**Yingqiao Xiong, Scott Woods, Nathalie Langlois, Yue Zhou**

Spring 2016

Restaurant Recommender

Version 2.0

# Use Cases

|  |  |  |
| --- | --- | --- |
| ID | Primary Actor | Use Case Title |
| 1 | User | Log in |
| 2 | User | Create a restaurant category |
| 3 | User | Generate a restaurant recommendation |
| 4 | User | Update restaurant history |

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: | UC-1 | | |
| Use Case Name: | Log in | | |
| Created By: | Nathalie Langlois | **Last Updated By:** | Nathalie Langlois |
| Date Created: | 3/13/16 | **Date Last Updated:** | 3/13/16 |

|  |  |
| --- | --- |
| Actors: | App User |
| Description: | When app is started, user must log in using existing credentials. |
| Trigger: | User opens app |
| Pre-Conditions: | None |
| Post-Conditions: | User has logged in and been authenticated. |
| Normal Flow: | UC-1.0 Log in   1. User enters email 2. User enters password 3. User selects “Log in” 4. System authenticates email and password combination    1. System sends information to data store    2. System validates the input 5. System displays logged in user’s home page |
| Alternate Flows: | UC-1.1 User must create account   1. User selects “Create Account” 2. System prompts user for account details    1. Enter email    2. Enter password twice 3. System validates input against requirements    1. If invalid, highlights incorrect fields and return to step 2 4. System displays confirmation message 5. System stores user information in data store 6. System displays main page   UC-1.2 User enters invalid log in information  *Begin after step 4.1*   1. System does not find username and password combination 2. System alerts user that the information is incorrect 3. System prompts user to try again 4. Return to normal flow step 1   UC-1.3 User chooses to log in with Facebook   1. User selects “Log in with Facebook” 2. System prompts user to log in using Facebook |
| Exceptions: | None |
| Includes: | None |
| Priority: | High |
| Frequency of Use: | 1x per use of app |
| Business Rules: | Each email may only be linked to one account.  A password must contain 6 characters. |
| Special Requirements: | All database calls must take under 1 second.  Initial load time must be under 2 second. |
| Assumptions: | None |
| Notes and Issues: | Revisit UC-1.3 once we have a more solid understanding of Facebook API  Version 2:   * Forgot password * Limit login attempts |

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: | UC-2 | | |
| Use Case Name: | Create a restaurant category | | |
| Created By: | Nathalie Langlois | **Last Updated By:** | Nathalie Langlois |
| Date Created: | 3/13/16 | **Date Last Updated:** | 3/13/16 |

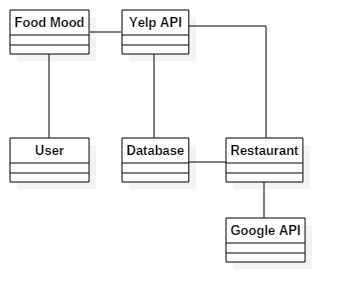
|  |  |  |  |
| --- | --- | --- | --- |
| Actors: | App User | | |
| Description: | User creates restaurant category (“Food Mood”) based off of inputted attributes. | | |
| Trigger: | User clicks button to create a food mood | | |
| Pre-Conditions: | User has logged in and been authenticated. | | |
| Post-Conditions: | Food mood is stored as part of user’s account configuration.  Food mood is displayed on main page. | | |
| Normal Flow: | UC-2.0 Create a restaurant category   1. User is prompted to enter a name for the food mood 2. User is prompted to select if desired option filter is cuisine, price, or dining type 3. User selects filter option 4. System displays elements for user within the selected option 5. User selects an element 6. System prompts user to enter more options    1. Return to step 2 (Repeat until at most 5 elements selected) 7. User selects “Done” 8. System stores food mood name and elements 9. System displays main page with food mood list updated | | |
| Alternate Flows: | UC-2.1 User does not complete food mood  *Begin at any time*   1. User clicks to exit food mood creator 2. System prompts user to finish creating the food mood    1. User clicks to continue, resume normal flow 3. User clicks to close 4. System displays main page without storing food mood | | |
| Exceptions: | None | | |
| Includes: | UC-1 Log in process | | |
| Priority: | High | | |
| Frequency of Use: | 1.5x per use of the app | | |
| Business Rules: | Selected elements are treated as additive (i.e. if a user selects “American” and “Japanese” the system will display American restaurants and Japanese restaurants, not American Japanese restaurants). | | |
| Special Requirements: | All database calls must take under 1 second.  Initial load time must be under 2 second. | | |
| Assumptions: | None | | |
| Notes and Issues: | Version 2:  Other attribute types: speed, location/distance, takeout vs. delivery vs. eat in, rating  Submit/create new elements for food mood (e.g. “Don’t like your options? Tell us what we are missing”) | | |
| Use Case ID: | UC-3 | | |
| Use Case Name: | Generate restaurant recommendation | | |
| Created By: | Nathalie Langlois | **Last Updated By:** | Nathalie Langlois |
| Date Created: | 2/16/16 | **Date Last Updated:** | 3/13/16 |

|  |  |
| --- | --- |
| Actors: | App User |
| Description: | After either selecting an existing restaurant category (“Food Mood”) or generating a new one, the system will return a recommendation based on the Food Mood and the user’s history. |
| Trigger: | User clicks button to generate a new recommendation |
| Pre-Conditions: | User has logged in and been authenticated.  User needs location services enabled.  User has created a Food Mood |
| Post-Conditions: | User has selected a restaurant.  Restaurant has been added to the user’s history.  System is set to prompt user for a rating of their experience. |
| Normal Flow: | UC-3.0 Generate a restaurant recommendation   1. User selects existing Food Mood 2. System retrieves user preferences from data store 3. System retrieves user’s location 4. System retrieves options from Yelp    1. Selects restaurants that match the food mood    2. Selects restaurants that are open    3. Selects restaurants that are near the user’s location 5. System generates list of recommendations 6. System displays first restaurant recommendation 7. User selects “go” 8. System displays restaurant information 9. System displays fastest route to the restaurant |
| Alternate Flows: | UC-3.1 User does not select recommended restaurant  *Begin after normal flow step 6*   1. User selects “next” 2. System generates a new recommendation 3. Resume normal flow at step 6   UC-3.2 User saves the restaurant  *Begin after normal flow step 6*   1. User selects “star” 2. Resume at normal flow step 7   UC-3.3 User does not select any restaurant  *Begin at any time*   1. User exits the Food Mood   UC-3.4 User goes through all recommendations  *Begin after UC-3.1 step 1*   1. System fails to generate a new recommendation because the user has seen the entire list 2. System prompts    1. “Select a new food mood” and begin at Step 1 of UC-3.3    2. “See recommendations again” and begin at Step 6 of UC-3 |
| Exceptions: | UC-3.0.E.1 Cannot find any recommendations  *Begin after normal flow step 4*   1. System does not find any restaurants that match the given specifications 2. System displays error message 3. System prompts user to try a different food mood 4. Resume normal flow at step 1 |
| Includes: | UC-1 Log in process  UC-2 Create Food Mood |
| Priority: | High |
| Frequency of Use: | 2x per use of the app |
| Business Rules: | Restaurants are not displayed if they have less than 3 stars.  Restaurant must be within 10 miles of the user. |
| Special Requirements: | All database calls must take under 1 second.  Initial load time must be under 2 second. |
| Assumptions: | Yelp servers are always available. |
| Notes and Issues: | Criteria that system uses to recommend restaurants  Is 10 miles a good location radius  How many restaurants does the system grab at a time?  Version 2  Customizable location radius  Incorporate reservation booking (Open Table)  Feature restaurants that we can make money from earlier  After X number of “next”, suggest user select a new food mood |

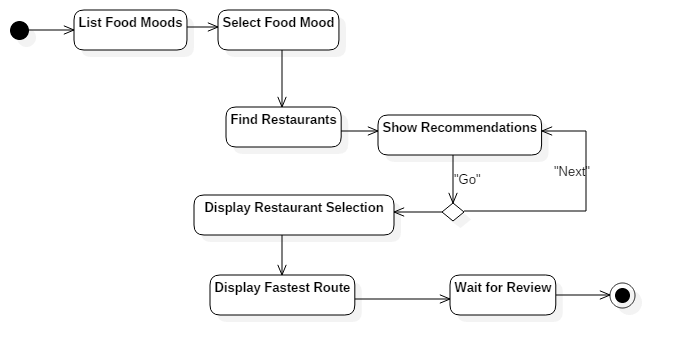
|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: | UC-4 | | |
| Use Case Name: | Update restaurant history | | |
| Created By: | Nathalie Langlois | **Last Updated By:** | Nathalie Langlois |
| Date Created: | 3/13/16 | **Date Last Updated:** | 3/13/16 |

|  |  |
| --- | --- |
| Actors: | App User |
| Description: | After selecting a restaurant, the user will be prompted to review the restaurant the next time they open the app. |
| Trigger: | User opens app |
| Pre-Conditions: | User has logged in and been authenticated.  User selected a restaurant during most recent use of app.  Restaurant has been added to the user’s history.  System is set to prompt user for a rating of their experience. |
| Post-Conditions: | User’s restaurant history has been updated with review.  Restaurant review has been added to data store. |
| Normal Flow: | UC-4.0 Update restaurant history   1. System prompts user to rate their most recent restaurant selection 2. User gives restaurant a thumbs up or thumbs down 3. System sends rating to data store 4. System displays logged in user’s home page |
| Alternate Flows: | None |
| Exceptions: | None |
| Includes: | UC-1 Log in process  UC-2 Create Food Mood  UC-3 Generate restaurant recommendation |
| Priority: | Medium |
| Frequency of Use: | 1x per use of the app |
| Business Rules: | None |
| Special Requirements: | All database calls must take under 1 second.  Initial load time must be under 2 second. |
| Assumptions: | None |
| Notes and Issues: | Alternate use case idea: update food mood criteria, view saved restaurants |

# UC-3 Class Diagram



# UC-3 Activity Diagram



# Architecture Choice

The base technology stack for this project will be as follows:

* **Client**: AngularJS
* **Server**: Node.js with Express.js framework
* **Data**: MongoDB with Mongoose ODM

## AngularJS

Angular offers several benefits that make it one of the most appropriate and best frameworks for our application. It is ideal for single-page applications, where the user is spending much of their time on one dashboard-style page and interacting with data. In our application, the user will spend most of their time on the Pandora-like dashboard, which will be the control panel to our restaurant recommendation generator. Angular enables fairly rapid prototyping because it handles a lot of the boilerplate code that doesn’t come with Backbone. Our group has little experience using a front-end framework, and learning a basic amount of Angular is quite simple compared to other frameworks.

## Node.js and Express.js

We chose to build a Node application for a few reasons. Most importantly, it keeps the client side and server side in the same language (Javascript), which will allow for increased efficiency when developing. Node has performance benefits that many other frameworks cannot match. The servers offer extremely fast execution and asynchronous I/O, which ultimately increases the speed of any I/O operation, including reading or writing to a network, disk, or filesystem. These types of servers are useful when many users are accessing and updating data at once. Another huge benefit of Node is the package management program, npm, which makes installing and maintaining project dependencies very easy.

The most common partner to Node is Express. Express is great for beginners, requires very little code, and works very easily with databases. It makes the creation of a RESTful API very simple. It also plays very well with MongoDB.

## MongoDB

Mongo is very frequently used with Node and various Javascript frameworks because it essentially allows the data to be saved in the same format that it’s transmitted in: JSON. Unlike traditional relational databases, Mongo allows data to be dynamic. Data can be created without defining structure, and the structure of data can be changed even once the database has been populated. It is flexible and allows changes to data models to be introduced easily. Mongoose, an object modeling tool, allows our models to be saved directly to the database without having to use any Mongo queries directly. It keeps code much more precise and easy to understand, and makes saving objects consistent.

## Other stacks?

Another more traditional framework used to develop websites is the LAMP stack, which is an Apache server running on a Linux computer Using MySQL as the database and PHP running on the server. This stack does not have a good front-end framework. Writing a single page application using pure Javascript/J-Query is certainly possible, but as it grows, it will become extremely difficult to maintain the different asynchronous calls. J-Query commands would need to constantly be called to replace HTML with new data from the server, whereas a framework like Angular will automatically tie templates to data in the client-side controllers. MySQL could certainly achieve the same goal, but MongoDB may be easier to work with in this situation because we are largely handling data from other APIs, rather than defining our own. Mongo will provide more flexibility with picking and choosing the data we need to save. Also, as our team has no experience with PHP, we believe that the challenge of learning a new language would be detrimental to the quality of our product.