GroupM8 Final Report

Michael Tummarello, Julian Carney, Juan Sanchez

May 14, 2017

1 Team Building and Motivation

The story of how our team was formed is also the story of what motivated our project. Prior to coming together, each member of our team had been looking for a project group for some time without success. None of us knew many people in the class or had sufficiently outstanding COS resumes to convince people we didn't know to let us join their groups, and the only tool we really had at our disposal to find groups was a thread on Piazza. In the end (and only barely in the nick of time, we might add) we did find each other through that thread, but it took weeks and was a great deal of work. Upon reflection, we realized that this sort of scenario was a fairly common one in a university setting - students often don't have a convenient mechanism to reach out to other students in their courses and form groups to work on projects or study, only going through the effort of reaching out to people at the eleventh hour, if at all. This shouldn't happen, least of all in a place like Princeton, so we wanted to create an accessible platform to help future students avoid having to go through what we and many other students did. That's how GroupM8 was born.

Looking back, we were comically unprepared for the task ahead of us. No one in the group had any meaningful experience with any of the languages and systems we would be using in the following days, nor did we have much time to learn them given that we were only three-strong and that two of us were juggling junior projects. Nevertheless, when the time came we pushed our limits, accepted that people only really need to sleep three or four nights a week, and did what needed to be done to produce a quality product that could genuinely help others.

2 Milestones

<u>Preliminary Preparation</u>
Before we could think about coding, we first had to take care of logistics. We knew we'd need to store some information, and that a database was therefore necessary. After some research, it seemed like MySQL was the best choice, being a well-regarded and well supported database system that maintained a Python module to make back-end system integration easy. Setting up the database went remarkably smoothly, and MySQL generally seemed to live up to its reputation. We needed some sort of unified repository so that we could share files while we were still working on getting the server up-and-running, so we made GitHub accounts and created a repository for the three of us. We (perhaps wisely) didn't try to do anything too fancy with GitHub immediately, just using it in the same way one might use Google Drive. In that capacity, it worked well and was not difficult to manage.

Servers, Web Frameworks, ICANN('T)

Of course, we needed a server to run everything on, so we explored some of the different hosting services: the big names like AWS and Google Cloud, as well as some of the less well-known services for amateurs and hobbyists. Though we initially signed up with AWS, it felt like we were trying to use a (expensive) jet engine to power a paper airplane - AWS offered so much that it was overwhelming, and it was difficult for us to know what we actually needed and what was just being well-marketed. We eventually decided to switch over to DigitalOcean, which had built a name for itself as an econo-hosting service that offered all the basic functionality needed for a group our size.

We decided that Flask was a good choice for our web framework since it was also in Python and was reputedly fairly straight-forward. The only warnings we read about Flask was that it was a pretty light-weight framework that wouldn't hold up to heavy traffic, but we weren't anticipating heavy traffic and it was in the same language as our MySQL module, so we went with it. Initial setup on the server went off without significant difficulty, and integration with MySQL was also quite simple.

Lastly we needed a domain name, so we went to the Internet Corporation of Assigned Names and Numbers (ICANN) to purchase a domain name. As it turns out, the Internet Corporation of Assigned Names and Numbers doesn't actually handle much of the assigning of names and numbers - instead, they contract it out to several thousand organizations of wildly variable repute. After several nearly successful identity theft attempts, we found an organization called GoDaddy. Encouraged by its highly professional name and vomit-green color scheme (and the fact that the consensus seemed to be that it was actually the best DNS provider out there for the money), we managed to find a relatively uncorrupted version of our name that was available for purchase - GroupM8.org. With a little tinkering, we were able to set up the requisite linking and were off to the races.

UI Templates and Basic Functionality

At this point, our group split to pursue different aspects of the application; one member started developing html templates on his personal machine, while the remaining two members worked on getting the backend functionality of the code.

Before splitting, the group spent some time discussing and sketching various UI designs, trying to balance aesthetics with ease and speed of use, before settling on a system that relied on four pages: a login page (we weren't trying to use CAS at this point), a personal home page for the user, a group page, and a search page to let members of groups search for users to add (we would have liked to integrate the functionality of the search page directly into the group page to minimize redirection, but there simply wasn't room). Equipped with pencil sketches of the currently non-existent page templates, the UI team of one began churning out html. Though he, like everyone else in the group, hadn't the foggiest idea what a "div" was when the project

started, he became intimately familiar with the principles of HTML very quickly, and soon learned to embrace the mind-numbing boredom that was pure HTML.

Meanwhile, the backend team was having a slightly more engaging time constructing the core functions that would actually make the website more than a pretty face. After tweaking the database design to be just right, they wrote the code for updating and/or looking up information in the database for each basic functionality on our website. These basic functionalities included creating a user, creating a group, joining a group, leaving a group, adding a course, leaving a course, searching for a group, searching for members, sending/processing requests, creating/deleting events, and toggling course/group availabilities.

Website Launch, jQuery and JavaScript

After the code for the basic functions were written, we decided to launch our website. However, we encountered a huge problem here - the website would not display when we accessed the domain. We tried looking up the problem on the web, asking our peers on Piazza, and asking our project manager to no avail. With the inability to access the website making working on the website somewhat challenging, we switched gears to polishing our database structure and the Flask and jQuery code for the basic functionality of our application. After being stalled for a week and a half, we finally identified the issue - we were configuring the website for an older version of Apache2 and had to change the syntax to match the newer version. With that done, the website worked just fine.

With our website up and running, we made huge progress. We completed a simple outline of our website's UI and worked on how to deliver the necessary data between the website and our backend (Python/Flask). To make our content more organized, we decided to format the data into JSON. We used jQuery to retrieve the data and a combination of jQuery and HTML forms to deliver the user's input to our backend. The whole linking process took a while since all of us were pretty new to linking up the frontend and backend in this way. The entire process sped up once we got a few examples working - apparently learning web development is a lot like riding a bike, since it's terrifying when you don't know what you're doing and deceptively straightforward once you do.

Polishing and Refining

After we got everything linked up between the frontend and the backend, we had users try out our website. From their feedback, we polished up our website's UI and added additional features like the upcoming events section on the user's home page (suggested by our project manager) along with a number of quality of life adjustments suggested by our other testers. Of course, our users found many bugs during their visit to our website. We compiled a list of the bugs that were reported and fixed them.

In order to more efficiently serve multiple clients, we had to configure Apache2 to properly implement multi-threading. While we were still in the creation process, our server would run into load issues whenever multiple users were on at the same time, and would occasionally go down

when all three of us were testing at the same time. To solve this problem, we changed the configuration of our Apache2 server to support multi-threading and multi-processing, a marked improvement over the system of launching a single child process for every client we had used up to that point. After further tweaks to the server settings, our server's performance increased even more, making it easier to test with multiple users at the same time.

As we were fixing these bugs, we tried to integrate CAS to facilitate the login and logout process for our website. We ran into a couple of obstacles implementing CAS for our website. Most of these had to do with whitelisting our domain through OIT so that CAS did more than inform users that our application wasn't authorized to use CAS. It was a major stumbling block to having a clean working website, but after a flurry of emails and a fair bit of hair-pulling OIT straightened everything out.

Demo Day

The day of the demo and the weekend leading up to it was easily the most stressful part of the project. The remaining little bugs were quickly fixed, but lacking CAS support was really bad, considering we hadn't have a working log-in or a registration screen that didn't involve CAS. Our prototype had a skeleton dummy as a login page, not meant to be seen or accessed by the users, and it certainly wasn't an issue that we could just write off for our demo. In the hours leading up to our demo on Monday, we got CAS working smoothly, barely making it in time for our demo to properly demonstrate the usage of CAS. The other issues were easy to fix and were done just in time to prepare a demo for GroupM8.

Since our project was very front-end focused, we elected to focus on demonstrating functionality rather than wasting a good portion of our demo on describing the enthralling syntax used to call databases or the intricacies of writing HTML.To that end, we decided the best mode of presentation would be a narrative demonstration of a "real life" user walking through the functionality of the application, illustrating motivations and use-cases for various features.

3 Surprises!

CAS

One of our biggest surprises was how difficult it was to actually get our domain CAS whitelisted and working. When we first contacted OIT about getting whitelisted we were told that our request had been approved and we were all set. However, after struggling with the profound lack of documentation on the proper integration of CAS and Flask, we eventually encountered a screen informing us that our application was not authorized to use CAS. Puzzled, we reached out to OIT again. After several days of emailing and testing, they identified a problem with their whitelisting and fixed it, allowing us to actually test and improve the way in which CAS was integrated.

Another big surprise here was that most of our ideas of what to do with CAS were mostly correct. We were afraid that something had gone drastically wrong with the way we coded CAS,

but it was actually an administrative error on the part of OIT. Finding out that we hadn't catastrophically messed up in some silly way was a big relief.

Github:
Going into this project, only one of our members had any previous experience with Github, and it wasn't very exhaustive. While conceptually Git is fairly straightforward, in practice it ended up complicating things much more than expected. The primary issue lay with the code working on the code on the live server as well as on local repositories. After fruitless days spent trying to get everything synced, we had only managed to have Git automatically undo any changes we tried to make to our code base. As you might imagine, this was less than ideal since it prevented us from writing a single line of code. In the end we decided to limit our Git usage to pulling changes made on the server and storing them locally. Not having more robust version control was tricky, but with only three people in the group we managed to get by without any particularly massive catastrophes.

4 Decisions and Consequences of Language and System Choices, Good and Otherwise

Although MySQL, HTML, and CSS were tedious to format properly, once we started getting them working properly it was very easy to make changes and to customize as we wanted to. CAS was slightly challenging for the reasons described above, but it worked well enough to justify the time committed to getting it working. JQuery scripts had an alarming tendency to not fire when they were supposed to, but were effective when they did. We eventually learned the various rituals needed to convince the jQuery gods to work with us, and on the whole are satisfied with what we were able to accomplish with it. It interacted well enough with our backend Python and was essential to our website's functionality. While Flask initially seemed nice to set up, it ended up having such abysmal performance that we would have been much better off with a different framework. Bootstrap was ok to set up an initial outline for our website, but its lack of customizability ended up being very painful in the long run, meaning a different formatting system would have been better.

5 Future Goals

Integrated Chat
The feature we would most have liked to implement would be an integrated chat for each group. While not essential to the functionality of the application, some way for members of a group to communicate directly with each other would greatly enhance the utility of our site, preventing users from having to resort to a third party chat system.

User Profiles
Another important future goal is the creation of a "user profile" page, containing relevant user-supplied information for the selection or rejection of the user as a group member by other users. The page would contain fields like "experience" and "interests," along with a short bio

section. This would allow other users to determine whether a particular user was a fit for their group in terms of level of expertise, expectations, and goals.

Dynamic Updating

Our website currently does not update its content unless an action has been taken such as refreshing the page or clicking a button that ends up refreshing some of the contents on the page. In the future, we will implement dynamic updating of the website's contents. For example, if a user requests to join a group, the request can be seen by the group's members without having to refresh/reload the page. It is crucial to keep the contents updated so that the website can run smoothly without encountering conflicts such as sending a request to a group that has already been deleted.

Notifications

Currently, our application relies on users actually having the application running to see that they've been invited to a group or that someone has requested to join their group. For small groups in particular, it may not be reasonable to assume that anyone will check for new requests to join a group in a reasonable amount of time after a request has been sent. To solve this issue, we would have liked to incorporate an email notification system to notify users about requests and invitations so that they could respond in a timely manner.

Increased Support for Mobile Users

Though our application runs on mobile devices and maintains a consistent layout, the UI gets pretty mangled by the differing screen proportions. Computer screens, which we focused on in our development, tend to be wide and short, while smartphones tend to be tall and narrow. To increase the usefulness of GroupM8 and make it more versatile, more work could be done with the UI to alter the layout and proportions of the pages for smartphones.

6 Lessons Learned

Don't skimp on the web framework infrastructure. While Flask was indeed fairly simple to set up, it performed poorly under any semblance of load. If our application were to ever see notable popularity, the performance hit from the increased traffic would be very substantial - a more robust framework, like Django, would be a necessity in practice.

When it comes to formatting tools, customization is better than convenience. Though Bootstrap was initially helpful because it provided relatively easy ways to create basic UI elements, it was very resistant to attempts to customize or enhance its functionality. This limited our ability to refine and adapt our UI in a significant way, particularly since it played very poorly with CSS - if a page tried to use both CSS and Bootstrap, unfortunate complications and interactions were guaranteed. Instead of Bootstrap, we would have been better off using pure CSS for all formatting. Though it would have taken more time for us to become comfortable with CSS, it would have ultimately paid dividends in the form of increased compatibility and customization.

Given the difficulties we had with integrating CAS, trying to integrate it earlier in the development process would have saved us a lot of stress and headache. Additionally, we wound up having to do a lot of backtracking once we replaced the makeshift login and session control systems we used before CAS - switching to CAS earlier would have saved us time in that regard.

7 Advice for Next Semester's Students

Project Selection

We were a little concerned that we weren't being ambitious enough when we first pitched the idea for a simple study group making platform; boy, were we wrong. Our "simple" application needed twenty hours of work from each of us each week to stay on schedule. Be sure to take on a project that can be reasonably accomplished, and don't be afraid to halt the development of features not essential to core functionality if you find yourselves struggling to keep up.

Stack Overflow

Stack Overflow was a great resource for us when we were learning new languages and systems or struggling with a particularly tricky concept, but take everything there with a grain of salt - oftentimes the solutions posted are for older versions of systems and no longer work, and sometimes they're a hammer when you need a screwdriver. Don't be afraid to learn from outside resources, but don't rely on them blindly.

8 Closing Thoughts

Though this project sucked away our social lives and our sleep, it also equipped us with invaluable tools. While an easier project or more time might have seemed more desirable at times, we've each come to realize that the rigor and challenge we railed against every day will serve us well in the years to come. Thanks you for this opportunity - it was truly the gift of a lifetime.