QuickCanvass: Final Report

COS 333: Spring 2017

Team Information:

- Jessica Ji (jmj5@princeton.edu) - Project Leader
- Luisa Goytia (lgoytia@princeton.edu)
- Sam Russell (samueljr@princeton.edu)
- Grace Turner (gracekt@princeton.edu)

ORIGINAL PLANNING

It is amusing to reflect on the progress of our project and the evolution of our ideas. We were vastly overconfident about most aspects of the development and deployment of QuickCanvass. The initial idea was for a canvassing platform for anyone to use, not just Princeton students. It would use GPS for location and contain a list of registered voters and their addresses. However, it soon became apparent that established platforms for this already contained a much more robust dataset than publicly available voter lists. After discovering this, we decided to restructure QuickCanvass to only target Princeton students for two reasons. Firstly, tools for standard political campaigns are in great abundance, but there was nothing to serve the need for Princeton students. Secondly, by taking advantage of the particular nature of elections at Princeton, we could build a much more complete tool than we could have for a generalized campaign.

Another misconception we had was the level of difficulty and costs involved in deployment. The learning curve for deployment using Google Cloud platform was very steep. It took us quite a while to successfully set up all the components needed for deployment and to work out the bugs involved. Additionally, cost became an issue post-deployment. For some reason, Google Cloud Platform costed approximately $50 dollars each week after the project had been initially deployed. App Engine, our deployment engine, commandeered the majority of the cost. It was very difficult for us to discern how to reduce costs because the costs being reported did not match any of Google's quota or cost documentation. Thus, after three weeks using Google App Engine, we transferred our project to the much cheaper (~$5 per month) Python Anywhere.

One of our biggest victories was modularizing the project so as to arrive at the minimum viable product (MVP) first before building new features. Experience is a great teacher, especially when it comes in the form of a multi-week large scale project. We knew intellectually from our lectures that modularity is key, but not until we took on QuickCanvass did we know it in our bones. Instead of half-building a lot of different features, we built a "smaller-scale" but still workable version of the same product, and then added more features until it was fully fleshed out. For example, with the "edit campaign" page, we first built the front-end "look". Then, we created a

basic connection to Django forms, and then we connected the form so that it would update the database. Afterward, we protected the database from malicious updates, and so on and so forth. On a more macro level, we built the basic search function before we increased the possibility for customization, such as filtering by "school year". While we balked at the slow pace in the beginning, we found it much easier to keep track of progress and debug the code then a faster pace. Our modular focus helped us avoid the worst pitfalls and carried us through to a successful demo day.

MILESTONES

The differences between a one-week coding assignment and a long-term project include the scope, group communication (for group assignment) and time required for completion. Before this assignment we were, more or less, used to working individually on assignments. This lead us to not meeting our first personal milestone. Our original idea for that first week after spring break was that we would work individually on small tasks and put it all together, but we neglected to account for the interconnectedness of the project, which early on lead to delays. This, combined with our overconfidence, led us to set very high expectations that were ultimately not met. We showed up to our second meeting with Robin, our TA, with almost no new code. After that week, we regrouped and tried having daily 2-4 hour meetings where we programmed in the same room. We discovered that this was more productive than working alone. From then on we consistently hit our milestones. The quality of our communication increased dramatically and we developed a healthy progress pace.

Significant milestones in our project included:

- April 3-8: Finishing most of the frontend, successfully connecting everyone's local machine to Google Cloud Platform services (most critical Google Cloud SQL), beginning to link the frontend to the Cloud SQL database, and successfully deploying to Google App Engine
- April 9-15: Completing the login/logout and user database storage, completing the ability to create a campaign
- April 16-22: Transferring deployed web version to Python Anywhere; finishing the manager and volunteer dashboards, completing the ability to create a survey for the campaign; completing the ability to survey individuals after searching for them
- April 23-29: Allowing managers to add volunteers to their campaign without having to give them the campaign code and have the volunteer manually enter it; cleaning up the search function to capture more irregularities in the data (eating clubs, Witherspoon Street, etc.); ability to download survey data; displaying campaign process
- April 30-May 6: Further refining the search function; adding polishing touches to the UI (customizing the layout to different screens, standardizing our CSS, etc.)

DESIGN, INTERFACES, LANGUAGES, SYSTEMS AND TESTING

The beauty of the project is the process, not the result. Most of us had very little Django, HTML, or CSS experience. Back-end coding became less and less mysterious to us as we gradually

learned more about Django. Front-end coding became less an exercise in frustration as the "look" of the website slowly coalesced around custom edits to the standard bootstrap CSS.

Front-end development began with a paper design of our vision for the website. Luisa and Grace (the main front-end team) began with the basic paper wireframe to decide the user flow before actual coding, or rather, googling began. With the design in mind, we began to create simplified, non-dynamic, versions of each page to get a sense of what we liked and what needed to be redesigned. Because the back-end did not exist yet, our site's early components functioned as placeholders. As the database was created and the back-end functionality was developed, we began to connect the front-end with the back-end. This was exciting and completely new to us. We replaced the placeholders with dynamic code that could communicate with the back-end. This process was both useful and ineffective. On one hand, superficially coding the UI allowed us to dive into CSS and arrive at the final UI design we all liked. On the other hand, our inexperience and ignorance of what django could "take care of" for us, such as with forms and models, made the design process longer than it would have otherwise been, as we would often code up a hack-and-slash first version, and then later replace it with another that uses the smoother Django functionality. Luckily, almost all rework occurred because of our ignorance and not because we had an unclear idea of user flow. Overall, we think the front-end experience was very rewarding.

Obtaining dorm, class year, and other such data about Princeton students was rather easy and was obtained from a site (http://bparks.mycpanel.princeton.edu/search/data.json) suggested by Maxim Zaslavsky, the USG IT Chief Developer. Obtaining clean data about Princeton students was much harder. Since the purpose of the app is to help with door to door canvassing, having an accurate address for each student is very important. However, approximately 100 or so of the students in the original dataset had addresses that were uninterpretable, such as having their dorm room listed as "Princeton." We chose to manually remove these people from the dataset, as it makes little sense to include them if they wouldn't be reachable anyhow.

For the backend, deployment was definitely the most time-consuming and difficult aspect. When we initially deployed using Google App Engine, it took several hours to correctly configure the app.yaml and settings.py files for Django, manage all of our dependencies, and even figure out basic necessities such as a requirements.txt file for the aforementioned dependencies. Google Cloud Platform documentation contained only highly specific examples which did not adequately provide a general-purpose guide to deployment. The process of deployment on App Engine Flexible Environment also took a substantial amount of time (approximately ten minutes per deployment). Deployment with Django and CAS added an additional layer of complexity, but thankfully Piazza was very helpful in resolving those issues. Luisa contacted OIT early in the process to have our App Engine URL whitelisted, so thankfully there was no delay on that front.

By mid-April, Google Cloud Platform (particularly Google App Engine) was consuming a significant amount of money for reasons that could not be deciphered due to a confusing and inconsistent billing system. By the week of April 17th we had already used $150 of Google credit,

so we decided to switch deployment engines. Sam successfully transferred the deployed web version to Python Anywhere and we deactivated the Google Cloud project. Thankfully, the transfer went very smoothly and no functionality or code was lost in the process. However, it took another week for OIT to whitelist the new URL so that we could use CAS login, although we continued to develop with CAS deactivated during that time.

Overall, Django's models greatly simplified the process of linking the frontend to the backend. We cannot overstate how useful Django's models and forms were to us - they made the entire process of manipulating and collection information much easier than they would be had we had to make custom HTML forms and SQL commands each time. Django turned out to be an excellent choice overall because we all had some experience in Python by the time we actually began the implementation.
One unusual, but ultimately beneficial aspect of our data storage was that we kept our static Princeton student data stored in a single JSON file as opposed to in a database. We felt like this was helpful because that file always had to be accessed all at once, which negates many benefits of databases; keeping it as a JSON file will also be easier for future developers as that is how the data is given originally.

We primarily tested the app locally by both using it ourselves and encouraging other students to try it as new features were added. Rigorously testing the mobile version was of particular interest to us because we expected most users would want to be able to canvass without lugging their laptop from door to door. We tested new features on both web or mobile immediately after they were implemented/deployed so that changes could be made immediately. Feedback from other students was particularly helpful in indicating valuable features, such as the ability to view the progress of your campaign.

SURPRISES
The biggest surprise and mystery of this project relates to the Google Cloud SQL costs. We spent about $150 in 3 weeks for reasons that are unknown to us due to a lack of transparency in Google's billing process. We also struggled a lot with deployment, because the deployment pages were too documented in the sense that they were written primarily for users who already knew exactly what they were doing, and not for users who were still figuring out which features of Google Cloud applied to them. As a result, large majorities of Google's documentation was irrelevant to us or undecipherable until you had checked two other documentation pages to learn the relevant vocabulary, and often both. However, once we got over that initial hump of connecting the SQL database, CAS, static files, and deployment, we managed to keep redeploying successfully as we updated our system.

Django's abstract representation of the database structure through models and its simplified method to connect to a database were both brilliant and incredibly helpful. The entire script for connecting and manipulating the database was greatly facilitated as it was written in python, a language that we already knew. Prior to this project, most of us had little knowledge of

databases or only had the ability to access them through clunky calls via python's MySQLdb library. We were all very excited to learn a better way.

The amount of lost code as a result of our inexperience with github was unpleasant and frustrating at first. Conflicts with merges, failing to recognize when to abide by git's safety features, and general naivete created multiple issues at the beginning. At the top of our list is a time when we accidentally deleted the commit history of our repository after using a "git push origin master --force" command. This was ironically only fixed by applying the same command by another group member who had a better local copy, teaching us a valuable lesson on when and when not to override git's built in safety features, as well as the value of making a temporary local backup copy of the code before attempting to push/pull.

WHAT CHOICES WORKED? WHAT CHOICES DIDN'T?

Our inexperience led us down the Google Cloud SQL path and other Cloud Platform products. While this may be platform of choice for many, the entire process of learning and applying the technology for our project was not enjoyable. Even after achieving the implementation of this tool, the associated costs were undesirable in the long term, resulting in the adoption of Python Anywhere. We would not recommend the Google Cloud arsenal based on the difficulty in implementation and costs of maintenance.

Django was a successful choice for a design framework. As with any unknown technology, the adoption and implementation stages tend to be confusing and restrictive with a very steep learning curve. However, once we familiarized ourselves with the structure, Django became extremely useful at organizing and keeping track of files to deploy. Overall, the documentation and structure that Django provides is very useful in the creation of web-based platform.

Efficiency was optimized by creating a soft line between "frontend" and "backend". Initially, we roughly divided our team into "backend" and "frontend" to create a sense of structure and easily assign tasks. However, depending on the tasks that were available, people congregated to their specific interests/knowledge base. For example, while Luisa mainly worked on the frontend component of QuickCanvass, she handled CAS login because of her work connection with the OIT office. At one point or another, all of us had to complete a task from end-to-end regardless of our preferred side. This gave each of us a fuller learning experience in the course of this project.

Working in the same room worked really well for two main reasons: instant feedback and social pressure. In terms of UI, it facilitated the creation and approval of themes and structure of a page. For the backend, it made it easier to design functions that would match the built UI. Even though we met for only 2 hours a day, being together created a determined environment were progress was optimized. Overall, this method was especially helpful when one person had more knowledge of one aspect of the system then another, or had successfully debugged a similar piece of code. Schedule conflicts seldom occurred as, luckily, we all had similar schedules so we barely worked separately.

FUTURE WORK

Our ambitious plan had to be simplified over time resulting in the following "wouldn't it be nice" list organized by order of importance:

(1) Ability to create multiple campaigns with one account
(2) "Forgot Password"/ "Forgot Username" feature
    (a) As we are currently relying on the assumption that usernames will be the same as netIDs, we did not implement a "forgot username" feature. Extending that, we also did not implement a "forgot password" options. However, if we move away from these assumptions, both features will need to be iAmplemented.
(3) Ability to create more personalized surveys. Currently, campaign managers are limited to up to 3 free response questions, but the ability to add different question types (multiple choice, dropdowns) could be useful in some cases.
(4) Automated Graphing Presentation of statistical results. The data collected can be displayed in a myriad of ways to provide the user an understable overview of the campaign's trajectory over time. Some ideas include a timeline based on the length of the campaign, and more detailed statistics on the number of canvassed students.
(5) Ability to track/view which volunteers have submitted the most surveys and from where

WHAT WOULD YOU DO DIFFERENTLY NEXT TIME?

We would not use Google Cloud Platform again. While we were on track for all of the proposed deadlines (such as the alpha and beta testing deadlines), we did miss the campus USG election season for potential early user testing because we were barely at our alpha test at that point. If we had a second chance, we would aim to have our alpha test finished and released to USG candidates during actual election campaign season. The main reason we could not devote as much time in the first half is because we had weekly assignments for this class and we had other class work to keep up on. That being said, Grace had no experience in Python before this class's assignments and she appreciated the practice. The first 6 weeks were a great levelling experience because they brought the less experienced programmers up to speed before the real work began.

FINAL THOUGHTS

Overall, the most valuable decisions we made ended up being completely unrelated to the code itself. The first was deciding on an initial idea that had an easily implementable MVP, so we could finish the basic functionality relatively early and then spend most of our time refining and adding features as needed. The second key decision was discovering that our time was best spent working together to prevent communication mishaps and accidental overwriting of the codebase. By setting aside time each day to work on the project together in the same room, we were able to work much more efficiently and hold each other accountable throughout the entire project.

Finishing our MVP early made the later stages of the project much easier. We were able to deal with complications that arose (most notably switching deployment services from Google Cloud Platform to Python Anywhere) relatively quickly and painlessly because everyone was on the

same page, understood the code, and was able to reach agreement quickly. Thankfully, we did not become mortal enemies by the end of the project. Instead we were able to maintain healthy and productive working relationships (despite seeing each other every single day for a month straight). No all-nighters were pulled, progress went surprisingly smoothly, and everyone's sanity remained intact up to--and after!--Demo Day and the submission deadline.