosing secondary tags is not shown in Figure 2.23) for their events. And this is of great help for users who would like to search for events by their categories.

Data Management

The most significant part of the codes would be the “Data Management”, which controls the uploading or downloading events, uploading or downloading user information, etc. Below Figure 2.25 shows the header file of “Data Management”.

```
9  #import <Foundation/Foundation.h>
10 #import "MyEventInfo.h"
11 #import "MyUserInfo.h"
12 #import <UIKit/UIKit.h>
13 #import "Cloutinary/Cloutinary.h"
14
15 @interface MyDataManager : NSObject <CLUploaderDelegate> {
16
17 }
18 |
19 + (void)saveEvent:(MyEventInfo *) event withViewController:(UIViewController *) thisVC;
20 + (NSMutableArray *)fetchEvent;
21 + (void)saveUser:(MyUserInfo *) user;
22 + (MyUserInfo *)fetchUser:(NSString *) username;
23 + (void)saveParticipantToEvent:(MyEventInfo *)event withUser:(MyUserInfo *)user;
24 + (void)removeParticipantFromEvent:(MyEventInfo *)event withUser:(MyUserInfo *)user;
25 + (void)updateUser:(MyUserInfo *) user;
26 + (void)updateUser:(NSString *) username postsNumber:(NSNumber *) myPostsNumber;
27 + (void)updateUser:(NSString *) username usrProfileImg:(NSString *) usrProfileImg;
28 + (void)saveEventsToUsrDefaults:(NSArray *)events;
29 + (NSArray *)extractEventsFromUsrDefaults;
30
31 @end
```

Figure 3.9 Data management header file

For better illustration, the specifications of these methods above are listed below:

- (void)saveEvent:(MyEventInfo *) event withViewController:(UIViewController *) thisVC;

Upload event posted by users to online database, in this case we use Firebase.

- (NSMutableArray *)fetchEvent;

Fetch all the events from the online database.

- (void)saveUser:(MyUserInfo *) user;

Save new user information up to the online database.

- (MyUserInfo *)fetchUser:(NSString *) username;

Fetch user information from Firebase with the username

- (void)saveParticipantToEvent:(MyEventInfo *)event withUser:(MyUserInfo *)user;

Add participants to certain event and upload information onto Firebase.

- (void)removeParticipantFromEvent:(MyEventInfo *)event withUser:(MyUserInfo *)user;

Remove participants from certain event and update the event data on Firebase.

- (void)updateUser:(MyUserInfo *) user;

Update user information on Firebase with the local updated user information.

- (void)updateUser:(NSString *) username postsNumber:(NSNumber *) myPostsNumber;

Update user posts number on Firebase with local data.

- (void)updateUser:(NSString *) username usrProfileImg:(NSString *) usrProfileImg;

Update user profile image on Firebase with local chosen image.

- (void)saveEventsToUsrDefaults:(NSArray *)events;

Save events fetched from Firebase into local database called userdefault.

- (NSArray *)extractEventsFromUsrDefaults;

Extract events from local database.

Additionally, for image uploading, we use a service called “Cloutinary” that can store the actual image data and then return a URL linked with that image. Therefore, when users uploading images onto the database, what actually stored in Firebase is not the image data itself, but the URL linked to it instead, saving lots of space.

Arcade Machine



Figure 4.1 Encapsulated Controller of the Arcade Machine

The arcade machine is controlled with a joystick and two buttons. The control signal goes into the raspberry pi (on the right side of the box). We connect a Wi-Fi module with the raspberry pi so that it can download the events information from our database. As for the display, we connect a monitor with the raspberry with a DVI → HDMI cable.

Pygame Display Implementation

We use pygame library based on Raspberry Pi to display events on an external screen. Main part of the codes used to display events are shown in Figure 5.1. First we call `updateDisplayBuffer` method to update the display buffer, then we use `updateDisplay` method to actually update the display. The variable `posY` in `updateDisplay` method is used for scrolling the event view up and down. And the subsurface is chosen to show which part of the whole display buffer of the current event.

```

144 def updateDisplay():
145     global buffer_surface
146     global posX
147     global posY
148     screen.fill((0,0,0))
149     if posY < 0:
150         posY = 0
151     if posY > SCREEN_HEIGHT - DISPLAY_HEIGHT:
152         posY = SCREEN_HEIGHT - DISPLAY_HEIGHT
153     screen.blit(buffer_surface.subsurface(posX,posY,DISPLAY_WIDTH,DISPLAY_HEIGHT),(0,0))
154     pygame.display.flip() # update the display
155
156 def updateDisplayBuffer():
157     global buffer_surface
158     global events_arr
159     global cur_event_idx
160     print "cur_event_idx", cur_event_idx
161     print "cur_img_idx", cur_img_idx
162     buffer_surface.fill((0,0,0))
163     next_y = addTitleToSurfaceMiddleAlign(buffer_surface, events_arr[cur_event_idx].getTitle(), 0)
164     next_y = addTimeAndLocationToSurfaceMiddleAlign(buffer_surface, events_arr[cur_event_idx].getTime(),
165                                                     events_arr[cur_event_idx].getLocation(), next_y)
166     next_y = addImageToSurfaceMiddleAlign(buffer_surface, next_y)
167     addDescriptionToSurfaceMiddleAlign(buffer_surface, events_arr[cur_event_idx].getDescription(), next_y)
168

```

Figure 5.1 Codes to display events with pygame

We also refer to the source codes here at <https://github.com/ryanrouleau/bouncing-balls-with-pygame> and modify it to a airplane game for users to play while they are searching for events. This game can be run with the event view as the background. Airplane is controlled by users using the joystick to dodge the rocks flying around on the screen.

```

133 def update_all(self):
134     self.mouseXchange = self.mouseX - PLAYER_SIZE[0]/2
135     self.mouseYchange = self.mouseY - PLAYER_SIZE[1]/2
136     self.timeObject.update()
137     self.time = self.timeObject.getTime()
138
139     self.ballData.addBall(self.time)
140
141     self.gameDisplay.blit(self.bg_surface, (0,0))
142     self.playerPosUpdate(self.mouseXchange, self.mouseYchange)
143     self.displayScoreText(self.time, "current")
144
145     for ball in self.ballData.getBallList():
146         if collisionDetection(ball.getPos(), self.mouseX, self.mouseY) == True:
147             #----Game over-----
148             self.gameDisplay.blit(self.bg_surface, (0,0))
149             self.gameOverText()
150             self.displayScoreText(self.time, "current")
151             self.pygame.display.update()
152             self.done = True
153             break
154         else:
155             ball.update()
156
157     self.pygame.display.update()
158     self.clock.tick(60)

```

Figure 5.2 Codes for update game view

The codes above in Figure 5.2 shows how the game view is updated, the mouseXchange and mouseYchange are variables used to determine where to draw the image of the airplane. The for loop aims to draw each rock separately on the screen at different positions. Since every time before we call this update_all function, we call updateDisplay to refresh the background view containing the event information, it seems that the game is run with the event view as the background.

Implementation of the Python Web Crawler

Fetch and Parse Events with Python

User and groups can post their own events from the websites and the iOS app. For the events sponsored by Cornell, we can find them from Cornell event website <https://events.cornell.edu>. It is a stable source of events, so we want to build a crawler to fetch events from it.

We build the Python web crawler using urllib, and BeautifulSoup. Urllib is used to open URL and BeautifulSoup provides handy functions to parse HTML elements. Observe the HTML structure of the website then we can extract the field we want and store them.

```
# get information from each events detail pages.
for item in toVisit:
    tmp = {}
    url = item
    html = urllib.urlopen(url).read()
    soup = BeautifulSoup(html, "html.parser")
    event = soup.select("div.box_header.vevent")[0]

    title = event.h1.span.get_text().strip()
    time = ""
    for abbr in event.h2.findAll('abbr'):
        time += abbr.get_text().strip() + " "
    print time
    time = timePreprocess(date, time)
    print time
    if time is None:
        print("can't resolve time, discard this event")
        continue
    display_location = event.h3.a
    real_location = event.h3.small
```

Figure 6.1 Extract HTML field with beautiful soup.

Some of the events on the website has some missing field. We decide to discard such events.

Geocode the locations

The location on the website is being displayed as location name instead of latitude and longitude. To get the latitude and longitude, we use Geocoding functions provided by Google Map. To make our geocoding method more accurate, we had to append the city name, state name, country name and zip code to the end of the

location. Based on our experience of practice, adding “Cornell University” would make the event being geocoded to the School coordinate, instead of the building coordinate, so we won’t want to feed “Cornell University” into the geocoding function.

```
if "Cornell University" in real_location:
    real_location = real_location.replace("Cornell University", "")
    print "real_location becomes: " + real_location

if "Ithaca, NY 14853, USA" not in real_location:
    real_location = display_location + ", Ithaca, NY 14853, USA"
```

Figure 6.2 Preprocess the address before feeding it to Google Map API

```
# Taken from:
# http://stackoverflow.com/questions/15285691/googlemaps-api-address-to-coordinates-latitude-longitude
def decodeAddressToCoordinates( address ):
    print "address: %s" % address
    import urllib
    import urllib2
    import StringIO
    import json
    urlParams = {
        'address': address,
        'sensor': 'false',
    }
    url = 'http://maps.google.com/maps/api/geocode/json?' + urllib.urlencode( urlParams )
    response = urllib2.urlopen( url )
    responseBody = response.read()

    body = StringIO.StringIO( responseBody )
    result = json.load( body )
    if 'status' not in result or result['status'] != 'OK':
        return None
    else:
        return {
            'lat': result['results'][0]['geometry']['location']['lat'],
            'lng': result['results'][0]['geometry']['location']['lng']
        }
```

Figure 6.3 Geocode the location

Decide the Category for Fetched Events

We asked user to choose a category among six when posting a event. For the events fetched by the Python web crawler, we also need an algorithm to classify them into one of these categories. One way to do that is using machine learning to learn the pattern of how user choose the category for their events, and use the learnt model to do classification. However, we can’t build a powerful classification machine due to the lack of training data. We will discuss the efforts we had made in the “Problems We Encountered” section.

We end-up-with a keyword-matching classifier. We simply check if the title or description of the events contains some keywords, and classify the events to the category accordingly.


```

tags_list = ["Professional Events", "Social Events", "Performance Events",
             "Political Events", "Seminars", "Athletics"]

keywords_list = [
    ["professional", "scholarship", "internship"],
    ["social", "lunch", "dinner", "club", "sponsored", "coffee", "camp"],
    ["performance", "music", "exhibition", "museum", "art", "theatre", "fashion", "show"],
    ["politic", "policy", "mtg"],
    ["seminar", "research", "colloquium", "lecture"],
    ["sport", "yoga", "ball"]]

# fed the title, location and description of the events to the classifier
# it returns the primary tag of the event.
def classify(text):
    import re
    for i in range(6):
        for j in range(len(keywords_list[i])):
            if re.match("(^.*\s"+keywords_list[i][j]+"[^a-z]+.$)|(^.*\s"+keywords_list[i][j]+"[^a-z]*$)|(^.*\s"+keywords_list[i][j]+"$)":
                return tags_list[i]
    return tags_list[4]

```

Figure 6.4 Keyword Matching Classifier to Decide Categories.

Problems we encountered Image Management Service

We had tried three different ways to store our image. At first, we encode the image file into base64 string and just store the string into Firebase. However, it could cause our website and app becomes very slow. Since Firebase is a NoSQL database, we had de-normalized the database schema and all the information of events are stored in the same directory. When downloading the title, location, time and other fields of the event, we had to download the huge encoded image string. Also, since we didn't compress the image before uploading it, all the image is in their origin size. Nowadays, mobile phones have a high pixel's camera, so a typical image user upload could be more than 2 MB. This problem made our website and iOS app very slow.

Our first optimization is to compress the image before uploading them. It did make our application run faster, but it is not a best approach because we still can't first display the text information of the event, then asynchronously download the image. Also, since we need to display the image on mobile device (small screen) and on computer (large screen), we don't know how much shall we compress the images.

Taking that into consideration, we turn to cloud image management service. We first use AWS S3 to store our image, and only store the URL of the image into Firebase. With this approach, we can first download the URL with all the other information of the events, and then fetch the image asynchronously from S3. But we still do not know how shall be compress the image and store it into S3 bucket.

Finally, we found a cloud image service called Cloudinary. We can upload image in its original size. When we want to download the image, we can specify the URL with size parameter, then Cloudinary will resize the image to the specified size for download. With this handy service, we can store image in one place, and use it on different platform by specifying different size parameters.

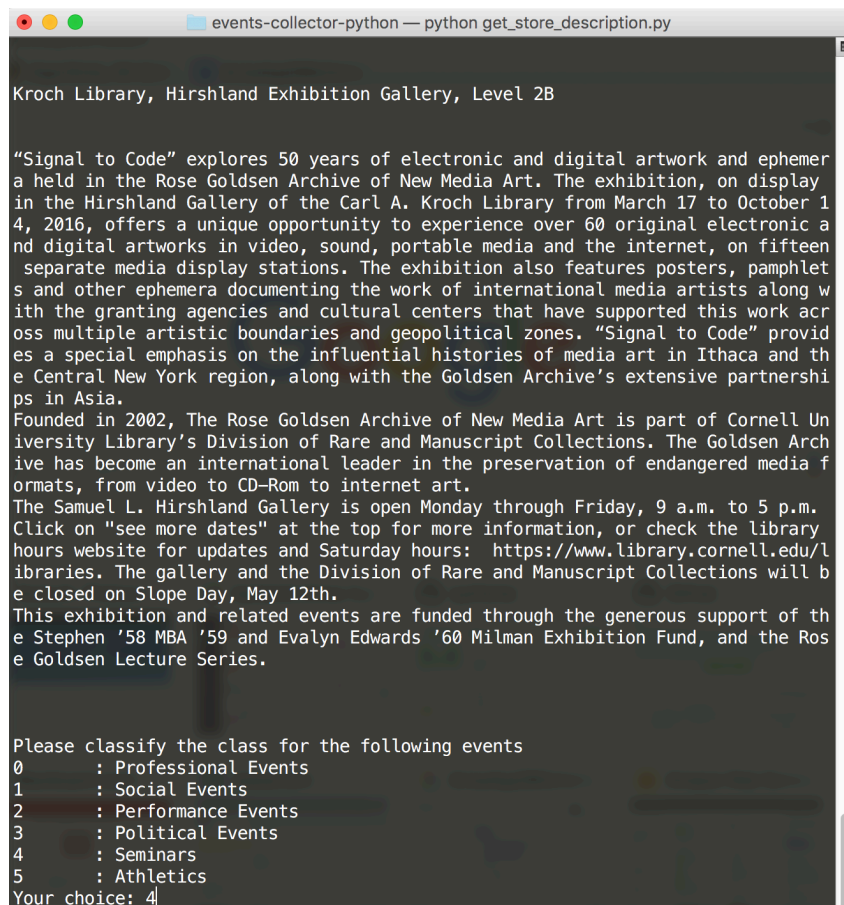
```
<!-- http://cloudinary.com/documentation/django_image_manipulation| -->
<img src='http://res.cloudinary.com/demo/image/upload/c_fill,h_150,w_100/sample.jpg'
      height='150' width='100'/>
```

Figure 7.1 Specifying size parameter in the URL.

Classifying Categories for the Crawled Events

As described in “Implementation of the Python Web Crawler”, we need to assign a category for each fetched events. We had described the work-around we end-up-with, the keywords matching classifier. However, that is not very accurate because some social events could have keyword “basketball” in its description, which would make it be classified into “Athletic Events”. Also, there is only a very limited amount of keywords can be recognized and put into the keywords pool by human. A better way to solve the problems is to use machine learning to learn the pattern of how user label their events into different categories. In this section, we want to talk about the efforts we had put when trying to incorporate machine learning to our project. Also, we will analyze the reason why we failed.

The data we use to train our machine learning model with events on <https://events.cornell.edu>. We use python script to crawl the events, and create a command line interface to ask for a label (category) before storing them to a file.



```
events-collector-python — python get_store_description.py

Kroch Library, Hirshland Exhibition Gallery, Level 2B

"Signal to Code" explores 50 years of electronic and digital artwork and ephemera held in the Rose Goldsen Archive of New Media Art. The exhibition, on display in the Hirshland Gallery of the Carl A. Kroch Library from March 17 to October 14, 2016, offers a unique opportunity to experience over 60 original electronic and digital artworks in video, sound, portable media and the internet, on fifteen separate media display stations. The exhibition also features posters, pamphlets and other ephemera documenting the work of international media artists along with the granting agencies and cultural centers that have supported this work across multiple artistic boundaries and geopolitical zones. "Signal to Code" provides a special emphasis on the influential histories of media art in Ithaca and the Central New York region, along with the Goldsen Archive's extensive partnerships in Asia.

Founded in 2002, The Rose Goldsen Archive of New Media Art is part of Cornell University Library's Division of Rare and Manuscript Collections. The Goldsen Archive has become an international leader in the preservation of endangered media formats, from video to CD-Rom to internet art.

The Samuel L. Hirshland Gallery is open Monday through Friday, 9 a.m. to 5 p.m. Click on "see more dates" at the top for more information, or check the library hours website for updates and Saturday hours: https://www.library.cornell.edu/libraries. The gallery and the Division of Rare and Manuscript Collections will be closed on Slope Day, May 12th.

This exhibition and related events are funded through the generous support of the Stephen '58 MBA '59 and Evalyn Edwards '60 Milman Exhibition Fund, and the Rose Goldsen Lecture Series.

Please classify the class for the following events
0 : Professional Events
1 : Social Events
2 : Performance Events
3 : Political Events
4 : Seminars
5 : Athletics
Your choice: 4
```

Figure 7.2 Asking for a Category for Each Fetched Event

We keep running the python script to fetch, parse, ask for label from user and store the events. In this way, we spent an hour to get 202 events happening in April 2016 with their labels. We store these ready to be processed data in a file.

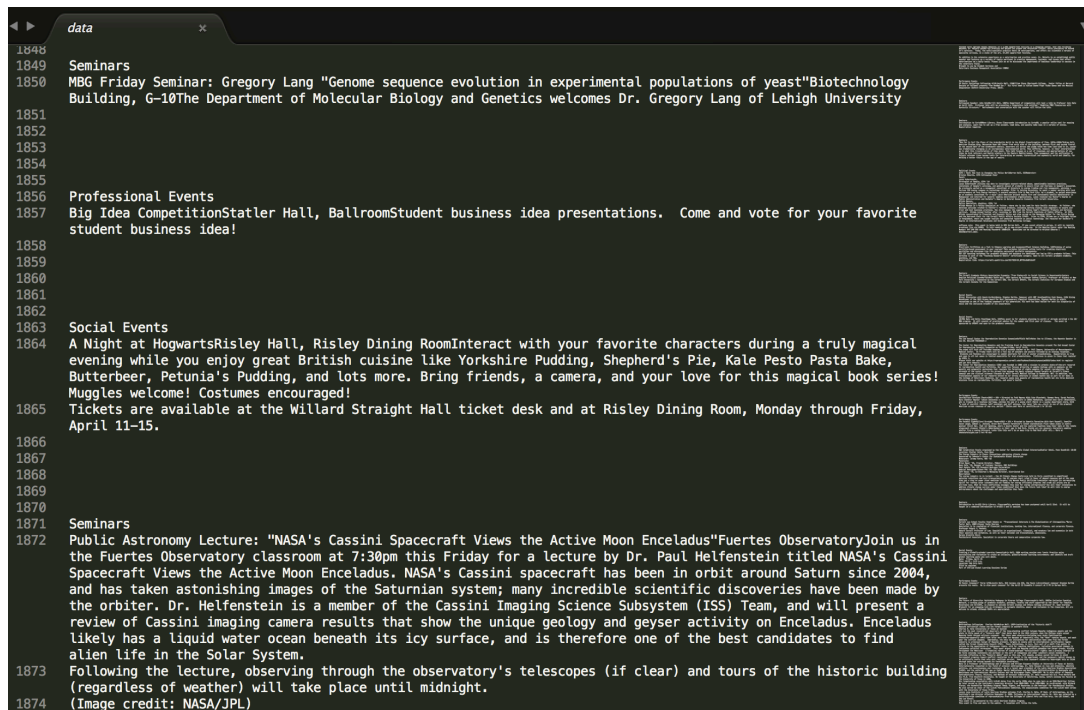


Figure 7.3 Data Ready for Processing

Currently, we have the category of the event as our label. Next we need to extract keywords from the title and description of the events as our features. To accomplish that, we use `topia.termextract` (<https://pypi.python.org/pypi/topia.termextract/>), which can “determines important terms within a given piece of content”. It filters out the useless words like “a”, “is” ... After extracting all the 202 events and filter out some unnecessary word like `\x02`, we found about 500 keywords in total, then we use the count of these keywords as our features. E.g. event 1 has 3 “seminar”, 0 “art”, 2 “talks”, ...

With features and label ready, we are ready to apply machine learning algorithm to it. We use scikit-learn python library to accomplish that. We tried decision tree algorithm, logistic regression algorithm, but the trained model end-up-with an accuracy of about 50%, and tends to misclassify many events to “Seminar Events”. We think there are two reasons for this result. First, there is not enough data for train a good model, especially there is too little data for “Political Event”, “Social Event”, “Political Event” and “Athletic Event”. It is because we don’t have enough user to generate the data. Also because there are much more events that can be classified into “Seminar Event” and “Performance Event” happening around the campus than other four categories.

Maybe we need to figure out a better way to specify categories to avoid this data skew. The second reason is that there could be many noise in the data. Since we generate the data based on our guess of how users would specify the category, it could be different from how users are really going to do. Both of these problem could be solved if we had enough data, and specify the categories more wisely to avoid data skew.

Joystick not Always Responsive

We use the joystick when navigating up-and-down in the event page. However, we found that the joystick is not responsive sometimes. When we are keep holding the joystick to one side, raspberry pi can't keep detecting low voltage on the corresponding side. This problem occasionally occurs. We try to fix the problem by checking the electric connection but end-up-with no help. This is a problem we still wouldn't be able to fix yet.

Conclusion

Within two semesters, we have successfully created a cross-platform system, consisting of a website, an iOS app and an arcade machine, aiming to provide students better ways to lead their lives. With our design, it would be really convenient for people to search for events they are interested with and would love to take part in. Both the website and iOS app are designed and implemented with concerning about the UI and UX, so they look really good and are pretty convenient to use. Furthermore, the arcade machine would also satisfy the curiosity of people, while searching for events they could also enjoy playing the airplane game with joystick.

We are quite sure that it still needs a long time for us to refine this project if we want it prefect, since we have trouble accurately classifying the categories of crawled events, and making joystick response quickly. And the functionalities and performance of the whole system also needs optimization.

Appendix