Assignment T1: Preliminary Project Proposal

- TeamShiba
  - Jianing Li (jl5543)
  - Yifei Wang (yw3229)
  - Eurey Noguchi (yn2377)
  - Chiqu Li (cl3895)

Part 1

Write a few paragraphs that provide an overview of the software engineering project that your team would like to do and answers all five of the numbered questions. 1. What will your project do? 2. Who or what will be its users? Your project must impose registration, authenticated login and timeout/explicit logout, so your answer should say something about this. 3. Your project must be demoable, but does not need a GUI if there's a command line console or some other way to demonstrate. (All demos must be entirely online, there will be no in-person demos.) What do you think you'll be able to show in your demo? 4. Your project must store and retrieve some application data persistently (e.g., using a database or key-value store, not just a file), such that when your application terminates and starts up again some arbitrary time later, the data is still there and used for later processing. What kind of data do you plan to store? 5. Your project must leverage some publicly available API beyond those that "come with" the platform; it is acceptable to use an API for external data retrieval instead of to call a library or service. The API does not need to be a REST API. There are many public APIs linked at https://github.com/public-apis/public-apis (Links to an external site.) and https://github.com/n0shake/Public-APIs (Links to an external site.). What API do you plan to use and what will you use it for?

Our project, ShibaMatch, makes selecting restaurants for a couple or group of friends and family a breeze. Sometimes a group cannot decide upon a restaurant to go to, which causes unnecessary time to ping-pong ideas of where to go to. Additionally, we always have to search through the web to check for good restaurants that suit the group's preferences, which takes more time. This is where ShibaMatch comes in. On ShibaMatch each individual user in the group can independently swipe left or right on restaurant cards to indicate if they like that particular place. When all group members of the group 'like' (swipe right) a certain restaurant, then it is a match. The group can easily decide on a restaurant. If they were unable to match, then we will be able to display the most liked restaurants to go to.

For our users, as mentioned above, we will be targeting a group of people where they each hold an individual account. Therefore, each user will need to initially register and login to use the app on their account to specify which restaurant they like. Users will also be able to explicitly logout of their accounts.

In our demo, we believe we will be able to show a web app of ShibaMatch. Specifically, multiple individual users will be able to swipe left or right asynchronously and display if there was a match or the most liked restaurant if there were no match. For this to happen, we will need to store data (some as cache) of the users, groups of users, restaurants, and likes. Additionally, for our public API, we will be utilizing Yelp data API to retrieve information about the restaurants.

Part 2

Write three to five user stories for your proposed application, constituting a Minimal Viable Product (MVP); registration/login/logout should not be included among these user stories, nor should 'help' or other generic functionality. That is, your application should do at least three application-specific things. Use the format

< label >: As a < type of user >, I want < some goal > so that < some reason >.

My conditions of satisfaction are < list of common cases and special cases that must work >.

The type of user (role) does not need to be human. You may optionally include a wishlist of additional user stories to add if time permits. Keep in mind that the type of user, the goal, the reason (if applicable within the system), all the common cases and all the special cases must be testable and demoable.

Consensus: As an attendee of an event, I want to know if my group has a consensus (aka. a match) so that I can see which restaurant the group wants to go to.

My conditions of satisfaction are:

- When all members of that group 'like' one or more restaurants in the list, the app would notify all members that they agree on these restaurants.
- When none of the options are liked by every member of the group, the app should be able to notify them that they didn't reach a consensus.

Adding my own options: As the attendee of an event, I want to also add my own choice of restaurant to the list so that if I know a restaurant that is great that is not in the list or if I don't like any of the restaurants, I can add my own suggestions.

My conditions of satisfaction are:

- I am able to search for a specific restaurant that is not already included in the list to the existing list
- Other users in the same group will be able to see my new addition of restaurant
- I should be able to filter searching restaurants by their cuisine and/or location
- The new restaurant can be selected as a match if all of the group members like it
- If the restaurant that I searched for is already in the list, then I should not be able to add it again to the list

Invite friends/family to match: As an organizer of an event, I want to send a link to my group of friends/family so that they can join the group and vote for their preferences in my particular group.

My conditions of satisfaction are:

- I am able to generate a link for my specific group

- Other users can use this link to join my group
- When other attendees join my group, I should be able to see their names to identify who is currently in the group
- After other attendees have joined the group, they should be able to start swiping
- If the person who received the invitation link is not logged in or registered, they should be redirected to a login/registration page and continue the process to join a group after logging in successfully.

Statistics graph: As an attendee of an event, I want to have a graph showing how many people chose each option when no match was made at the end of the restaurant list so that we can decide on an optimal option.

My conditions of satisfaction are:

- A statistics on how many people chose each option will pop up to every member of the group if everyone has finished matching but no consensus was reached.
- If someone decides to add new options to the list at the end of matching, the statistics should not be shown so that the members can continue matching.
- After the statistics is shown, members of the group should not be able to add new options to the list again.

Recommendation: As an attendee of an event, I want to see the most relevant options (like restaurants near me) so that I do not have to filter through a long list to find good places.

My conditions of satisfaction are:

- When adding new restaurants to the list, users should be able to click a “Recommendation” button to see a list of items recommended by the system.
- Users should be able to add items directly from the recommendation list.
- Recommendation should include popular items from restaurants that are near the user.

Part 3

Explain how you will conduct acceptance testing on your project. This means that every MVP user story must be associated with a plan for user-level testing. The test plan should address both common cases and special cases. Discuss sample inputs the user or client would enter and the results expected for the corresponding test to pass vs. fail. Note inputs might come from files, network, devices, etc., not necessarily from a GUI or command line, and results might involve changes in application state, files, outgoing network traffics, control of devices, etc., not necessarily outputs via a GUI or command line. You may optionally discuss testing plans for your wishlist additional user stories, if any.

Consensus:

Case 1:

When all members of a group agree on the same restaurants, the system should present the matching result to every member.

Sample input: 3 users in the group swipe right for the same restaurant. All attendees stay on that page after they completed their own voting process. All of them have voted for the same restaurants (let's say "Shake Shack").

Expected output: the same page shows that matching results after the matching ends on every members' device. The matching results are the intro of "Shake Shack".

Case 2:

When members of a group cannot agree on their preferences, the system should notify everyone that no match is successful and present the statistical graph to every member.

Sample input: 3 users in the group swipe right for some restaurant. All attendees stay on that page after they completed their own voting process. They voted for the different restaurants (let's say user 1 for "Shake Shack", user 2 for "five guys", user 3 swipe left for both of them).

Expected output: the same page shows that matching results after the matching ends on every members' device. The matching results are a corresponding graph.

Case 3:

When a member closes the app after voting for his or her options, he/she should be able to get notified when everyone has finished voting and a matching result is produced. When that member opens the app again, he/she would be able to see the matching result immediately.

Sample input: given an ordinary matching, one attendee finishes voting for all restaurants before others have completed theirs, and then that attendee closes the web page.

Expected output: when all attendees finish voting, that attendee would be notified by email or some notification mechanism, and when he or she opens the application again, the first thing presented is the list of restaurants that they have agreed on or a statistical graph if they reached no agreement .

Adding my own options:

Case 1:

When adding my option, we should see a separate screen when pressing on the corresponding button. In this screen, the user will be able to search for their restaurant by typing in the

restaurant name with some filters such as location and cuisine type if they want to. The user then should be able to pick the restaurant that they were looking for from the returned list and pick one to add to the existing list for the group. When they return to see the restaurant cards in the group, this new option added should appear as one of the cards. For example, a sample input may be ‘Ramen Jiro’ with location ‘Boston’. With this search, we will display a list of restaurants that matches this search such as ‘Ramen Jiro Newbury Street’ and ‘Ramen Jiro Harvard Square’. The user will pick what they were looking for, ‘Ramen Jiro Newbury Street’, and add it to the list. When they return to the list in the group, this new addition of ‘Ramen Jiro Newbury Street’ should be correctly displayed in the list.

Case 2:
When adding your option, there may be a case where the restaurant that you wanted to add was already included in the group list in the first place. For instance, continuing from the example above, if ‘Ramen Jiro Newbury Street’ was already included in the group list, then the user should not be able to add the same restaurant to the list. In this case, the app should display an error message ‘Restaurant already exists in the list’.

Invite friends/family to match:

Case 1:
After a user successfully creates a group, they should be able to see an invitation link created that they can copy/paste to share to other people that should join the group. After another user receives the invitation link, they should be able to click on it and be redirected to and join the group that invited them. Once the user joins the group, they should be able to immediately start swiping to indicate their preferences on the restaurants.
In this case, the sample input would be clicking on the button to create a group and the output will be the unique link associated with the group that is used to invite other people.

Case 2:
For each invitation link, no duplicate users should be able to join the same group. For example, if a user clicks on the link to join the group and successfully joins the group, then when they click on the same invitation link as the same user, they should not be able to join the group since they are already in the group. In this case, our sample output should be an error message such as ‘You are already in this group’. After displaying this message, it should not display a duplicate user in the group and the user who tried to join twice should be able to start swiping under their account.

Case 3:
For a person that has not registered for the app, when they open the invitation link, they should be redirected to the login/registration page. After they finish registration and successfully logged in, they can continue to join the group.

Statistics graph:

Case 1:
After all the group members finished swiping all the restaurant and there were no matches, a statistics graph will be displayed automatically, showing each restaurant's score. Top restaurants will be marked. If more than one restaurant ranked top, it will recommend the one with a higher rating score. In this case, the sample input would be all group members finishing swiping and the output would be a sorted restaurant information and displayed in graph.

Case 2:
After all the group members finished swiping all the restaurants, let one member add more options and the rest of members end swiping. The statistics graph should not be shown until all the members finished the second round of matching.

Recommendation:

Case 1:
Users should be able to see a list of recommended restaurants near them when they click the "Recommendation" button in the searching list. The top items in the list should either be close to the user or have a high popularity rating. After users click add for a specific item in the recommendation list, the item should be available for users to match.

Case 2:
Users should be able to see a list of recommended restaurants near them when they click the "Recommendation" button in the searching list. If the system fails to get the location of the user or the location is invalid, the "Recommendation" button should not be shown until we get a valid location, and a prompt will pop up.

Part 4

Identify the specific facilities corresponding to JDK, Eclipse, Maven, CheckStyle, JUnit, Emma, Spotbugs, and SQLite that your team plans to use. That is, state what you plan to use for compiler/runtime (or equivalent), an IDE or code editor, a build tool (or package manager if 'build' not applicable), a style checker, a unit testing tool, a coverage tracking tool, a bug finder (that's not just a style checker), and a persistent data store appropriate for your chosen language(s) and platform(s). If different members of the team plan to use different tools, please explain. It is ok to change your choice of tools later after you start developing your application.

Backend:

- Runtime: Python 3.7
- IDE: PyCharm (2020 versions) / Intellij IDEA
- Package manager: pip 18.1 or above
- Style checker: pylint + Black
- Unit testing: Pytest 6.1
- Coverage tracking: pytest-cov
- Bug finder: flake8
- Data store: consistent data on MongoDB; caches on Redis

Frontend:

- Compiler: TypeScript compiler (TS 3.8)
- Framework / Library: React v17
- IDE: Intellij IDEA
- Build tools: webpack
- Style checker: eslint
- Unit testing: Jest
- Coverage tracking: Jest
- Bug finder: eslint