Princeton Food Alert

Project Report

Introduction

Princeton Food Alert started with one student's love for enchiladas. An idea grew into a framework which set up a web application which has approximately 150 users at the time of publication. Creating Princeton Food Alert has taught us a lot as programmers and developers and we are pleased to have built an application that is useful to Princeton's campus.

The Idea

In order to decide on an idea for our project, we focused on identifying existing problems and proposing realistic solutions. Drawing upon our diverse experiences in different arenas of campus life, we evaluated a number of different options for feasibility, user base, and potential impact. While all the members of our group are computer science majors, we brought various life experiences to the table: athlete, A.B. major, B.S.E. major, international student, different residential colleges, various student organizations, and eating club affiliations. Two group members are independent, which means they have no meal plan or eating club membership, and they had several food-related ideas. One group member frequently commented on his love for enchiladas and how he wished he could know when they would be in the dining halls. Thusly, we finally settled on the idea of food notifications, which brings convenience to Princeton solution and provides a solution to a genuine problem.

Project Planning

We put a lot of initial thought into creating our project checkpoints. This foresight allowed us to create realistic goals and achieve significant progress from week to week. In addition to meeting the checkpoints we laid out in our design document, we were also able to incorporate some

additional features that originally were not planned for. We would recommend to future COS 333 groups to dedicate substantial time to set up checkpoints in the beginning of the project in order to make the process smoother.

During our demo, upon being asked what we would do differently if we could go back and start all over, we agreed that we would start testing earlier. We would modify our goals so that we would have at least 20-30 users by the end of week four. Had we done this, we would have been able to get more user feedback and adjust our goals accordingly to prioritize features requested by users.

Design Decisions

One of the difficult initial challenges in creating this app was deciding between different frameworks. Flask and Django were both reasonable alternatives for a web framework, while Heroku and AWS were both reasonable options for a hosting platform. Weeks could have been spent deliberating the various merits and flaws of the different options, but eventually we made an executive decision and moved forward. Ultimately, it was the simplicity of Flask and Heroku that were appealing, especially given the modest computational and storage demands of our app. Similarly, we also decided on using MongoDB for our backend database, as opposed to alternatives such as Postgres and MySQL. Making this decision relatively early in the process helped our database team leader become knowledgeable about MongoDB and was highly beneficial later when we needed to make changes to our database structure. Once we decided on MongoDB and set up a database, the team members working on the Flask app and the email composer were able to make calls to the database to access user information and update preferences. During this phase, we learned to seek advice from more experienced programmers, such as our TA, Ghassen. We would recommend for future groups to ask for help from the course

staff when making design decisions about frameworks they have never used before in order to prevent incompatibility and to increase efficiency.

Organization and Communication

Our team consisted of members with various programming backgrounds and expertise. Some of us had experience with building a scraper for a website, while others had experience working with databases. We divided up the work so that we were all working on areas that were familiar to us in order to maximize efficiency. None of us had any real experience with web design so that became more of a group effort spearheaded by a couple of members, but ultimately all design decisions were made as a group. We found that this enabled the right balance of specialization and group collaboration.

Our group used GroupMe to stay in contact, update one another on issues and changes, and to schedule group meetings. Being able to instantly share updates and screenshots made GroupMe far more useful than email, as did the GroupMe functionality of liking a message to express agreement. While working on a group project, there are inevitably moments of miscommunication. For example, two team members did not clearly communicate the default state of the database when a user does not add a food entry - whether this would result in an empty string in the “foodpref” field, or an absence of the field. This caused an error that took some time to debug. However, for the most part, our team managed to effectively communicate important information to each other in a timely fashion.

In a similar way, we used GitHub to share code and project data. This allowed us to share files like HTML templates, Python scripts, scraper output from the dining services website, and other important information. In this project, we had one team member serve as the Heroku expert who handled all deployments to the Heroku server using Heroku’s built in Git functionality. In the future, we could definitely look into authorizing all team members to deploy to Heroku

individually, to eliminate the lag time between someone making a change and getting the changes live. In order to do this, we would need to become more proficient in Git so that we could manage changes properly, especially if we are making changes simultaneously.

Importance of Modularity

Aside from the division of labor, we also wrote a lot of our code to work with different moving parts. For example, our scraper/emailer section was broken into four parts (scraper, matcher, composer, sender) for the sake of modularity. This allows us to run parts independently. For example, we can run the first three parts to see what emails the system would write, but not actually send them (so we do not spam our users). Or we can run just the scraper, and use that output to build onto our autocomplete feature. The modularity also makes it easy to change our code. For example, if we were to change the database where our user's food preferences are stored we would only need to change the matcher, and leave the other parts untouched.

Testing and User Feedback

Testing was vital to helping us locate bugs in our code and gaining user feedback. Through testing we learned the types of entries users would add to our database. We realized we had to make our matching script case insensitive and we decided to base the autocomplete data on previous dining hall menus in order to reduce spelling mistakes and increase the possibility of matches for users. From testing and feedback from various mobile devices, we learned that the native autocomplete in many browsers is too slow and aesthetically unpleasing, which led us to a third-party autocomplete implementation called Awesomplete. From TA feedback, we learned that our original email format was not very readable so we changed our emails to a table. As mentioned earlier, we would have wanted to start testing earlier in order to obtain more user

feedback. Moving forward, we intend to base most updates on feedback from our growing base of users.

Handling Live Applications

One of the most rewarding but also nerve-wracking aspects of working on this project was realizing that we have real users. With approximately 150 users, our website could be getting hits every hour of the day. Therefore, it is even more vital to deploy clean versions of our program and to employ defensive programming in order to reduce potential crashes. This project trained us to work calmly under pressure and to test our code thoroughly before deploying. Once our site was live, we couldn't bring it down again, so we had to be careful when adding major changes. When we defined Heroku Scheduler tasks, for example, it took a while to get the scheduled scripts to actually work, and in the meantime we had to manually run the scheduled scripts at 12:30am every day.

Future Work

One feature we could implement in the future is to display the nutritional information provided on the dining services website when hovering over your preferences. If we were able to implement that, the next step would be to add a feature that allows a users to filter foods based on their nutritional information. For example, if a user wants to receive notifications for 'Chicken' but doesn't want to receive any notifications for chicken with high levels of sodium because of dietary restrictions, that person would be able set a limit for the amount of sodium in a single serving.

Another feature we are interested in exploring is giving our users the option to sign up to receive SMS notifications. One reason we weren't too keen on SMS notifications was because we didn't want our users to feel like we were spamming them. At the same time, we feel the option

could be useful in the future for someone whose inbox is swamped by emails or someone who doesn't own a smartphone.

Moving forward we intend to reach out to USG regarding the feasibility of adding our web application to TigerApps.

Takeaways

Regarding group dynamics and teamwork, our biggest takeaways were accountability and patience. We all learned to treat our commitment to the project seriously and make the effort to meet in person as a team several times per week.

Overall, this project was immensely helpful to all of us in gaining experience as programmers. In our academic work, we often focus on getting assignments turned in and mastering abstract concepts. However, this project required a different approach to problem-solving. There wasn't always a right answer, and we needed a new level of self-direction and accountability in order to make progress. Learning the hands-on, practical realities of getting an app off the ground and actually available to users was a boost of confidence and a new way of thinking about programming.

Acknowledgements

We would like to thank our TA, Ghassen Jerfel, for his guidance throughout this project. We also thank Professor Kernighan for his advice on our project topic and his engaging lectures. Lastly, we appreciate the past work of COS 333 teams who served as inspiration and motivation for our project.