MBUSuite:
Registration & Scheduling System for Merit Badge University

A Major Qualifying Project
Submitted to the Faculty of
Worcester Polytechnic Institute
in Partial Fulfillment of the requirements for the
Degree in Bachelor of Science
in
Computer Science
By

Gregory T. Giola
Sean E. McCrone
G. David Modica

Date: 4/30/2015
Project Advisor:
Professor Michael Ciaraldi

Abstract

MBUSuite is a web application that will manage the registration and scheduling of Merit Badge Universities. The MBU hosted at WPI is run Omicron Iota chapter of Alpha Phi Omega. The software was written in Python using the Django framework, as well as standard HTML 5/CSS. It uses a heuristic algorithm to build a dynamic schedule for the MBU, and is customizable to meet a variety of event formats.

Table of Contents

Table of Code Examples

Table of Figures

Introduction

This project was designed to create a software suite to handle the registration and scheduling of the Merit Badge University held on the WPI campus every spring by the Omicron Iota chapter of Alpha Phi Omega. The brothers of APO have for several years handled the registration and scheduling by hand, and were looking to automate it online. One of us (Modica) has been a brother of Alpha Phi Omega since the spring of 2012, and the request for this software was given to him by officers in the spring of 2014. From this request, the project was born. This software would make it much easier for the brothers, as well as streamlining the process for Boy Scouts registering for the project. While certain things, like paying for the event, are required to be handled through WPI, the brothers handle everything else on their own.

The plan is for Boy Scouts to register for the Merit Badge University on a custom built website, where they list which Merit Badges they would like to earn. Once the registration period has closed, an administrator would start a scheduling program. This program would take the input from all of the registered Scouts and attempt to build a schedule that puts as many Scouts in their top choices as possible. Scouts would then be able to use the website to view their schedules, look at course material posted by instructors, and even figure out which rooms they need to be in for their Merit Badges.

The website would also be useful for instructors. One of the most important tasks for Merit Badge instructors is that they keep track of which requirements the Scouts have completed. Each instructor has access to a grid sheet for the Merit Badge they are teaching, and can update it in real time to make sure that they know which Scouts have completed which requirements. They can also post class notes and material for the Scouts to review, or even email the class about homework or reminders for later classes.

The ultimate goal of this project is to develop and ultimately distribute this software to other chapters of Alpha Phi Omega that host Merit Badge Universities, as well as to the Boy Scouts of America. This software would be highly beneficial towards other Merit Badge Universities that are held around the nation, and we hope that it will become a standard tool for the Boy Scouts at a national level.

In the background section of this report, we will explain what services the Boy Scouts of America provide to young men, how a Merit Badge University is run, and what the Alpha Phi Omega National Service Fraternity, particularly the Omicron Iota chapter, does to assist the BSA in regards to the Merit Badge University held on the WPI campus. In the following sections, we explain the various technologies that we used while working on this project, including Django, HTML5/Javascript/Bootstrap, and Python. We will also discuss the layout of our web application and database, and close with an overview of how we set up our server.

Background

Boy Scouts of America & Merit Badge University

The Boy Scouts of America (also known as the BSA) were founded on February 8th, 1910 [1], a few years before the start of World War I. They were founded in the spirit of training boys to be upstanding citizens and learn wilderness survival skills. Over the last 105 years, they have been fulfilling this purpose, and have impacted the lives of millions of young men and their families. The BSA is divided into many regions, each led by a council of adult volunteers that runs the Scouting program.

One of the methods that the BSA uses to keep a record of what Scouts have learned is Merit Badges. There are currently 136 Merit Badges listed on the official Boy Scouts of America website for Scouts to earn [2], encompassing a huge variety of topics, from Astronomy to Welding. Each badge also has a list of requirements that must be met in order to actually earn the badge. As technology has advanced, Merit Badges have been added to encompass new skills that Scouts will need to master in the twenty first century, including Programming and Digital Technology. In order to attain the rank of Eagle, the highest rank in Scouting, Scouts must earn at least 21 Merit Badges [3].

There are a variety of ways that Scouts can learn the skills required for a Merit Badge. The most common is by working independently or with a Scoutmaster to meet all of the requirements for the Merit Badge. Another is to find a Merit Badge Counselor, a Scouting volunteer that specializes in teaching a particular Merit Badge. However, for regions with a large quantity of Scouts all seeking similar Merit Badges, or for Merit Badges that require specialized knowledge, a local Scouting Council can hold a Merit Badge College or Merit Badge University. The only difference between the two is the scale; Merit Badge Colleges tend to be smaller than Merit Badge Universities. Both allow dozens or even hundreds of Scouts to come together in one place to learn together from many Merit Badge Counselors.

Many of these MBUs are held on the campus of local colleges and universities, as they can provide the space needed for dozens of classes simultaneously, volunteers to staff them, and the technology and passion to make the event happen. Volunteer Merit Badge Counselors are often college students majoring in fields related to the Merit Badges they teach, but other counselors are parent volunteers employed in those jobs.

Alpha Phi Omega & the Omicron Iota Chapter

The Alpha Phi Omega National Service fraternity was founded in 1925 at Lafayette College, Pennsylvania. The inspiration behind the founding was to resolve the issues of the world in a manner other than war, as the founder, Frank Reed Horton, served in the Navy during World War I, and vowed to find a better way. His plan was to inspire college students to provide community service across the world, and grow into the leaders of the future. In the 90 years since then, the Fraternity has grown from a single chapter to over 360 chapters in America, and even expanded to the Philippines, Australia, and Puerto Rico. The Fraternity voted to allow women to become members in 1976, and for a time, considered losing the title of Fraternity. However, a nearly unanimous vote from the women joining Alpha Phi Omega revealed that they wished to be called brothers as well, and the Fraternity remained a fraternity.

The Fraternity promotes three major aspects that it considers important in its members: Leadership, Friendship, and Service. Brothers are expected to become leaders in their chapters or other organizations, be friendly towards others, and provide service for less privileged members

of the community. Services provided by brothers can be one of four types: to the college, community, nation, or the Fraternity itself. The precise requirements to maintain active in a chapter are determined on the chapter level, rather than the national level. However, all chapters are expected to participate in activities such as National Youth Service Day and Spring Service Week.

The Omicron Iota chapter was founded on the WPI campus in 1964, and has upheld the main tenets of the National Fraternity for the last 50 years. Every brother is required to perform a minimum of 20 hours of community service each semester, and many are elected to positions within the chapter as officers or chairs. Three of their four annual events are assisting with Freshman Move-In, Service Auction, and U.M.O.C (U.G.L.Y. Man On Campus [Understanding, Generous, Loyal, Youthful]). The fourth is Merit Badge University, explained below. These services are primarily directed at helping the campus community, rather than the Worcester community at large, because it is required for all brothers to attend and serve at these events.

However, other services, known more commonly as "weekly services" are rendered on a weekly basis, and do not require as many participants. Brothers assist soup kitchens, harvest crops at the Worcester branch of Community Harvest Project, volunteer at Habitat for Humanity's ReStore, and maintain trails for the Audubon Society, Worcester Greenways, and the Southside Community Land Trust. The brothers of Omicron Iota have had a hugely positive impact on the region, and provide thousands of hours of service every semester.

Merit Badge University at WPI

The Merit Badge University held on the Worcester Polytechnic Institute campus is hosted by the Omicron Iota Chapter of Alpha Phi Omega, and is one of four annual service events hosted by the chapter. First held in the spring of 2011, it has rapidly grown from a pet project created by a single brother to a massive service event that attracts 250-300 Boy Scouts annually and requires participation from every brother in the chapter in order to run.

In 2013, it was decided that Merit Badge University would become the 4th annual service event held by Omicron Iota, to be held in C or D Term every year. As an annual service, brothers became required to participate, and there are many roles that need to be filled. The most common is as a Merit Badge Counselor. Brothers can choose any Merit Badge (within reason) to teach, and individually or in small teams develop a lesson plan that they then deliver to the Scouts. Many of the badges taught are technical or require special knowledge or materials, such as Robotics, Nuclear Science, and Programming. However, brothers are allowed to teach anything they are passionate about, and badges such as Astronomy, Theater, and Cooking have been offered, in addition to Eagle-required Merit Badges such as Citizenship in the Nation, Citizenship in the World, and First Aid[4]. Other brothers act as classroom assistants, helping the Counselors in controlling the class and grading homework assignments.

Equally important to the Counselors are the brothers serving on the "Blue Card Team". These brothers are responsible for making sure that each Scout receives a Blue Card, which is a ubiquitous reporting method for receiving Merit Badges. On the Blue Card, the brothers record which badge the Scout is attempting to earn, as well as which requirements the Scout has completed. Typically filled out by the Merit Badge Counselors, with only 2 hours to fill out the cards and return them, a special team of brothers is trained to fill out all of the information required by the Counselor, who then can sign off on the Blue Card if the Scout has finished all of the requirements for the badge.

The most important role, however, are the brothers organizing the event. Usually former Scouts themselves, this group of brothers is responsible for working on all of the logistics of the

event, making sure brothers fill out their paperwork, and generating a schedule spanning two Saturdays over the course of two weeks. However, thus far, all of this work has been done manually, requiring dozens of hours of work from these brothers in order to make sure everything happens as smoothly as possible. In order to manage all of this, the brothers who organized the Merit Badge University in 2014 requested that this software be built to assist them and put all of this information in one place online.

Our Work

Language – Django & Python

During the initial planning stages, the plan was developed to allow any and all of the three members to work on any and all aspects of the project. It was for this reason that we chose Python. Python was the language we were most in agreement with for the scheduling algorithm. In production however, the task for the creation of the algorithm fell to a single member (Modica).

Django acts as the middleware between all of the languages: Python, MySQL, and HTML5 [5]. This means that Django manages all of the interactions between our webcode and database, as well as managing a local copy of a server. Because of the way Django interacts with other servers, it does require a dedicated machine to be host it, as well as to ensure continuous accessibility to online users.

On setup, Django defines its folder structure with the 'manage.py' python file, the 'mySite' folder, the 'apps' folder, and the 'templates' folder, shown in Appendix III [6]. The 'manage.py' python file is the file that directs command line commands for the project. The 'mySite' folder is responsible for settings and back-end structure of the website, containing the 'settings.py,' 'urls.py,' and 'views.py' python files. 'urls.py' and 'views.py' specify the pages that will be displayed when a user navigates to a designated web URL. 'settings.py' defines locations of files, imports, and traits like database information. Database interaction is handled through the 'apps' folder. 'apps' defines models, which are the object traits the user designates. Each model has a database table created to properly store any object of that type. The 'templates' folder is where the HTML pages are stored. It is within these templates that Django-specific code is inserted to allow the HTML pages to interact with code from the other languages.

Inside 'views.py' and 'urls.py' is the code that controls navigation. In 'urls.py,' Django knows how to handle the URL request ending in login.html by directing it to the appropriate definition of 'login' in 'views.py'.

```
url(r'login.html', 'mbusuite.views.login'),
```

Code Example 1

In 'views.py,' login is defined as follows:

```
def login(request):
    user_list = Users.objects.order_by('user_name')[:5]
    return render_to_response('login.html', RequestContext(request, {
        'user_list': user_list,
    }))
```

Code Example 2

This code from 'views.py' specifies that a request for the specified webpage url ending in /login.html will have access to the variable named user_list, which contains up to 5 objects in the users table of the database ordered by their user_name. This url will use the template named login.html, which will know what to do with the variable it is being sent.

Models in the 'Apps' folder are constructed in this format:

```
class Users(models.Model):
    user_name = models.CharField(max_length=200)
    user_pass = models.CharField(max_length=200)
    def __str__(self):
        return self.user_name
```

Code Example 3

In this simple example, the 'class Users' defines this model as 'Users'. The class function defined by 'def __str__(self)' designates what the model will return when prompted by Django commands in a command terminal. In this example, that return is the user_name field. This model will create a table in the database with the 'makemigrations' command. Using the command 'python manage.py makemigrations storage' will prompt Django to create a table in the selected database adhering to the model's structure. In this example, it will have a user_name and a user_pass field. Django also gives the table an 'id' field for indexing purposes. After adding 4 example users, the table in the database looks like this:

id	user_name	user_pass
1	testname	testpass
2	Greg	GregPass
3	Dave	DavePass
4	Sean	SeanPass

Figure 1

In a template that uses this model, we can call information from the database using code of this format:

```
{% for users in user_list %}
    <li>{{ users.user_name }}</li>
{% endfor %}
```

Code Example 4

Django code in templates is marked by the {% … %} and {{…}} delimiters. This code specifies that for every user object in the variable user_list, (being sent by 'views.py') display it in a bulleted list. The user names display like this:

- Dave
- Greg
- Sean
- testname

Figure 2

Language – HTML 5

Django works with HTML5 by allowing the designer to write Django code into the .html template files. This can be inserted anywhere in the file, provided it is properly enclosed within the Django tags. The Django code is used to interact with the database models. Model objects are

accessed by a variable name provided in views.py. Individual traits are also accessible through the same variable. Django also provides control functionality like for loops and if statements for more precise control of given variables.

The following code is an expanded version of an existing example, now showing additional control structures. It also shows Django code working in conjunction with (and on the same line as) traditional HTML.

```
{% if user_list %}
    <ul>
    {% for users in user_list %}
        <li>{{ users.user_name }} as well as regular text </li>
    {% endfor %}
    </ul>
{% else %}
    <p>No users are available.</p>
{% endif %}
```

Code Example 5

This code produces the following output:

- Dave as well as regular text
- Greg as well as regular text
- Sean as well as regular text
- testname as well as regular text

Figure 3

This output shows how both plain text and HTML bulleted lists work in conjunction with the Django for loop.

Scheduling Algorithm

The scheduling algorithm was easily the most complex piece of programming that we created during this project. It uses a recursive heuristic method to iterate through all registered scouts, and build a schedule that would satisfy the maximum number of Scouts. To begin this process, an administrator would have to actively disable the registration system, in order to prevent changes to the database from occurring, which could have potentially disastrous effects. This would be accomplished either by defining a date and time in a configuration file for the server to close registration automatically, or through the use of a button that would cause people attempting to access the login page to be redirected to an explanation page. As such, we do need to avoid any issues with concurrency, and we felt that once the registration period closed, it would also be appropriate to shut down any attempts at changing registrant data.

With the concurrency issue out of the way, administrators can then press a button to begin generating the schedule. Because this is recursively heuristic, all permutations of possible schedules are "built" in order to find the best possible schedule. This means that the maximum amount of time required to run this algorithm, also known as Big-O, is $O(x^n)$, where x is the number of top choices a Scout can make, and n is the number of Scouts. This is a brute force

method, but it will be effective in the end. Unfortunately, the algorithm was never written in an actual language; the most we were able to complete was pseudocode.

However, even with only pseudocode, the design of the scheduling algorithm can be understood. There are five major functions: main(), create_schedule_plan(), recurse(), add_class_to_block(), and add_scout_to_class(). main() simply begins the program, and serves as an abstraction that protects all of the data-manipulating helper functions. recurse() is also very simple, as it only pops a scout off of a list of scouts, and then calls create_schedule_plan(), which is also the only function to call recurse(), aside from the initial call in main(). add_scout_to_class() is the last simple function, and updates class numbers and values in a Scout's profile. add_class_to_block() is more complex, and requires many checks to see if a room is available in a block, if the class requires more than one block, and other such constraints.

The last function, create_schedule_plan(), is the main body of the algorithm. It takes a scout and checks each of their requested classes, in order of preference. If a class is not full, the scout is added to it, and the function recurse() is called. If the class is full, the function skips to the next class, or in some cases, can generate a new section in a different block. However, if the class is unscheduled, it calls add_class_to_block, and then follows the procedure for a non-full class. What makes this function interesting is the heuristic. When a Scout is added to a class, the Scout's preference for that class is used to generate a heuristic value, which is then added to a score. The higher the Scout's preference for a given class, the higher the heuristic value will be. This score is used to build an entire schedule, and the schedule with the maximum score is considered "the best". However, the score value is complex, as it must be persistent through many levels of recursion and looping and only get modified when the score is greater than any previous iteration. This makes preserving the original value at each level difficult, and also imperative. We believe that we have accomplished this, but as with the rest of our work, it would need to be fully implemented before we can properly test it. For a more complete understanding of the algorithm, complete with notes, please see Appendix I.

Website Design

When creating the website, we wanted a design that matched the aesthetics of both Alpha Phi Omega and the Boy Scouts of America. We went with a royal blue text on a gold background, as these are the colors of APO. As the background moves closer to the edges of the screen, it fades slightly. The website was designed with mobile use in mind. This was very important for us to follow with the increase of mobile devices. We wanted Scouts to have the ability to access the site from any device with ease. To accomplish this, we used the frameworks Bootstrap and JQuery.

Bootstrap is a free and open-source front-end framework. It contains HTML and CSS-based design templates for web development. The benefit of using Bootstrap is that it efficiently scales websites with a single code base from mobile devices to desktops. This was a perfect fit for designing our website. JQuery is a free and open-source JavaScript library. This lightweight library simplifies document traversal and animations. JQuery allowed us to add the final aesthetical design decisions while creating the website.

The layout of the website was designed to allow each user to have easy access to the pages they need. When accessing the site for the first time, users are brought to the login page where they can login or register for an account. They can also view the contact page from below which provides information to contact site administrators, as seen in Figure 4.

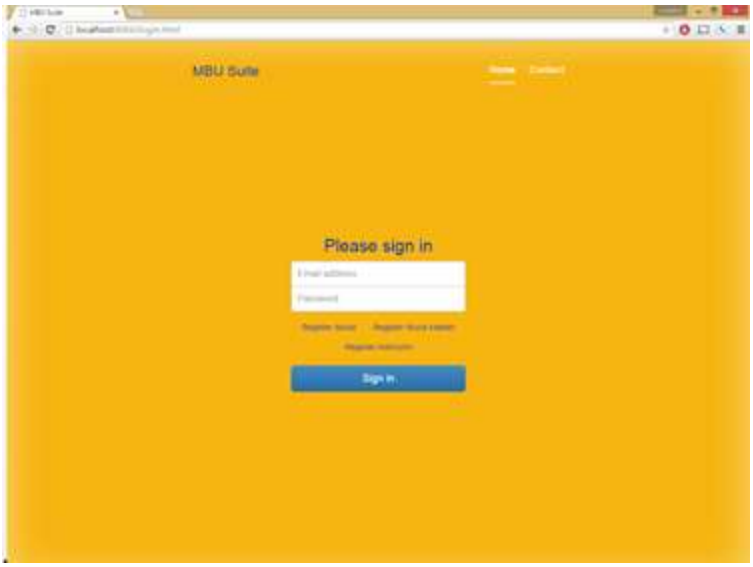


Figure 4

Registering for an account on our site is very simple and similar to most other websites. Below is the view of the registration page for a Scout. This page is similar for Scoutmasters and Instructors, both of which have to register and be approved in order to access our site. While registering an account, users will be asked to verify their identity with a site administrator through an access code that they would receive after paying for MBU admission, which must be handled separately from our site.

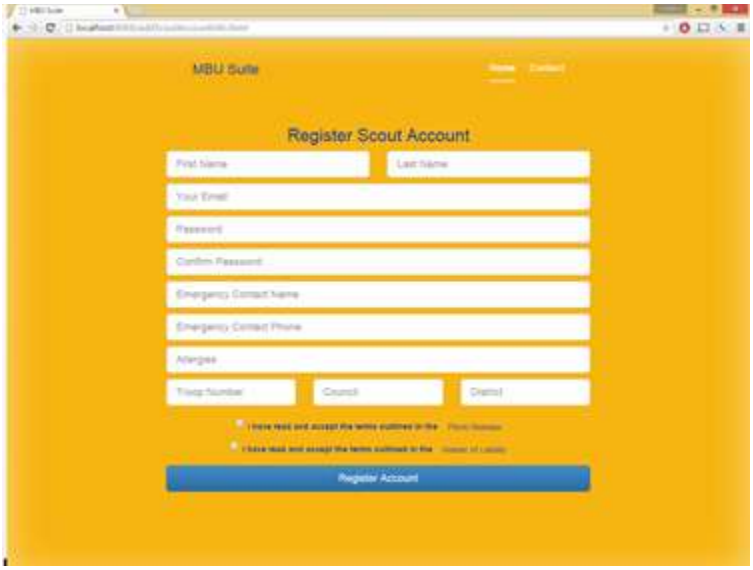


Figure 5

Once a user has a verified account, they will be brought to their home page. This home page will change depending on whether the user is a Scout, Scoutmaster, Instructor, or Administrator. Scouts will see all the courses they are currently registered for and a calendar showing when and where each of the courses will meet. A Scout will be able to click on each course they are registered for and see details about it. The details will also include the requirements the Scout has completed and the ones they still need to complete. A Scoutmaster's

home page will be a little different. They will have a list of all Scouts in their troop that are registered for the MBU. Below the list of scouts will be a calendar showing when all of their Scouts are in a course. The Scoutmaster will be able to click on an individual Scout and see a more detailed list of courses that individual Scout is registered for, and a calendar for when that Scout is in a course. Additionally, the Scoutmaster will be able to see what requirements the individual Scout has completed and which requirements still must be completed.

The Instructor's home page will show a list of courses they are teaching and a calendar for when and where those courses meet. Instructors will have the ability to click on a course they are teaching, and to mark off what requirements a Scout in their course has completed. The Administrator's home page will show any accounts that need to be verified, a link to add users manually, and a link to create a new course.

As a user traverses through the website, the navigation bar will stay very similar, only changing links that are pertinent to the current page. However, the home and user links will always stay constant. This allows users easy access to go pages that we expect will be most important to them. For example, on a Scout's home page, the "Course Registration" link would be the most important thing (seen in Figure 6), whereas on an Administrator's home page, the "Add a course" link would be more important. Also from the navigation bar, a user will have easy access to logout and access account settings.

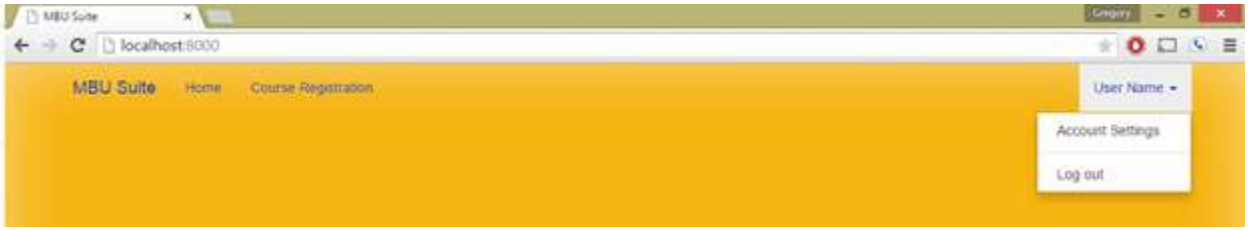


Figure 6

While designing the website, we made sure to keep security in mind. It was very important to us to make sure that all user information was stored securely and only accessible by administrators. All forms were created making sure that they were not susceptible to SQL injection or XSS attacks. In order to ensure this, we prevented the ability to enter SQL into forms, as well as encoding variable output in our pages to prevent XSS attacks. To protect users against any information leaks, we made sure that only administrators had the ability to view user information. We also made sure that all sensitive information was encrypted on our server.

Database Design

A database was needed for the backend of our system to keep record of all users and courses. This database was created using MySQL version 5.6. When creating the database, we wanted to make sure that only information we used was stored. This was to protect all users in case of data being stolen. To further protect users, we made sure that all data was encrypted when stored. We created an entity relationship diagram (ERD) to show the flow of information stored within the database, as seen in Appendix II.

Scouts are related to classes they are in, which are taught by Instructors, while an Administrator can create a Merit Badge, which has Requirements and is taught in a class. The details that are saved for Scouts, Instructors, and Administrators are fairly standard: name, email, username, password, and specific details for Scouts and Instructors. A Scout is part of a ClassList. This ClassList keeps track of what section of a class is being taught and which Scouts

are part of that section. Each ClassList relates to a specific class. These classes (ClassDetails table) keep track of how many seats are offered in the course, which Instructor is teaching the course, and what Merit Badge is being taught. The Merit Badges that are being taught are stored in the database, along with which Administrator created it and what the Merit Badge is. Each Merit Badge has its own Requirements table, each of which must be completed by each Scout in the ClassList. These requirements are stored by linking a requirement to both a Merit Badge and Scout through the use of foreign keys.

Host Server Architecture

We were given a Dell Optiplex 760 by the WPI CS department to host the server on. This machine was equipped with an Intel Core 2 Duo processor, 4GB of RAM, and 160GB of hard disk storage. We then installed Ubuntu desktop version 14.04.2 onto the Dell computer. We chose Ubuntu desktop because it is a free platform that can easily be installed and has a graphical interface. Version 14.04.2 was the most current release at time of installation. After installing Ubuntu, we installed the proper versions of Python, MySQL, Django, and Apache.

Apache 2.4.7 is the HTTP server that we used to configure the MBUSuite. We chose this because it allowed easy configuration with Ubuntu and the ability to work with Django. To access the site, we created a configuration file as shown on the following page.

We modeled this file after examples from the Django website and the default Apache site. This configuration file gave apache the proper access to the individual Django files. As one can see, every directory must be accounted for in order for the site to load properly. This configuration file is also running a WSGI Daemon. This was chosen because Django requires it to be running while using any version of Apache.

```
<VirtualHost *:80>
     ServerName mbusuite.duckdns.org
     WSGIScriptAlias / /var/www/mbusuite/mbusuite/wsgi.py
     WSGIProcessGroup mbusuite.duckdns.org
     WSGIDaemonProcess mbusuite.duckdns.org python-
path=/var/www:/usr/local/lib/python2.7/site-packages
     ServerAdmin mbu@wpi.edu
     ErrorLog ${APACHE_LOG_DIR}/error.log
     CustomLog ${APACHE_LOG_DIR}/access.log combined
     <Directory /var/www/mbusuite/static>
          Require all granted
     </Directory>
     <Directory /var/www/mbusuite/core>
          Require all granted
     </Directory>
     <Directory /var/www/mbusuite/templates>
          Require all granted
     </Directory>
     <Directory /var/www/mbusuite/>
          <Files wsgi.py>
               Require all granted
          </Files>
     </Directory>
</VirtualHost>
```

Code Example 6

To configure Django to run properly the wsgi.py file, seen in example 7, must be edited. This file controls how the server starts running.

```
import os
import sys
import threading
os.environ["DJANGO_SETTINGS_MODULE"] = "mbusuite.settings"
from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()
```

Code Example 7

Most of this file is set up when the Django project is started however the first four lines are added when running the server in Apache.

When testing, we used a free domain name service called DuckDNS. DuckDNS provided us with five free domain names which gave us the ability to test the site prior to deploying it. We went with a free domain name service because when the MBUSuite will be deployed on WPI campus. WPI does not require a server to have its own individual domain name.

Results

This project established a solid base for future projects, but unfortunately, we were not able to finish the application. We managed to build many of the web pages that make up the application, but there are still several that we need to build. These are the more complex pages that include complicated Javascript commands for Administrators of the system. Additionally, we have completely built the database and Django models that are needed to ensure the database can be properly accessed by Javascript and Python scripts. We have confirmed that we can properly register Scouts for the event, which satisfies one half of the requirements that we were given to start.

In addition, we have written pseudocode for the scheduling algorithm. While it has not been written as an actual program, and thus has never been properly tested, we are reasonably certain that it will work. There are further refinements to the algorithm that will need to be done, particularly in the optimizing classroom placement and pruning. The current form of the algorithm is too brute-force, and does not rely enough on the heuristics to know when to abandon a particular branch. A method for optimization could be to use a breadth-first heuristic, rather than the current depth-first one, which would enable us to prune more options much more easily, and drastically reduce the run-time for the algorithm.

We would also like to be able to deploy our architecture so that it can be accessed through the web, rather than by using local copies. During the development phase, we encountered an issue between Django and Apache that would prevent access to the website from non-local addresses, a basic HTML 403 “Permission Denied” error. Due to the complex interactions between Django and Apache, we were unable to resolve this issue. However, an alternative is to host the application directly on the WPI servers, through the use of the Omicron Iota group account hosted by WPI. We were unable to obtain access to this account during the project, but one of us (Modica) will be able to work with the chapter to allow access to project teams in the future.

Conclusion

This was an ambitious project that required the team to learn new technologies, and the way those technologies interacted with one another. These interactions were the most difficult part of the project. Every technology was built to be expansive and capable of working with a variety of software. The trade-off for this is lack of documentation for every single permutation. This is exactly the issue we found in trying to run Django with Apache on our Ubuntu desktop. Official documentation did not exist, and the unofficial documentation that we followed provided no solutions to our problem. Asking questions on forums like StackExchange were also unsuccessful. As such, we failed to finish the project, but we did manage to build a solid base to continue working from in the future. Towards this end, we have attached to this report our entire repository and server configuration files, as well as all of our documentation and notes on the project, and hope they will be sufficient for future teams to pick up where we left the project.

Our website will continue to be developed, either by member(s) of the current team as an independent project or by future project teams. It will be hosted on a private Github repository until a full version is developed and can be deployed by the Omicron Iota chapter. After a successful trial run is complete, the chapter will be able to distribute the software to other chapters of Alpha Phi Omega or to the Boy Scouts of America in order to plan other Merit Badge Universities.

Appendices

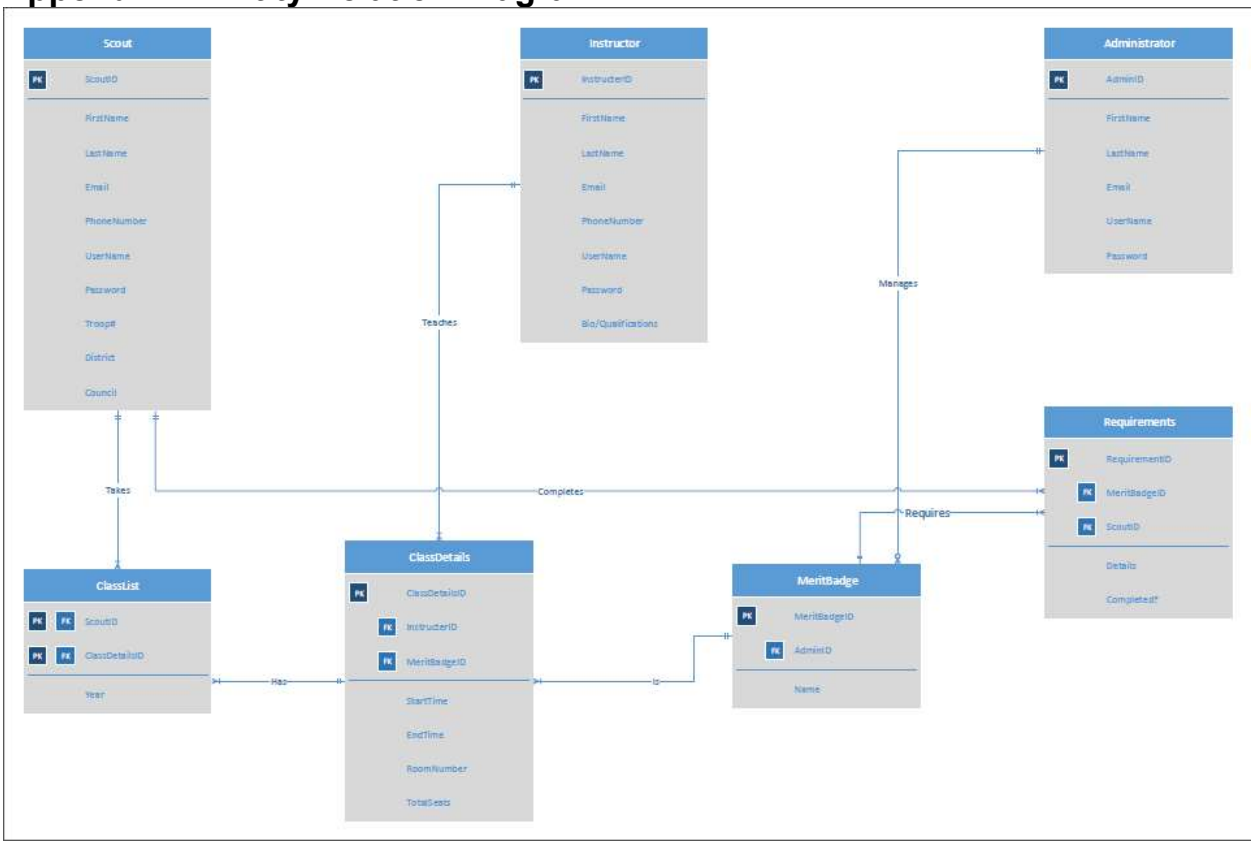
Appendix I. Scheduling Algorithm

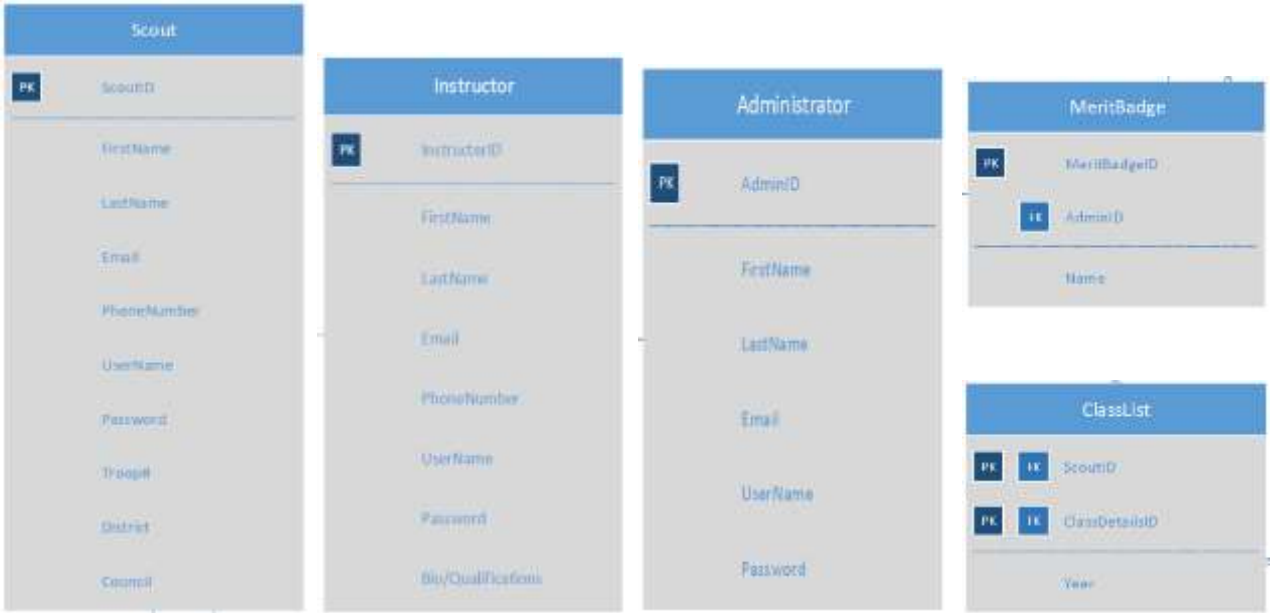
```
Algorithm Variables
       Num_Students
       Num_Blocks
       Num_Rooms
       num_room_avail_array       //length = Num_Blocks, each element init to Num_Rooms
       available_double_blocks
       scout_list
       class_list
       current_schedule
       best_schedule
       current_score
       max_score
              //init to 0

Class Variables
       current_scouts
       max_scouts
       instructor
       is_double_block
       block_num                  //Which block the course is taught in, init to -1

Scout Variables
       pref_class_list
       act_class_list
       num_classes
       reg_time


Algorithm
Main(){
       sort scout_list on reg_time
       //earliest time first
       recurse(scout_list, 0)
       return best_schedule
}

create_schedule_plan(scout, scout_list, last_score){
       counter = 0
       for each class in pref_class_list{
              if scout.num_classes == num_Blocks                    //Makes sure a
scout can only take as many classes as there are blocks
                     break;              // breaks out if this is hit, because the
scout is full up
              if class.current_scouts != max_scouts

                     if class.block_num != -1{                                    //Dont
need to add class to schedule, can use to start pruning
                            add_Scout_to_Class(scout, class, scout_list)
                            current_score = last_score + (100 - 7 * counter)
                            counter++
                            recurse(scout_list, current_score)
                     }
                     else{
                            for (i = 0, i < Num_Blocks, ++)
                                   try{
                                          add_Class_to_Block(i, class)
       //Adds class to earliest available block
                                          add_Scout_to_Class(scout, class, scout_list)
```

```
                                        current_score = last_score +(100 - 7 *
counter)            //randomly chosen values for scores
                                        counter++
                                        recurse(scout_list, current_score)
                                  } catch error{
                                        if i == Num_Blocks - 1
      //Should never occur, means class cannot be scheduled
                                              error code
                                        else
                                              error code and continue
                                  }
                  }
                  if(current_score > max_score){
                        best_schedule = current_schedule
                        max_score = current_score
                  }
                  scout_list.push(scout)
      //Re-add the scout to the list so the list doesn't get emptied.
      }
}

recurse(scout_list, score){
      if (scout_list.length > 0)
            first_scout = pop(scout_list)
            create_schedule_plan(first_scout, scout_list, score)
}

add_Class_to_Block(i, class){
                  //Always chooses the earliest available block
      if num_room_avail_array[i] == 0{
            throw error
      }else if (class.is_double_block && i == Num_Blocks - 1){
            throw error
      }else{
            if(class.is_double_block && i not in available_double_blocks){
                  throw error
            }
            else if (class.is_double_block){
                  if class.instructor == curr_class.instructor
            //make sure no instructor conflict
                        if i == curr_class.block_num || i+1 == curr_class.block_num
                              throw error
                  else
                        current_schedule += class
                        //not as simple as "+=", but equivalent
                        num_room_avail_array[i]--
                        num_room_avail_array[i+1]--
            }
            current_classes = get_classes_from_curr_schedule()
            for each curr_class in current_classes{
                  if class.instructor == curr_class.instructor && i ==
curr_class.block_num       //make sure no instructor conflict
                        throw error
                  else
                        current_schedule += class
                        num_room_avail_array[i]--
            }
      }
}

add_Scout_to_Class(scout, class, scout_list){
      scout.act_class_list += class
      scout.num_classes++
```

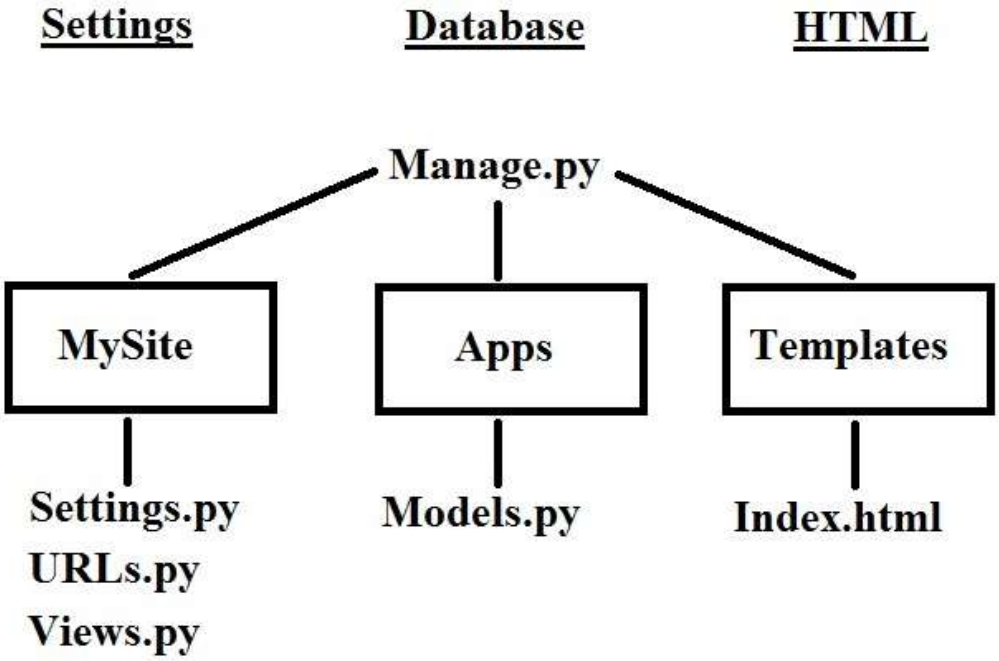
```
        class.current_scouts++
        if class.is_double_block
              scout.num_classes++                    //Adds a second block to make sure
scout doesnt take too many classes
}
```

Appendix II. Entity Relation Diagram





Appendix III. Default Django Architecture



Works Cited.

[1] http://www.scouting.org/About/FactSheets/Founders.aspx
[2] http://www.scouting.org/meritbadges.aspx
[3] http://www.scouting.org/About/FactSheets/EagleScouts.aspx
[4] http://www.scouting.org/scoutsource/BoyScouts/AdvancementandAwards/eagle.aspx
[5] https://www.djangoproject.com/
[6] https://docs.djangoproject.com/en/1.8/intro/tutorial01/