I. Abstract

This project involves developing a website for Whim n’ Rhythm, a Yale a cappella group. The existing website is not particularly useful for current members because it is geared towards providing information for public visitors and is hard to maintain. The site created in this project is distinguished from others of its type for its design which provides utility to current members, easy maintenance for future groups, authentication, and robustness. This website consists of three parts for the visitor – the public site, the private site only for members, and an admin portion. The public site displays announcements published by members about upcoming performances, and displays information about all current members of the group. The admin part of the website allows for easy maintenance of information in the database, for example, the current members of the group and their information. The private part of the website is for members with a valid username and password, has a schedule of upcoming rehearsals and performances, and an internal announcement board.

The project is built using Django, a Python web framework, and Git as the revision control system. The backend includes a Postgresql database which stores three models: member (which is displayed on the public part of the site), private_post (which is on displayed in the member site), and public_post (announcements which are displayed on the public part). The website is hosted on Heroku. The final report describes in further detail the motivation, process, technology, final result, feedback, and future steps of the project.

II. Introduction/Motivation

Currently, there are many Yale club websites that are efficient, but are not particularly easy for future years to update or maintain. Also, while many of these websites are useful for the public to view, they do not have any advantage to the current members. The website I have built for Whim n’ Rhythm is reliable, easy for future generations of the group to maintain, and useful for current members as well as the public.

In order to create a website that was useful for current members as well as the public to view, I designed a part of the site that was only for current members. On this part of the site, members can post announcements and view the current schedule for performances or rehearsals. After a member logs in, they can view this part of the site. The non-private, public part of the site informs people of news, updates, bios of the members, and lets them listen to music directly on the website.

I have also ensured that the website is easy for this and future generations of group to maintain by customizing the administrative section of the website. The admin part of the website allows an administrator of the group to log in and update member bios and certain public parts of the website.
This website is distinguished from others of its type for both of the above reasons, and for the technologies it uses. In this report, I will discuss the different technologies used, final result, testing, and improvements.

III. Process and Technology

This website was built using Django, which is a framework for web development in python. Django dynamically generates the HTML displayed on the webpage, and uses the Model-View-Controller framework. I chose to use Django because it has many advantages which were useful for this project specifically. Django allows easy integration with the database so that there is no need to directly write SQL queries. Also, Django has an admin part of the website that is automatically generated and customizable for every web project. This administrative website allows for easy access to adding, updating, or deleting any instances from the database.

I used HTML, SCSS, and Javascript to write the web pages. I used SCSS to generate the CSS for the project for a number of reasons. SCSS is CSS-like language that compiles down to CSS, but provides many advantages for development and maintaining consistency of rules. In particular, I took advantage of variables to reduce the amount of repeated code, and used nesting to help manage my code and easily scope the formatting to specific parts of HTML.

I initially intended to use React as well, to create a richer experience for viewers (both public and members). After exploring the project more and working on an initial prototype, I realized that it made more sense to focus on certain aspects, like member only posting, which would provide more immediate and long term value to users of the group. While working on the prototype, I realized that the advantages of React are not particularly useful for the website I ultimately designed.

I used Git as the version control system to ensure efficient development.

In order to have members log in to view the private site, I added authentication to the website. Django has an “auth” package that allows for authentication with a login and password. After the user enters the username and password, I authenticate whether it is valid with the auth package. The user can also logout of the site. This auth system is also used for the admin part of the website. I use the auth package to check if the user is already logged in when they return to the site.

I used an object relational database, Postgresql.

There were 3 primary models in this website that I stored in the database. They were Member, Private_Post, and Public_Post. Each Member has a name, whim name, hometown, college, major, voice part, and picture. Every Member in the database corresponds to a current singer in the group, and these are primarily used for the public page “members” which displays the bios of every singer. The Member model is also used as the author for every Private_Post, explained next.

Private_Post consists of a title, published date, content, and author. These are the announcements that are posted on the private part of the website that only members can access with a username and password. Each author is a Member instance. These private announcements can be published by any member of the group directly from the private member site.
Finally, the Public Post has a title, published date, and content. These are announcements published in the “news” page of the site and can only be created by an administrator of the group in the admin page.

I have included a schema of the database here for clarity.

```
Database schema
```

Every instance of these 3 models can be easily deleted, added, or updated in the admin part of the website. This provides easy maintenance for current and future generations of the group. Public posts, private posts, and members can be added, removed or updated as required by an admin.
Screenshots of the admin site. Example of modifying a public_post instance.

I embedded Spotify, iTunes, and Youtube playlists onto the site so that the visitor can listen to music without leaving the site. These widgets will allow the user to play live and recorded songs by the group.

The website is hosted on Heroku, a layer above Amazon’s Elastic Compute Cloud (EC2). Heroku allowed me to move quickly by taking care of many of the administrative parts of hosting the website (setting up a server, hosting a database, adding alerts for downtime), ultimately allowing me to focus on the application.

Heroku’s servers, or dynos, that run commands and host the web application have a filesystem that is temporary. In other words, when a user uploads a media file, that file will get stored on one of Heroku’s dynos. For example, when someone logs into the admin site and creates a new member, they upload an image and it gets stored on a Heroku dyno. Because the website is not guaranteed to run on the same dyno each time and because the filesystem of each dyno is temporary, these files will not get stored somewhere permanently. As a result, I stored all the user uploaded media files and static files in Amazon S3. Amazon S3 is a storage service using buckets. I created a bucket for the Whim website that holds all of the user uploaded images and static file files/images. So now, when a user uploads an image on any part of the site, it gets stored in the Amazon S3 bucket.

IV. Final Product

There are three main parts of the website, each consisting of its own subparts. The public part of the website is the part that is immediately viewable by anyone who has the url.

The public part of the website has 6 pages — home, about, events, members, contact, and music. The about page includes the history of the group and general information about the group. The events page displays all of the Public_Posts currently in the database in descending order so
that the most recent published post is displayed at the top. These are announcements of upcoming performances of the group.

Screenshot of the public announcements page

The members page displays information about each Member that is stored in the database. This page will have a photo for each member, along with a short amount of biographical information stored in the database. The contact page has information about who to contact in the group for performances. Finally, the music page allows the user to listen to full songs without leaving the website. I embedded Spotify and iTunes widgets so that the visitor can play the group’s music. I also included a YouTube playlist so that the visitor can watch videos of past performances without leaving the site.
The second part of the website is the private members site. This members site requires a valid username and password to enter, and has content relevant to the current members of the group. I embedded a Google calendar which has the schedule of all the upcoming performances and rehearsals. Also, this page displays all the Private_Post objects in the database. These private posts are created by members and displayed in descending order, with the most recently published post appearing first. This announcement board will be useful for the group to post messages for only each other to see. Finally, there is a place for members to make their own announcement for the private message board. This adds a new private_post instance to the database.
The last part of the website is the admin site. The admin site, created by Django, has been customized so that a user with admin permissions can modify, update, add, or delete Member, Private Post, or Public Post objects in the database. From the admin site, a user can also update, modify, add, or delete users’ credentials (username and password). The admin site will be useful for maintenance in the following years.

The site can be seen at the following URL: http://rocky-bastion-90400.herokuapp.com/

V. Testing/Discussion

This website has been developed so that future generations of the group will find it easy to manage and update. The backend and admin have been customized to make it easy to add or delete members or announcements, and the private members site has been created to make a more useful site for the current members.

The feedback I received was very useful. There needed to be a clearer way to navigate the website since the current design made it a bit confusing. In response to this, I added the word menu to the top left button to make it more obvious to users. Also, the loading time for the background image was very long upon arriving on the website. In order to fix this, I reduced the size of the image so that it would load more quickly.

I solicited feedback from some current members of the Whim group with the following plan: visit the homepage, and then every public page and provide any feedback; go the admin site and create a username and password; go to the member site, login, make a private post. From these steps, I found that it needed to be more clear what “author” needs to be for the page where a user can make a post. In order to fix this, I have a dropdown list of the current members as the author field instead of a text box. There were also UI changes (for example, making the make post content box bigger to indicate that is the main part of the announcement). Also, one member suggested that after successfully making a post, they should be redirected to the member site, which I implemented.
Load testing the website showed that it is able to handle many sequential or concurrent requests. 1000 hits, with a concurrency of 10, revealed an 11 millisecond time per request. The transfer rate was 236.98 Kbytes/sec. I ran many tests to determine the trend of the time per request (in milliseconds) vs. the number of concurrent requests. The website was able to handle large numbers of hits and concurrency.

Looking forward, there are a few areas which could be improved. I want to associate each Member instance in the database with the username/password that they create so that they are linked in that way. Currently, they are separate so there is an opportunity for increased consistency there. Also, I would like to more clearly announce upcoming performances or events to visitors on the homepage. Before enabling this website for users (transferring to the domain name whimnrhythm.com), I plan to add Google Analytics to track which pages are most popular and where traffic is coming from.

VI. Acknowledgements

I would like to thank my advisor, Professor Holly Rushmeier, for all of her guidance with this project.

VII. References

1. https://www.codeschool.com/blog/2016/08/03/why-django/