SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

Date: November 27, 2018

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Jack Ryan
Sam Burns
Simran Judge

ENTITLED

Alumni Attendance Reporting Application

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

_____
Thesis Advisor

_____
Department Chair

Alumni Attendance Reporting Application

by

Jack Ryan
Sam Burns
Simran Judge

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
November 27, 2018

Alumni Attendance Reporting Application

Jack Ryan
Sam Burns
Simran Judge

Department of Computer Engineering
Santa Clara University
November 27, 2018

ABSTRACT

Currently, the Santa Clara University Alumni office lacks a system that will help them take detailed attendance reports on nationwide alumni events. This data will help them to better plan future events and to reflect on past events to maximize alumni involvement. This report describes a proposed system to encourage and facilitate the self reporting of attendance by alumni at these events. The web application will function as both advertisement for future events while also encouraging users to check-in and record their own attendance. This system will improve the attendance data received by the Santa Clara University Alumni office by offering confirmed attendance data in the form of a computer generated report rather than relying on often inaccurate RSVP information.

Table of Contents

1 Introduction **1**

2 Requirements **2**
2.1 Functional Requirements . . . 2
2.2 Non-Functional Requirements . . . 2
2.3 Design Constraints . . . 3

3 Use Cases **4**
3.1 Use Case Diagram . . . 4
3.1.1 Alumni Office User Logs In . . . 4
3.1.2 Create Event . . . 5
3.1.3 Approve Event . . . 5
3.1.4 Confirm Attendance . . . 6
3.1.5 View Attendance Report . . . 6

4 Activity Diagrams **8**

5 Technologies Used **10**

6 Architectural Diagram **11**

7 Design Rationale **12**
7.1 User Interface . . . 12
7.2 Technologies Used . . . 12

8 Description of System Implemented **14**

9 Testing Procedure **16**
9.1 Unit Testing . . . 16
9.2 Alpha Testing . . . 16
9.3 Beta Testing . . . 16

10 Difficulties Encountered **17**
10.1 Misunderstood Requirements . . . 17
10.2 Corrupted Database . . . 17
10.3 Lack of Familiarity with Framework . . . 17

11 Suggested Changes **18**
11.1 Use of Dropdown Menus . . . 18
11.2 Event Categorization . . . 18
11.3 Increased Modularity . . . 18

12 Lessons Learned **19**
12.1 The Importance of Clarifying Requirements . . . 19
12.2 Unit Testing . . . 19
12.3 Synchronizing Git Commits to Avoid Collisions . . . 19

A Installation Guide **20**
A.1 Install the Correct Python 3 Version (Python 3.7) . . . 20
A.1.1 Mac OSX . . . 20
A.1.2 Linux . . . 20
A.1.3 Next Steps for All Operating Systems . . . 21
A.2 Install VirtualEnv and Create the Environment . . . 21
A.3 Installing the Dependencies . . . 21
A.4 Setup Database and Create Administration User . . . 22
A.5 Start the Application Server . . . 22

B User Manual **23**
B.1 For All Users . . . 23
B.1.1 Register Attendance . . . 23
B.1.2 Create Event . . . 23
B.2 For Alumni Office Use Only . . . 23
B.2.1 Login . . . 23
B.2.2 Approve Events . . . 23
B.2.3 Report Generation . . . 24
B.2.4 Manage Events . . . 24

List of Figures

3.1 Use Case Diagram of the Alumni Attendance Reporting System 4

4.1 Activity Diagram for Alumni User 8
4.2 Activity Diagram for Alumni Office User 9

6.1 High Level System Architectural Diagram 11

Chapter 1

Introduction

The Santa Clara University Alumni Office is in need of a system to track the participation and engagement of the alumni community. With a large number of events held every year, it is difficult for them to fully measure the benefit of these events without accurate and reasonable data. The system needs to provide a detailed report to the Alumni Office of the event containing a description of the event, where it was located, and how many alumni participated. Events must be administered by the alumni office, but may be created by any alumni user with approval from the Alumni Office manager. Once the event is created and is in progress, alumni may check-in to the event through the system which will allow the system to generate the report containing details about the event. The current SCU Alumni Engagement Recording system is simply based upon RSVPs, which is an inaccurate estimate of the number of participants. With no post event follow up and no guarantee that the the alumni who RSVP will actually attend the event, the data gathered from this solution is not useful to the Alumni Office.

Our solution to this problem is a lightweight web application with users of different classes designated for general use and administration. The administrator users will need to log in before they are able to access the interface for alumni events. This web application will offer two separate forms, one for official events created by the SCU Alumni Office, and another for unofficial events created by the alumni community. Unofficial events will need to be approved by the Alumni Office before they will appear to the alumni community. Users of both designations will be able to check-in and confirm their attendance at a specific event. Once the event has concluded, the application will generate a report containing details about the event, where it was located, and how many alumni participated. Using the data gathered through this system, the Alumni Office Manager will be able to review and analyze past events and improve future events. This system is an improvement upon previous solutions because it offers the Alumni Office access to confirmed, post-event data that is typically not collected by the current system.

Chapter 2

Requirements

2.1 Functional Requirements

Critical:

- Alumni Office users must be able to log in to the system.
- Users must be able to create an event.
- The Alumni Office must be able to approve alumni-suggested events.
- Users must be able to check-in to an event to record their attendance.
- The system must generate a detailed report containing the name of the event, where it was located, and how many alumni participated.

2.2 Non-Functional Requirements

Critical:

- The system will be able to scale to handle high traffic times such as events.
- The system will be easy to use to encourage participation.
- The system will be secured by verifying alumni status of users before allowing participation.

Recommended:

- The system will guarantee data persistence.

2.3 Design Constraints

- The system must be a Web Application.
- The system must function on the Santa Clara University Design Center computers.

Chapter 3

Use Cases

This section outlines five different ways in which a user may interact with our system. The system requires two types of users with different permissions. The use cases defined below produce a system which is easy to use and reliable.

3.1 Use Case Diagram

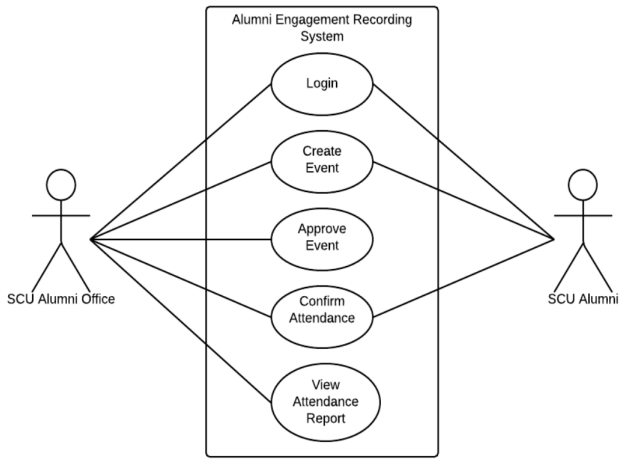

Figure 3.1: Use Case Diagram of the Alumni Attendance Reporting System

3.1.1 Alumni Office User Logs In

Goal: Allows administrator user access to Alumni Engagement Recording System website

Actors:

(1) SCU Alumni Office

Pre-Conditions:

(1) Application must be open on device

(2) Login page must be open

Steps:

(1) Enter login credentials

(2) Press Login button

Post-Conditions:

(1) Application provides access to user's account

Exceptions:

(1) User must enter valid login credentials

3.1.2 Create Event

Goal: Create SCU alumni event

Actors:

(1) SCU Alumni Office

(2) SCU Alumni

Pre-Conditions:

(1) Application must be open on device

(2) Create Events page must be open

Steps:

(1) Enter event information

(2) Press Submit Event button

Post-Conditions:

(1) Event is submitted for review

Exceptions:

(1) User must fill all required fields

3.1.3 Approve Event

Goal: Approve SCU alumni event

Actors:

(1) SCU Alumni Office

Pre-Conditions:

(1) Application must be open on device

(2) User must be logged in as Alumni Office employee

(3) Approve Events page must be open

Steps:

(1) Press ”Approve Event” button

Post-Conditions:

(1) Event shows up on events page

Exceptions:

(1) User presses ”Reject Event” button

3.1.4 Confirm Attendance

Goal: Submit attendance information

Actors:

(1) SCU Alumni Office

(2) SCU Alumni

Pre-Conditions:

(1) Application must be open on device

(2) Event Detail page must be open

Steps:

(1) Press ”Register Attendance” button

(2) Enter name, email, and graduation information

(2) Press ”Submit” button

Post-Conditions:

(1) Attendance is confirmed for user

Exceptions:

(1) User must fill all required fields

3.1.5 View Attendance Report

Goal: View attendance report for specific event

Actors:

(1) SCU Alumni Office

Pre-Conditions:

(1) Application must be open on device

(2) User must be logged in as Alumni Office employee

(3) Reports page must be open

Steps:

(1) Press ”View Attendance” button

Post-Conditions:

(1) Display detailed attendance report for specific event

Exceptions:

(1) None

Chapter 4

Activity Diagrams

The following activity diagrams show the work flow for the two different types of users in the system, the Alumni User and the Alumni Office User. The Alumni User represents the general user of the website, able to suggest events and record their attendance. The Alumni Office User represents the administrators in the Alumni Office. They are able to view attendance reports, approve alumni suggested events, and instantly post new events to the application.

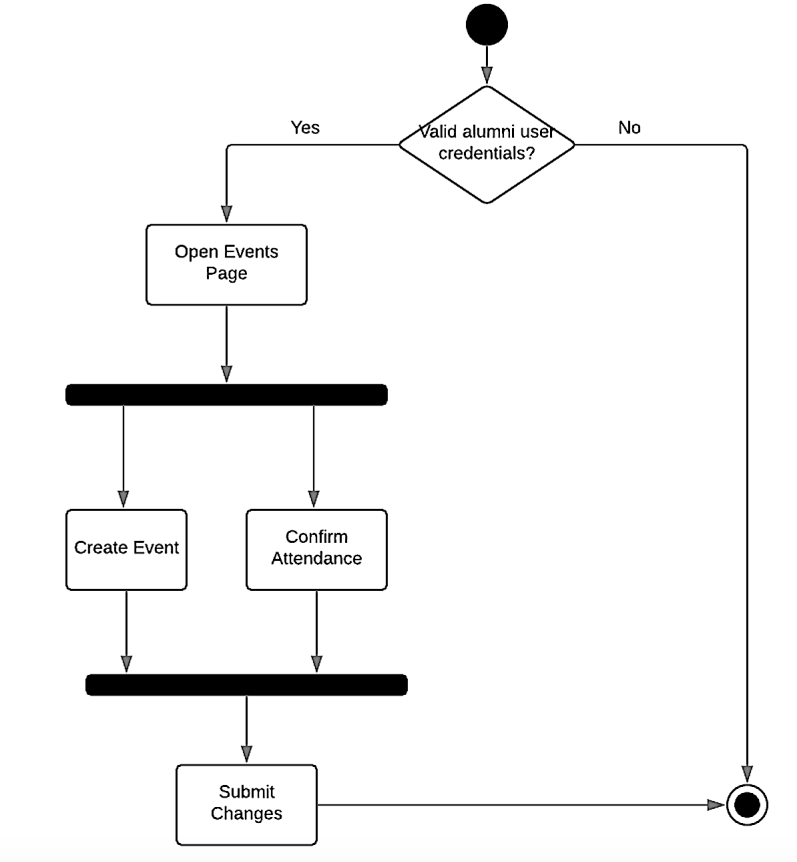

Figure 4.1: Activity Diagram for Alumni User

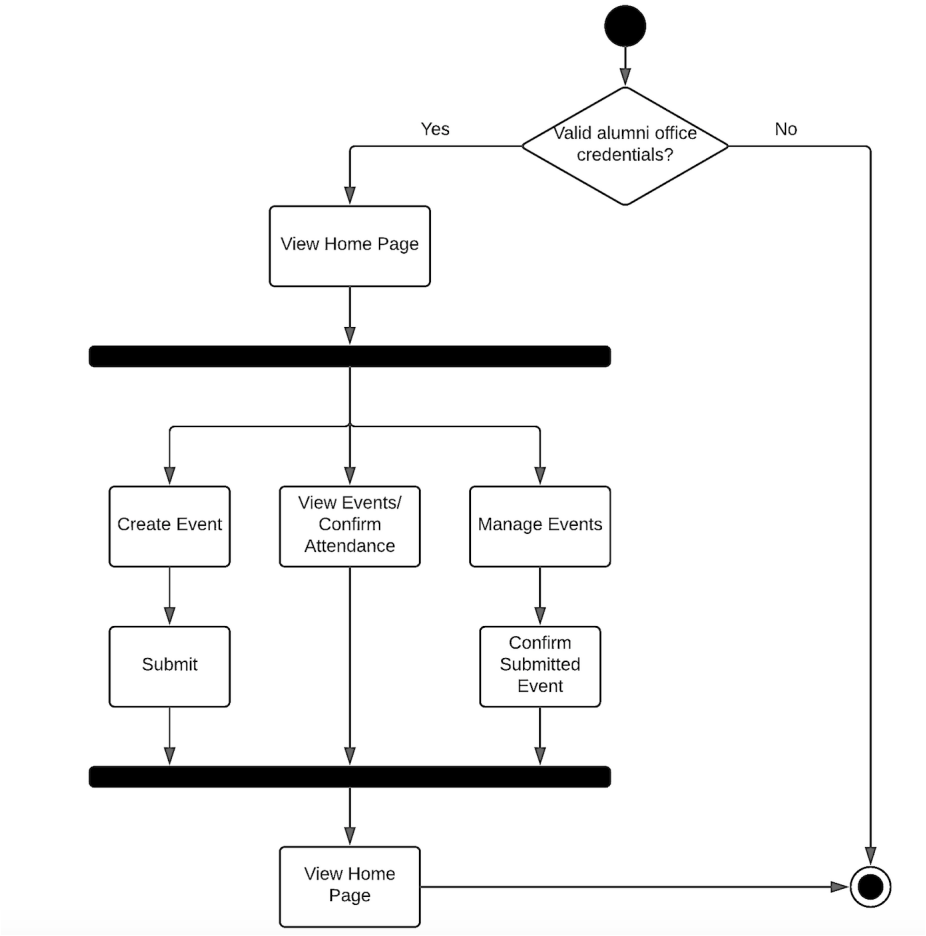

Figure 4.2: Activity Diagram for Alumni Office User

Chapter 5

Technologies Used

- **Django**
  - Main framework used to build the app. Uses a hybrid Model View Controller architecture. Helps to build databases and templates for pages using Python. Django also helps to manage the security of our application.
- **Bootstrap 4**
  - The CSS and HTML front end component library used to build the web pages and to create a responsive design.
- **SQLite**
  - The embedded database engine that will hold the information of users and events to be displayed by the application.
- **Apache and WSGI (Web Server Gateway Interface)**
  - Server system used to deploy our application for final use.

Chapter 6

Architectural Diagram

The system uses a hybrid Model View Controller architecture to handle and format information from the database to be placed inside of an HTML template and to pass the views to the user. When a user interacts with a view, that action is passed to the controller. The controller updates the model, the model gathers information from the SQLite database, then notifies the controller of this information. Then, the controller is able to update the view with this new information and the view is then delivered to the user in the form of a web page. We chose this architecture for our system because of the adaptability that it allows for. We can create simple templates for each of our predicted pages that will format and update themselves after the controller updates the view. This is also the underlying architecture for Django, which is the framework we are using to handle security and URL mapping.

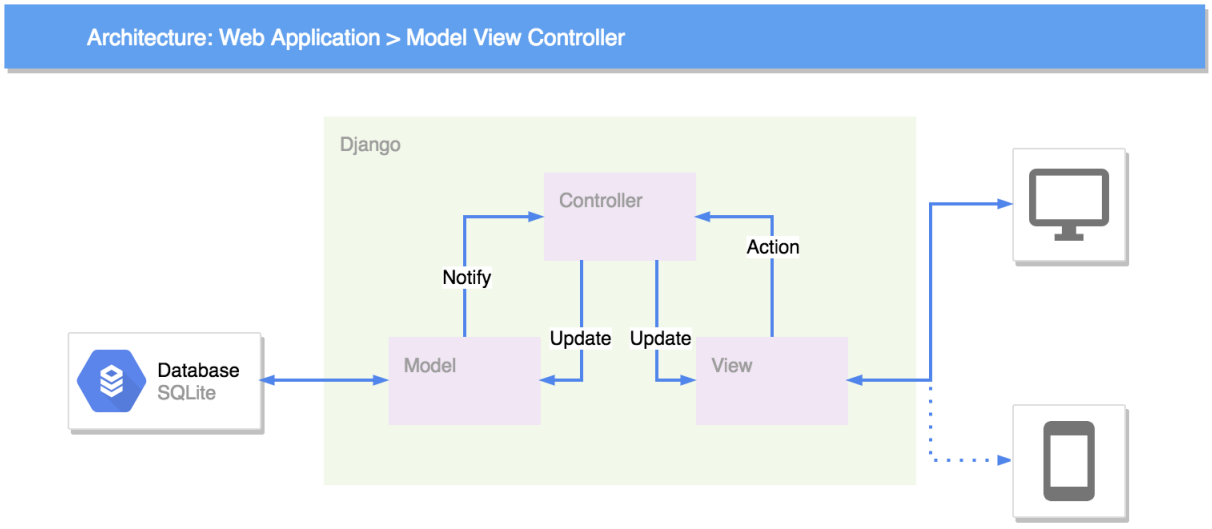

Figure 6.1: High Level System Architectural Diagram

Chapter 7

Design Rationale

7.1 User Interface

Our design for the user interface was chosen because it focuses on the specific functionality of the application without cluttering the screens. Traditional dashboards often clutter the screen and make key actions difficult to reach. The use of the upper navbar allows users to quickly shift from browsing directly to a more focused searching. The event submission page is another example of a focused interface. It is a single window used exclusively for the one form, and the user is redirected back to the events page after the form is submitted. The use of cards to represent events was chosen to allow users to quickly sift through a large number of events, and then select one to focus on within the specific events detail page.

7.2 Technologies Used

The first technology that we are making use of in our design is the Django web framework. This Python web framework provides templates for URL mapping and handling, database access, model building, and includes a view templating language to build dynamic pages. This framework also handles many mundane tasks for the developer such as basic security and scaling. By using a prebuilt framework such as Django, we can save ourselves time in the setup stage of the project and instead devote that time into providing well built features.

We will also be using Bootstrap 4 components because the library offers us the ability to use a uniform and well spaced selection of components to avoid design clash and confusing interfaces. Bootstrap 4 also uses a grid system to place components on a scaled webpage which allows us to develop clean user interfaces that will work on any sized device. The large library of available components will also allow us to build complex pages without needing to create any assets for ourselves.

Our database for the system will be held in SQLite. We are using SQLite because the expected size of our database is small. SQLite also comes built into the Django framework. This combination of framework and database will keep the necessary setup to a minimum as queries and tables are all handled by Python and built to match the data models.

SQLite is also being used because it does not require its own database server and can instead run alongside our web application on the Apache and WSGI server

Apache and WSGI have been chosen as our web servers because of the cross platform capability and lightweight server impact. Apache and WSGI are preconfigured within Django and will allow our web application to work on all operating systems and web browsers. We will be able to deploy our application onto any Apache web server configured to use the WSGI calling conventions.

Chapter 8

Description of System Implemented

Our system is built using the Django framework, used by major companies such as The Washington Post and Mozilla. We chose this framework for its simple scalability, rapid development, and large user base. The structure of Django keeps the front end and the back end very closely knit. All of the front end webpages are translated as templates by the Django view controller and populated with information from the views and the models. When developing a new feature, four sections of our system must be edited. The ”views.py” file must be edited to implement the functionality of the specific page. A ”.html” template file must be created for the view to populate and point to. The ”models.py” file must be edited to make any updates to the organization or structure of the database. Finally, the ”urls.py” file must be updated to connect the views to the templates and direct the user to the right place depending on where they navigate in the web application.

Our system opens up to the home page, a simple welcome section and a showcase of the three upcoming events. Users can either click one of these three events or choose to view a list of all upcoming events. When the user chooses to view an event they are redirected to the event detail page. This page contains a name, a description, a date, a time, and a location for the event. This page also contains a button allowing a user to record their attendance by submitting their name, major while at Santa Clara, school within the university, graduation year, and their email address. This report of attendance is submitted to the system and can be viewed in a report by the Alumni Office. Users can also choose to suggest an event that they would like to hold for all SCU alumni. They can select the Create Event tab in the navigation bar and enter their personal information and the event information. These events are reviewed by the Alumni Office and can be approved and posted, or declined and deleted.

If the user is a member of the Alumni Office, they can select the login page and enter the administrator credentials for the system. This gives them access to extra features not available to the general user. Once logged in, a user can view the Reports page, a list of events containing all event details, as well as a detailed list of all attendees. The alumni office also has access to the Approve Events page which allows them to review all alumni suggested events and manage their acceptance through a simple user interface. Finally, on the Events Detail page, administrators are able to

view and select an Edit Event button taking them to a page where they can change any of the event details they wish in case of updates to schedule or description.

Our system offers a simple, easy to use work flow for both types of users. By using Django for development, our system offers security features, scalability, and speed that are not found in other development frameworks.

Chapter 9

Testing Procedure

Our application underwent three types of testing: unit testing, alpha testing, and beta testing.

9.1 Unit Testing

We broke our web application down into separate pages, each of which we worked on separately and tested after completion. As we implemented each functional requirement, we verified that we were getting the desired outputs.

9.2 Alpha Testing

In the final stage of our project, our group performed alpha testing on our web application. We consulted our list of functional requirements and performed tests to verify that event creation and attendance confirmation for both alumni and alumni office users. We then checked that event verification, report generation, alumni verification, and log-in access were all working for only alumni office users.

9.3 Beta Testing

After performing alpha testing, we found willing participants among our peers, friends, and family members who would perform beta testing on our final product. We used our list of non-functional requirements to guide the participants in their feedback. We had them verify the functional requirements additionally. The participants were prompted to perform the designated use cases for alumni users; we then asked for feedback about their user experience, their opinions about the user interface, and about any issues they encountered.

Chapter 10

Difficulties Encountered

10.1 Misunderstood Requirements

In the early phases of development, we decided on a specific work flow for our users. According to the requirements as we understood them, our work flow would have been the best option to keep track of user actions and provide security. However, after our initial system demo our requirements were clarified with the client and we refactored our system to meet it. This refactoring took a significant amount of time and required a full system rework. Had we clarified these requirements earlier we could have saved effort, time, and confusion.

10.2 Corrupted Database

Immediately before our initial system demo our database became corrupted containing all of our demo events. This left our system completely empty with no examples to use during our demonstration. We were able to quickly add several events to use during our demonstration, but stress and time could have been saved by keeping a backup of our database on our code repository. From that point on, we kept a copy titled ”backup_db.sqlite3” in our repository to use in case of emergency again.

10.3 Lack of Familiarity with Framework

When beginning this project, it seemed that Django would be the best framework for our project. The Hybrid Model View Controller architecture and the close knit database and templating engine offered functionality that would typically be difficult to implement. However, only one member of our group had any experience with the framework. This left two group members playing catch up to learn the framework and begin development. Had we picked a framework that our entire team was familiar with, we could have saved ourselves the hassle and confusion of learning a new framework.

Chapter 11

Suggested Changes

With our project completed, there are some suggested changes to the system that could better improve usability, functionality, and organization.

11.1 Use of Dropdown Menus

Dropdown menus increase the number of page options we can display in the navigation bar at the top of each page. We could limit the clutter in the navbar when an administrator account logs in and improve the workflow for each use case. Currently, to reach each page, we must navigate through a series of windows. This makes sense when it comes to the organization of options, but it increases the number of clicks necessary to reach each page. Dropdown menus can allow a user to jump directly to the desired page with fewer clicks to speed up the process.

11.2 Event Categorization

Currently events are only organized by date. Users must sift through a large number of upcoming events to find the one they are looking for. If we provide categorization of events, users can look through only the kind of events they want to see. We could use categories such as dinner and drink events, networking events, keynote presentations, and workshops. By providing this feature, we could increase usability and speed up a user's ability to find the event they are looking for.

11.3 Increased Modularity

Currently, due to the limitations of the Django framework, many different types of features are included within the same packages and directories. By separating the different features into different applets, it would be easier to edit and read only the code we are interested in. It would reduce collisions in Git merges, and simplify the development process for all involved.

Chapter 12

Lessons Learned

Based on the problems we faced throughout the development process, we have identified several lessons we learned while working on this system. These lessons are listed and explained in the sections below.

12.1 The Importance of Clarifying Requirements

Through our difficulty with misunderstood requirements we learned the importance of frequently clarifying requirements with the client. We worked for several weeks on development with a poor understanding of an important requirement. Because of this, we had to go back and rewrite much of our system to meet this requirement. Had we worked more closely with the client through all stages of development, we could have avoided with issue all together.

12.2 Unit Testing

Our testing for this system relied on simple testing procedures defined early in the design process. As our system evolved, we did not adapt our procedures to the changing structure. This left us with several features in the system that were untested. We were able to quickly identify and fix bugs as they arose in alpha and beta testing phases, but had we adapted our unit testing procedures early on we could have identified them sooner and fixed them while the code was still fresh in our minds.

12.3 Synchronizing Git Commits to Avoid Collisions

This project was one of the first experiences that many of us have had with group programming. We decided to use Git and GitHub to host our code repository, help with version control, and allow us to share our code among all developers. We learned that it is important to specify what files each developer is editing at a given time to avoid merge conflicts when committing code. We also learned to commit early and often when working on a feature to allow others the opportunity to edit a file for their own work without causing a merge conflict.

Appendix A

Installation Guide

To install the system locally we must first install the required dependencies, place the code in the correct directories, create an administrator account for the Alumni Office users, and start the web server. The commands required are included in the installation guide that follows.

A.1 Install the Correct Python 3 Version (Python 3.7)

This system runs using Python 3. We must first install Python 3 on the machine to ensure that the system will run properly. The easiest way to do this is using Homebrew. If you do not have Homebrew installed on your machine, open the terminal and run the following command to install it.

A.1.1 Mac OSX

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/
    install/master/install)"
```

A.1.2 Linux

```
sh -c "$(curl -fsSL https://raw.githubusercontent.com/Linuxbrew/install/
    master/install.sh)"
```

Next, we must add Linuxbrew to the PATH and the profile for Bash.

```
test -d ~/.linuxbrew && PATH="$HOME/.linuxbrew/bin:$HOME/.linuxbrew/sbin:
    $PATH"
test -d /home/linuxbrew/.linuxbrew && PATH="/home/linuxbrew/.linuxbrew/bin
    :/home/linuxbrew/.linuxbrew/sbin:$PATH"
test -r ~/.bash_profile && echo "export PATH='$(brew --prefix)/bin:$(brew
    --prefix)/sbin'":'"$PATH"' >>~/.bash_profile
echo "export PATH='$(brew --prefix)/bin:$(brew --prefix)/sbin'":'"$PATH"'
    >>~/.profile
```

If this does not work, we must install the dependencies using the following command. This command is only functional for CentOS operating system, such as the Santa Clara University Design Center Linux computers.

```
sudo yum groupinstall 'Development Tools' && sudo yum install curl file
    git
```

A.1.3 Next Steps for All Operating Systems

With Homebrew installed, run the following command to install Python 3.

```
brew install python3
```

To ensure that Python was installed successfully, test with the following command.

```
python3 --version
```

This should return "Python 3.7.x" with the value of 'x' depending on which version was installed. Any version of Python 3 will work.

A.2 Install VirtualEnv and Create the Environment

For the system to work correctly, we must install dependencies only within a virtual environment. If installed globally, the system will not work. To install VirtualEnv, use the pip3 command in Python 3. Run the following command.

```
sudo pip3 install virtualenv
```

Next, navigate to the directory in which you would like to install the system. Then create a new sub-directory in which to create the environment and then navigate inside.

```
mkdir alumniAttendance
cd alumniAttendance
```

Now, create the virtual environment by running the following command.

```
virtualenv ENV -p python3
```

Navigate inside the the ENV folder, create the application directory, and start the virtual environment using the following commands.

```
cd ENV
mkdir app
source bin/activate
```

When you are ready to stop the virtual environment, run the command:

```
deactivate
```

Finally, unzip the program files inside the "app" directory. With the environment running and the files in place, we can install the necessary dependencies.

A.3 Installing the Dependencies

To install the required dependencies for the system make sure that the virtual environment is running. Next, navigate inside the program files using the following commands.

```
cd app/AlumniReporting
pip3 install -r requirements.txt
```

A.4 Setup Database and Create Administration User

With the dependencies installed, we must next setup the database for use with the application. Django has built in commands to help with this process. Run the following commands.

```
python3 manage.py makemigrations
python3 manage.py migrate
```

With the database set up, we must next create a base user for the Alumni Office to use for administration. Run the following command and follow the prompts to set a username, email, and password for the account.

```
python3 manage.py createsuperuser
```

A.5 Start the Application Server

We must run one final command to start the application server and access our application.

```
python3 manage.py runserver
```

Now, open your web browser and navigate to `http://127.0.0.1:8000/attendance/`. You should see the homepage for our application. The installation is now complete.

Appendix B

User Manual

B.1 For All Users

B.1.1 Register Attendance

1. From the ”Events” page or the home page, click on the event which you are attending.
2. When the event page is open, click the ”Register Attendance” button on the bottom left of the event details.
3. Enter your information in the designated text fields and then press the ”Submit” button.

B.1.2 Create Event

1. Click on the ”Create Event” tab on the top right of the page.
2. When the ”Create New Event” page is open, enter the correct information in the designated text fields and then press the ”Submit” button.
3. The event will show up on the home page once an Alumni Office employee has approved it.

B.2 For Alumni Office Use Only

B.2.1 Login

1. Click on the ”Login” tab on the top right of the page.
2. Enter your Alumni Office credentials (username and password).
3. Click the ”Login” button underneath the username and password text fields.

B.2.2 Approve Events

1. Click on the ”Approve Events” tab on the top right of the page.
2. When the ”Approve Events” page is open, you will be able to approve or deny the events by clicking the ”Approve” or ”Deny” buttons.
3. To view more information about the person who submitted a specific event, click on the name under the ”Submitted By” column for that event.
4. To view more information about a specific event, click on the name of the event under the ”Event Name” column.

B.2.3 Report Generation

1. Click on the ”Reports” tab on the top right of the page.
2. The ”Reports” page should open with a list of all previous events.
3. To view a list of attendees for a specific event, click on the number under the ”Number Attended” column for that event.
4. To view more information about a specific event, click on the name of the event under the ”Event Name” column.

B.2.4 Manage Events

1. From the ”Events” page or the home page, click on the event which you want to manage.
2. To edit the event, click on the ”Edit Event” button.
3. To delete the event, click on the ”Delete Event” button.