Discovering

Description

Discovering is a web-based platform that allows organizations to share their content to public, let people explore new worlds and get connected to others with common interests.

The platform provides specific features to support the goal:

- Organization profile
  for displaying and promoting contents
- Content pushing
  according to user interests
- Search
  for user to access certain content
- Share
  share interesting content with fellows
- Follow
  keep user updated with information that they marked as interested
- Private message
  get connected with others
- Relation management
  easily get organized
- Timeline
  get to know how far you have go and feeling achieved

Inspiration of the project is that nowadays in internet era, information explodes while valuable contents are often buried under massive irrelevant data. We are aiming at building a platform for people to efficiently get exposed to information that can be truly valuable for them and help information resources to spread their content which will be mutual beneficial.

This platform could have multiple user scenarios,
e.g. Museum create a quiz and put a QR code of the link at the door, kids can play with it to get familiar with theme of the museum and be rewarded with prizes if performs well.

Development Methodology

For this project, we adopted the extreme programming process (XP).This is a current industry standard for several a reasons: The process allowed us to better-design our software, frequently update each other, have small and attainable, realistic goals, and keep on-track for each two-week iteration of tasks that need to be completed. Basically, it is a software development process that helps keep imperfect, human developers efficiently organized in a way that is industry-recognized as the norm.

At the beginning of every two weeks which is the time length for an iteration, we would hold a whole team meeting.During the entire-team meeting at the beginning of each iteration, we would discuss logistics, such as our progress as a team, where we stood according to the project timeline we made, what (user stories) we needed to implement for that week, and whom each team member would pair up with. In the first week, each team member gather with their sub-group to break their user story up into more measurable tasks, each allotted an expected portion of the user story points. Since we had three sub-groups in total, we would divide the total user stories into three parts. Overall, Hao and Kyo's group worked on the part related to search engine, message notification system and following unfollowing staff. Jin and Guoqiao's group worked mostly on the quiz information and ranking system. Chris and Riyad's group mostly focus on the timeline. After the division of the workload to each sub-group, we would usually work as a sub-group until the group meeting before the iteration meeting. The most common problem during our iteration would be in the testing stage. For example during this iteration we only implemented half of the functionality which allow the user to follow other user through the following button, but we needed the message which notify the user of new followers to write the test for testing the effects of click the following button though that was scheduled to be implemented a following iteration. In such a case, the test would be limited to only checking the database for the followers of a specific user, so we were unable to test the effects of a message notification for every user, and thus unable to test weird potential edge cases pertaining to specific users. Also for the most part, I pair programming with Guoqiao most of the time. The idea behind splitting into pairs is that one person, known as the "driver", was able to sit behind the keyboard and the other person, known as the "observer", was able to help out with bigger ideas making meaningful and insightful suggestions (whether catching an error or mistype to sitting back and thinking of the main purpose at-hand and if the solution being pursued was being done in the best way feasible). The pairs would switch roles from time to time, depending on productivity or preference. Usually one of us sit in front of the laptop and another one come up with the new ideas for the user story such as how to display the information about one specific quiz. Which kind of information do we need to include. When we meet a problem during the coding development, one of us usually check the requirement on the wiki page and search for the possible solutions. Also we usually prioritize the user story we need to implement by the importance to our user. For example, the notification new follower system is less important than the quiz information since the primary goal of our system is to provide the quiz to the users and organizations.

Collaborating problem comes in our collaborative development. Since we all work on own branch during the iteration, we usually have conflict during the merging stage. Usually I and hao come to solve the conflict together as a small group. For the testing staff, we adopt the normal python test and also parameterized test. Usually we have two to three unit tests for each function when we pass two different parameters to the function we wrote. In addition we use the parameterized tests to make sure that we could run several test function through a single call to the testing class. After the testing and merging, during the iteration meeting we usually present our code and user interfaces to TA by each sub-group and then describe about our test cases. After our presentation, TA will give us some suggestions and advice to improve on the later iteration.

Requirements & Specifications

The user stories for our project are fairly straightforward. Throughout the entire project this semester, we split our group into three teams, each one working on the same set of user stories (either of S01, S02, or S03) for every iteration. More specifically, each team created new user stories related to their previous user stories, either creating a user story which adds more features to their previous user story, or creating whole new user stories – but teams generally didn't intertwine their user stories. Sometimes tasks from one user story would possibly overlap with features from more than one user story (i.e.: when S03 refactored database-related or cookie-related things that all groups/user-stories would end up depending on in some way – and this was easy to do since we were using git for SCM).

Each team worked both in Python using the Flask framework, and also on the front-end using HTML and JavaScript, so each person on the team has a general idea of how to do each others' parts from personal coding experience, if not alone from our weekly code-presentations. Each team also created unit tests, either in Python (including parameterized testing) or in JavaScript (using Jasmine.js run via Casper.js).

- S01: created a login, search, and notification system.
- S02: created a homepage, an organization page, and user-specific dashboard with a recommendation system and commenting.
- S03: created a timeline page with a Google Chart, Google Maps, and other user-specific data from the database.

We will now elaborate more specifically on the use cases associated with each user story. S01 was more responsible for the general functioning of the web app. All their features would be used across multiple pages. S02 and S03 both focused more on creating interesting pages and features within those pages for the user to interact with.

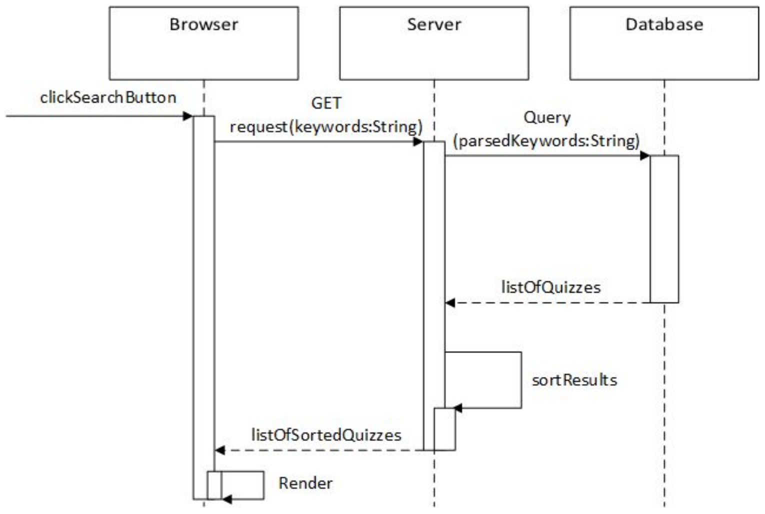
Architecture & Design

Top-Down Manner of Architecture

In general, all the code in our system is run from our Python server via the Flask framework for routing the URLs passed in from the front-end address-bar on the browser by a user. Once the endpoint is processed in our Python server, we use the Parse API to retrieve relevant data from the Parse database (stored on a different server) and then parse that data in our Python server, sending the formatted data to the front-end to be further parsed to fit into our html front-end using Flask's built-in Jinja2 html templating engine. Once we parse the data using Jinja2 at the html/JavaScript-level, we either display the data directly in html, or we process it through JavaScript, and finally use the output of the JavaScript code to populate the html for our webpage.

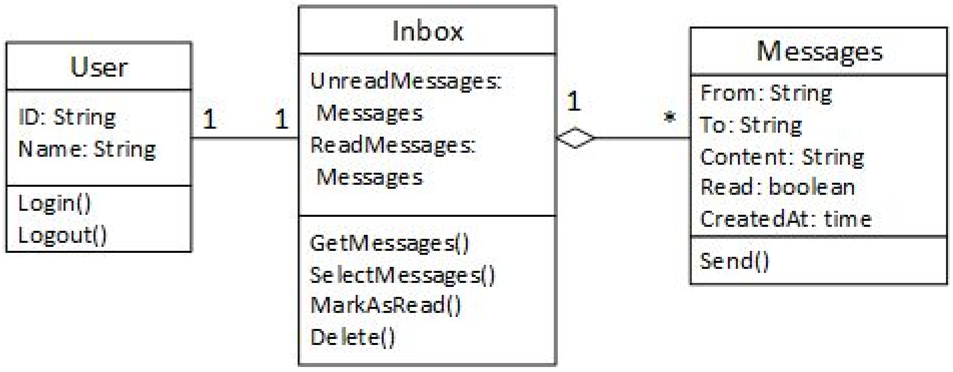
Searching

Discovering provides a search engine where a user can search for other users and quizzes. Given a query input, the search engine matches the substring of the input to the name and the description of the quizzes. Then the user can filter and sort by each attribute.
When the user clicks search, the input string is sent to the server via POST AJAX request. The server responds to the request by sending a list of users and quizzes in JSON. By default, the quizzes are sorted from newest to oldest. When the user sends a filter or sort request by POST AJAX request, the server looks at the cached listed of quizzes previously sent to the user and filters/sorts the quizzes and returns a list of filtered/sorted quizzes. The client takes those quizzes and parses it into correct HTML format.
Diagram of Searching:



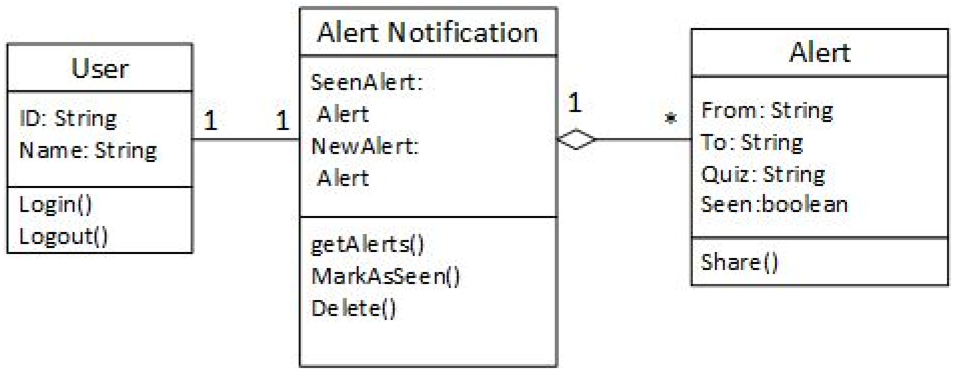
Message

A user can send a message, mark a message as read, and delete a message. Send is implemented such that when a user fills out the message form and submits it, the client sends an POST ajax request. The back end server handles the request by storing the form in the Parse database. When a user logs in, the client will send a POST request querying the messages. Mark as read and delete are handled by first checking the HTML input boxes for the selected messages. Then, the client sends the list of the selected message ids in JSON, which are embedded in the html, to the server. If the server receives a mark as read request, it marks the read attribute of the message to true. If the serve receives a delete request, it removes the messages from the database.
Because Parse treats data like a relational database tuple, there is a single inbox table in the database that handles messages for each user. If we were to use another database such as MongoDB, the inbox would have been an attribute of the user.
Diagram of Messaging:



Alert

When a user (user A) shares quiz with another user (user B), the other user (user B) gets an alert notification stating that the user A shared a quiz. Implementation of alert system is similar to messaging system. Alert system is separate from messaging system because users should not be desensitized from messaging system since a user could get a lot of share alerts if the user is following another user.
Diagram of Alerting:



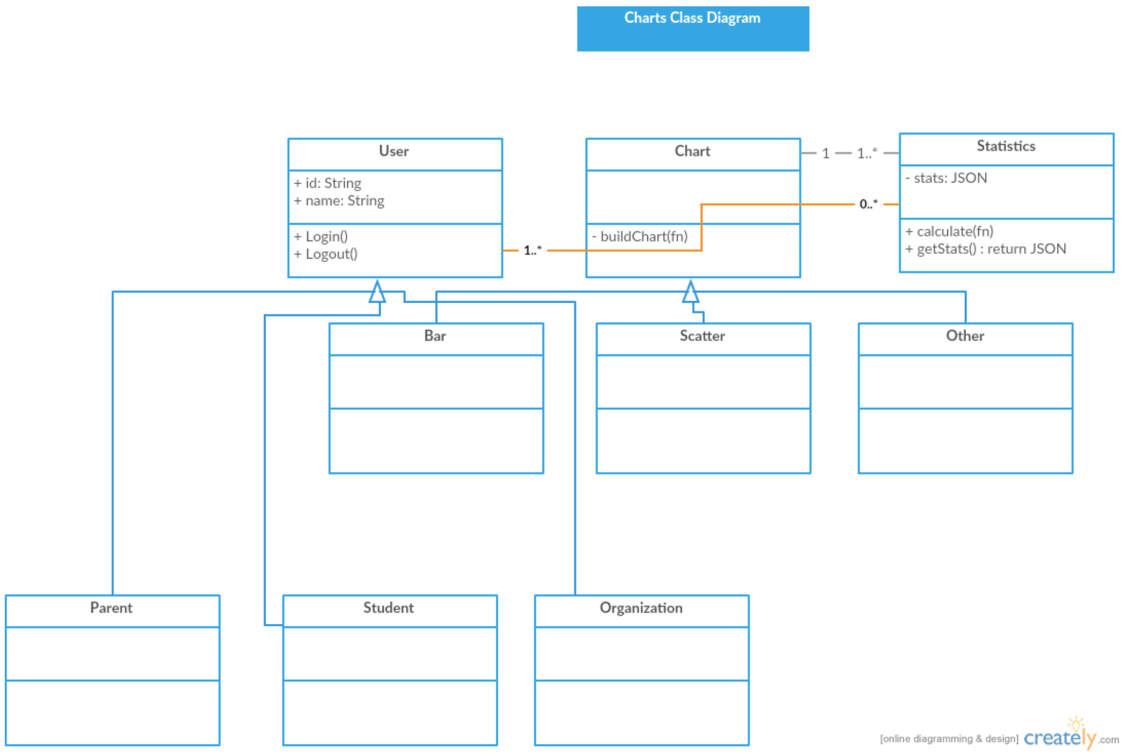
Timeline

Let it be noted that we would test the functionality of this page on a user which had data associated with is account (username: dev10, password: dev10). Since many users were dummy users created for testing, doesn't display accurate data when logged in as those users.

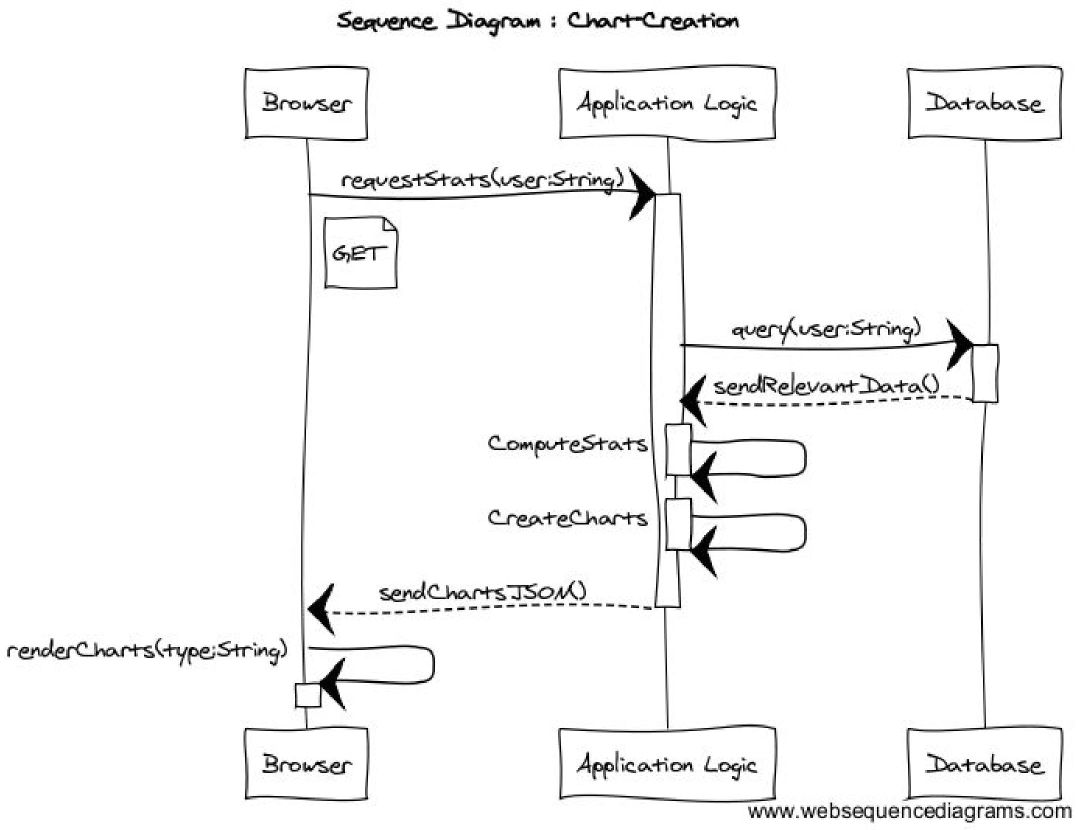
To view the Timeline page, a user must simply go to {our webpage}/timeline. Once our Python/Flask server receives this request, a sequence of Python methods are called (initiated in "def timeline():", which is a method that acts as a Flask router for the "/timeline" endpoint). The required data (once fetched and processed from the Parse API) is then sent to the timeline.html page via the Flask framework. From timeline.html, we extract the data using Jinja2 (the html templating engine that comes with Flask). In the html, we use Jinja2 loops to extract relevant data which either needs to be processed using either inline JavaScript (for Google Maps-related html), or which sends data to the charts.js module to create the Google Charts chart. To generate the retrieved Achievements data, we simply used pure html generated on the front-end solely using the Jinja2 engine.

Let it be noted that tests directly related to the Timeline user story are stored in test_timeline.py, test_chart.py, and SpecRunner.html.

This Class Diagram (below) shows the general logic for creating charts for our Timeline. We used a bar graph in our Timeline.



This Sequence Diagram (below) shows the basic logic for creating a chart in our Timeline.



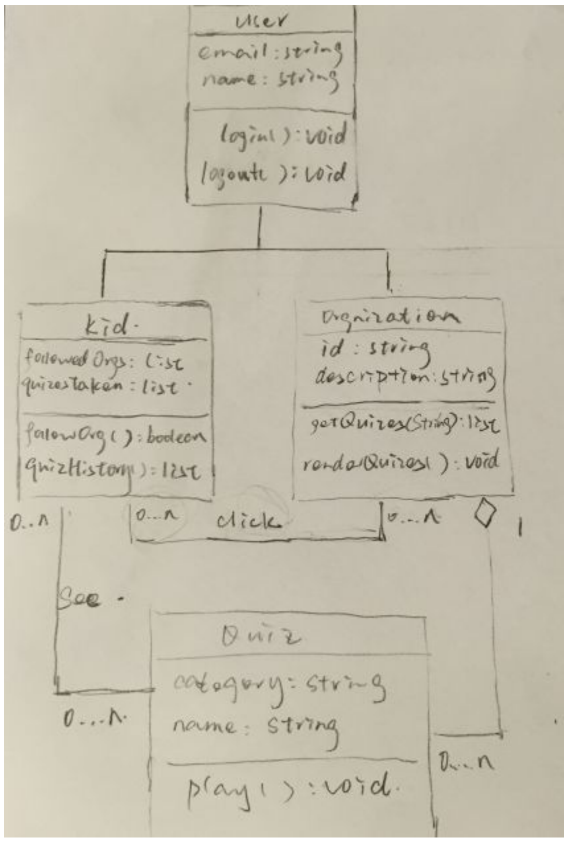
Explore Organizations

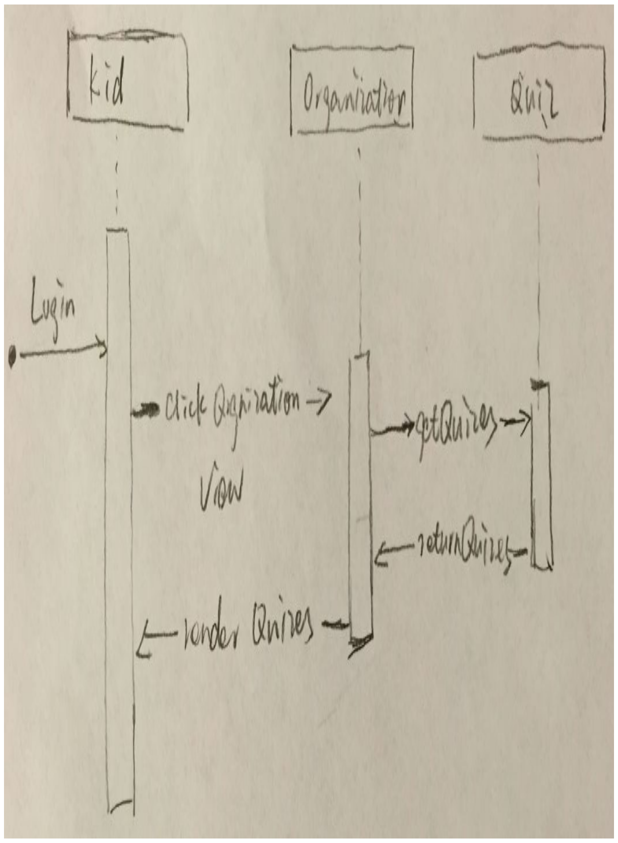
Let it be noted that we would test the functionality of this page on a user (username: dev10, password: dev10) and organization (dtl) which had data associated with the account. Since many users were dummy users created for testing, doesn't display accurate data when logged in as those users.

To view the all the organizations, a user should choose "Explore" tag under the icon button. Once our Python/Flask server receives this request, a sequence of Python methods are called. The required data (once fetched and processed from the Parse API) is then sent to the organization.html page via the Flask framework. In the html, we use loops to extract and display relevant data. And then to view the quizzes, you will need to click a specific organization from all the listed organizations. Once our Python/Flask server receives this request, we will render dashboard.html. This page will show the details of the organizations and its quizzes.

Let it be noted that tests directly related to the Organization and quizzes user story are stored in test_organizations.py.

This Class Diagram (below) and UML shows the general logic for creating charts for organizations.





Reflections and Lessons Learned

Hao

The final produced program meet our initial purpose with all main functionality and minor features to be improved. During the development I found that proficiency in version control system greatly helps team collaboration which should be a skill every programmer must have. Agile development has its unique advantage especially today as most software are complicated thus subject to change during development. Even though, make right decision of which libraries/services to make use of to maintain the project stable in a relative long term in still fairly important.

Riyad

The toughest and most time-consuming part of this project was learning how to properly use different technologies that I hadn't really used before. I didn't know what endpoints were before, hadn't used Flask before, had never used a database API with a database residing on another server, and had never used an html-templating engine. Also, there were some difficulties of figuring out how to efficiently query certain columns across multiple tables, due to Parse's kind of weird database querying interface. This was made especially weird given we were using an unofficial API for Parse ported as a Python module. Due to these difficulties, we ended up querying more data than needed from the Parse server to our Python server, then filtering out results (e.g. when a join needed to be done, for instance). This ended up being quite slow, especially since one table contained encodings of entire images in a column, so that was a big limiting factor/bottleneck on the throughput of our database querying, which definitely slowed down the reaction time on the front-end, when the data was finally parsed using Jinja2 (Flask's built-in html templating engine).... And don't even get me started on writing test parameterized test cases in Python – that's a nightmare due to Python's unintuitive structure of parameterizing test methods in the unittest module. Simply figuring out the extended and unintuitive structure/syntax of that specific type of test case took about a user point or more (the first time, and dealing with refactoring or revising breaking test cases with such a messy parameterized unit testing framework only added to the code-complexity and debug time).... The easiest test-cases to write were the trivial, white-box, front-end Jasmine scripts executed with Casper, though (I feel those were written more for regression-testing purposes – trivial to write and seemingly unimportant in the short run, yet useful in the long run, should basic dependencies we assume to always work be changed).

Kyo

Through this project, I learned the importance of division of work. Each group member has their forte: some people are good with the front-end while some others are good with the back-end and some are good at creating tests. Generally, it is better for a person to work on something that they are good at since it will increase the development efficiency. If a group member to work on something that he/she is not familiar with, there is an overhead of learning the new materials which would in return slow down the overall development process. However, it is important that each group member is familiar with other member's work so that integrating the work all together will not become a problem.

Chris

The most valuable thing I learned during this project was that seemingly small changes in the development plan can have a large effect on the rest of the course of development. Originally, we planned on randomizing the pairings every iteration. We did this for one iteration, but once people found the group/partner they were most comfortable and productive working with, we decided not to stick strictly to the original plan, and allow the groups to remain for the rest of the project. This dynamic and flexible decision making process was very important to the success of our project. Good leadership is very important as well. Comparing this semester's Software Engineering II where we had a group leader to Software Engineering I where that role was more ambiguous, I found this semester's development much more flexible and less stressful.

Jing

One of the most important lesson I learned from this course is that to divide the time frame properly. There are several times that we spend too much time on learn and debug something that is not very important and thus we don't have enough time to implement all the functions we used to plan to finish. Another important thing I learned from this course is to always communicate with your partners. There was a time that my partner and I implement a function twice in different branch. Also it's much easier to figure out bugs when you are doing pair programming.

Haotian

After working on this project, I find the ability to learn new technology and master them in a short period of time becomes a critical skill for everyone want to work as a software engineering. Since we usually have a lot of new things like language, framework, testing to learn at the start of each project, so we usually need some time to get familiar with the project. Usually the faster we could get familiar with the project, the earlier we could work on it. During the future project, I will keep improving my skills on adopting new knowledge.

Guoqiao

From this project, I learned the two following things. (1) First thing I learned is flask and parse. (2) I'm more familiar with doing team project using github, especially for coding in my own branch and the merge into master. (3) Doing product is not like doing research, in most cases, what we have to do is to make some products but not figure out how it works. What's more, we should always start at the fundamental things and extend features in the future.