MealMatcher Report

Kevin Wang, Drew Wallace, Andreas Dias, Valerie Morin, Andrew Charette

Save for Andreas, whom had some prior projects with Ruby on Rails, none of us had experience with any sort of web application development before beginning this project. At our first group meeting and in our initial design document, we discussed this concern, but we were confident that we would be able to familiarize ourselves with the appropriate frameworks and learn the necessary languages well enough to be able to follow through with our idea. Despite finding the learning curve about as steep as we had expected, we were able to get a basic web page with databases up and running quickly (see *Milestones* for specifics). We all took on the challenge successfully and are now well acquainted with how Django and Bootstrap can be used for web development. While our main concern when beginning our project was our lack of experience, this proved not to be a hindrance but rather a positive contribution to our experience by giving us the opportunity to gain a lot of skills and knowledge.

Milestones

We were able to achieve all but one of our milestones on schedule: rather than launching the site advertising it to the entire student body, we tested it using only a select group of friends. The impacts of this choice are discussed in other sections below. For comparison, below is our original planned roadmap followed by the dates on which we actually achieved the milestones.

Planned Milestones:
Week of March 23: All group members set up for development and basically comfortable with tech
Week of April 6: Minimal version of site online and usable. This will include a website and database that allows the user to sign up for meals, receive email notifications, and view what meals they have signed up for. Begin advertising and iterate based on initial feedback
Week of April 13: Support for editing posted meals, feedback features
Week of April 20: Some form of messaging for coordinating meeting at the time of the meal
Week of April 20: Initial documentation and write-ups
April 24: Alpha test
Week of April 27: Support for open meals (more than 2 people), talk and demo preparation

May 1: Beta test
Week of May 4: Demo
April 12: Project due

Actual Milestones:
March 27: First commit, Git setup and Django installed
April 1: Basic models set up
April 2: Basic front-end, admin created, back-end for homepage, CAS login
April 3: Back-end for find meals and view meals pages functional
April 4: Functional navigation bar locally
April 5: First dabbling with Heroku
April 6: Back-end and front-end successfully integrated. Matching functionality successfully tested
April 8: Post office set up, deleting meals implemented for view meals
April 12: Notifications to validate find meal forms, emails introduced
April 15: Open meals back-end created and basic calendar functional (front-end)
April 18: Email notifications up and appear to be working
April 19: Introduced brunch option on weekend (back-end and front-end changes)
April 22: Heroku server site working, front-end visual upgrades
April 23: Edit attire added and working (back-end and front-end)
April 24: Alpha test
April 25: Major front-end visual upgrade
April 27: Bug fixes, code clean up, about page made
April 29: Email templates to avoid Proofpoint, minor front-end updates
May 1: Emails no longer blocked by Proofpoint. Beta test.
May 2-5: Bug hunt and fixing, about page integrated with home page
May 6: Demo
May 7: Started write-up, realized asynchronous emails were not working
May 8-11: Bug hunts, clean up front-end code and back-end code
May 11: Asynchronous emails fully working
May 12: Project due

Design and Testing

We more or less adhered to the design framework laid out in our design document. The only major differences were the tutorial and FAQ section, the open meals calendar, and organization of the table containing meals the user has signed up for.

The addition of a tutorial and an FAQ section on our homepage improves the usability of the site by making the available options clearer to users and explaining how to use the site. We have found this especially important to clarify details about the process of finding your match and to alert users that the email notifications may be caught in the spam filter (see *Surprises* for more). The addition of a calendar for viewing and finding open meals proposed by others makes it easier to find an open meal on the day that you are looking for. For users, it is a simple way to directly match a meal rather than enter info and hope someone else will enter the same info.

In addition to the table of matched and unmatched meals that we anticipated having, we also created a separate table for expired meals and a separate table for currently occurring meals. Having a separate table for expired meals makes the integration of a feedback form instead of an edit button natural and aesthetically pleasing, and having a separate table for any meal happening within the hour highlights the important information for ongoing meals that was previously buried in a longer list.

There was one major choice that we made in the original design and kept in the final version that we are still unsure about: the method of finding your match in person. While knowing the clothing your match will be wearing and having a specific meeting place (the card swipe desk) is helpful, it may not be enough, especially since the clothing is self-reported. We did not make any of the students testing the app actually show up for their meals, and since this is only something that can be judged through use, we are unclear on this current system's effectiveness. Our demo gave us a lot of feedback and suggestions about this feature, and while the code for it fully works, we intend to have further discussions to improve or even replace it before launching the site school-wide in the fall (see *Improvements*).

Finally, there were numerous small design decisions that were either unanticipated in the original design doc or too small to include there. One such decision was that of when exactly to send the notification emails. We have found that sending an email when a user is matched, when a user is unmatched, when an attire is changed, and an hour before as a reminder as most effective. Another question we only addressed later in development was that of the feedback form. We decided to manually review feedback given by users in order to more properly judge the legitimacy of the users' complaints. In this way, we can use our discretion to decide whether or not to temporarily disable a user. As of now, we plan to disable a user for two weeks if he or

she does not show up for 2 meals within 2 weeks, which is reported by their matches for the meals they missed. This required administrative access to disable/enable of users on the site (see *Surprises*).

For testing, we relied on both self-testing as users performing typical actions and having friends play around with the test without instruction to "try and break it." We found both methods to be very effective, especially having friends test the live website, as having a standard testing script would not work outside of testing model functions. Testing the email notifications was especially tricky, due to the blocking by Proofpoint but also having to wait in order to see if we get the notification emails.

Surprises: What Could Have Gone Better

The two biggest issues we encountered were Proofpoint blocking our automated emails and difficulties setting up a development environment on one of our computers due to specific local settings. Each of these surprises took a good week or more for one of us to sort out. The Proofpoint email blocking was eventually resolved using HTML email templates to render the messages, and involves small hacks such as including the words "subscription preferences" and "unsubscribe" to help pass the Proofpoint filter. Emails sometimes get sent to the spam folder, due to a Gmail filter we could not always bypass. The development environment difficulties were a result of poor standardization in the environments, especially with the version of Python used. *internals.pdf* now fully lists the environment requirements, though using a common virtual environment for development is a more elegant solution for future projects.

There were many hidden bugs in the code that were eventually resolved, such as the ordering of the users in how Django handled relationships. Because we needed to keep track of the attires of the users, it was important to know which UserProfile was added to the meal first

(i.e. is user1 and is associated with attire1). We initially assumed Django stored these relationships as a queue, but found that attires would be switching randomly. We concluded that we could make no assumption on how Django ordered the UserProfiles and had to retroactively add specific user1 and user2 text fields to the models to manually keep track of who had attire1 and who had attire2.

We also ran into issues with the Django admin site, which we intended to use to manually disable users. The admin site worked fine on everyone's local website, but after we pushed to Heroku we found the admin login always resulted in the infamous "Programming Error" from Django's own library code. Our back-end team (Kevin and Andreas) spent many hours trying to get the Django admin working, and while we concluded there must be some difference in the Heroku database migrations vs. local database migrations that was causing the issue, we were unable to get the Heroku admin site working. As a result, we were forced to a "hack" of directly manipulating the databases by connecting with PG Commander in order to manually disable and re-enable users. The details of the process are in *index.html*.

Improvements: If We Had More Time

Our main goal for the future would be to launch our application the student body. Our application was not functional or polished early enough for us to launch it as planned in our design-doc's original milestones, though that would have provided us with some invaluable feedback on usability. We were still able to get some friends to be test users, but a larger sample size would have provided more feedback and found bugs faster.

Specific back-end improvements we would have liked to implement are an improved matchmaking system, attire input lexical analysis, and a site-based feedback form. Our current matchmaking system just checks if there is an open meal at the specific time and location to join,

and if there is, to join it. It is very possible for users to match with the same person repeatedly, especially if there is a small user pool. We would like to store more data on whether users have eaten together, and if so to not match the users together unless absolutely necessary. This would involve parsing older meals in the database to see if two users have eaten before. We would also like to perform lexical analysis of the input for the attire box if we decide to keep it for the launch, since right now the user can input anything 100 characters long. Parsing the inputs to look for clothing related input would restrict misuse of that form. The Google forms based feedback is convenient, but allows users to input anything as their identification. We would like to implement a meal rating feedback form that rates meals directly in our database. This might include automatically flagging when a user reports their match missed the meal, so that we can move to an automated system for disabling users.

Other possible areas for future work include matching for other activities outside of just meals and building native iOS/Android applications to provide GPS-based notifications when a user's match arrives at the dining hall. The site currently works fine on mobile devices thanks to Bootstrap, but the native applications would allow for phone vibration notifications when the two users are close to each other, a possible alternative to the attires for finding your match.

A known security issue we didn't have time to fix is we currently have the Django debug on due to issues with static file hosting when debug is off, leading to Django debug pages if the user gets the site to crash. We added all sorts of checks to prevent crashes from being possible.

Looking Back: What We Would Have Done Differently

If we were to do the project again with the skills we have now, the biggest difference would be that we would work much more quickly and efficiently. Since the beginning of the project, we have gone from practically zero knowledge of web development to a level of

competency. Beginning with this level of competency, our code would be better organized and cleaner from the start, which would speed up development while creating a much more sustainable codebase, and also decrease the number of refactors and hopefully avoid retroactively changing models. This would also lead to a faster and more reliable final product.

Perhaps the only major design decision we would have handled very differently would have been to attempt to establish a better way of finding the person you are matched with. At this point, the system is designed around the assumption of matching based on attire, and changes to that assumption would likely require a major re-haul of our system, specifically with the models and forms. This would involve changing the databases, a dangerous move that requires major changes throughout the code. It is another example of what would be different if we were to rewrite the project with our new knowledge: we would have the familiarity with the model-view-controller architecture and could better define our models from the get-go in order to avoid dangerous drastic changes at later points. We strongly feel spending more time defining the models at the beginning will save lots of time later on.

Logistically, development went very smoothly, so we would likely keep that style similar. It took about a week to get going, but after we broke into front-end (Drew, Valerie, Andrew) and back-end roles (Kevin and Andreas), our project manager Kevin was able to systematically assign work at the start of each week with goals and checkpoints to be reached for each team. We met in-person Sundays and Wednesdays, with partial group meetings as needed and individual work ongoing. Our group was split fairly evenly with two members working on backend and three on frontend, and neither group ever got so far ahead that it had to wait for the other to finish a task. Both subgroups were thus able to continually integrate and use each other's progress.

Advice for Next Year's Class

The most important piece of advice we could give to next year's class is to manage scope well. Most students know of the 2X time rule (budget twice as much time as you think you will need), but in our experience few truly keep it in mind. While we did not entirely follow the rule ourselves, we did choose a project that we expected we could finish in 80% of the time scheduled in our milestones without any all-night programming sessions. This not only led to a much healthier development cycle, which would translate to a sustainable one on a larger project, but also improved the quality of the final project. Every project takes longer than expected, so it is important to plan accordingly. Another piece of advice we have is to learn Django over Spring Break if you are going to use it for web development and have not used it before. There are lots of good tutorials online, such as How to Tango With Django (http://www.tangowithdjango.com/). Also, spend a good period of time mapping out the models for the databases. It will save time when you try to retroactively add features to the models.

Conclusion

Thank you Professor Kernighan for a wonderful semester, and thank you Tom Wu for helping us through our project!

"I lol'd so hard after seeing Professor Kernighan as an extra on the Princeton Snapchat story." - Deborah Alaine Sandoval, who posted this picture on Kevin's Facebook timeline