Restaurant Recommender

VERSION 3.0

Yingqiao Xiong, Scott Woods, Nathalie Langlois, Yue Zhou

SPRING 2016

Table of Contents

Project Proposal

Restaurant Recommender is a responsive web app that generates restaurant suggestions based on location and user preferences. The app makes use of the Facebook API for account setup and Yelp's API to generate suggestions.

The user will start by creating an account, or using their Facebook log in. Once they are logged in, the user is able to create a Food Mood to capture the kind of restaurant they are looking for. This could be anything, for example "budget Italian" or "mid-range Asian-fusion." Our app will query Yelp to find results that match these preferences. The user will receive relevant restaurant suggestions one at a time, with the ability to save an individual restaurant to a wish list. The listing will display key information about the recommendations such as the name, description, and rating.

Restaurant Recommender will help the user discover new restaurants and will make the entire process - from discovery, to eating - as simple as possible.

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	View wish list

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:**	4/24/16

Actors:	App User
Description:	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 4.1. System sends information to data store 4.2. 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 “Register” 2. System prompts user for account details 2.1. Enter name 2.2. Enter email 2.3. Enter password twice 3. System validates input against requirements 3.1. If invalid, highlights incorrect fields and return to step 2 4. System stores user information in data store 5. 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 3 characters.
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: 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:**	4/25/16

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 details for the Food Mood 1.1. Enter name 1.2. Enter desired location 1.3. Enter description 1.4. Enter tags 2. User selects “Submit” 3. System searches tags and location on Yelp 4. System stores restaurant IDs of search result in data store 5. System displays main page with Food Mood selected
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 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:	Yelp servers are always available.
Notes and Issues:	Version 2: Other attribute types: speed, location/distance, takeout vs. delivery vs. eat in, rating Use the user’s location as the “location” Improve tag functionality Submit/create new elements for Food Mood (e.g. “Don’t like your options? Tell us what we are missing”) Prompt user to continue before exiting an incomplete Food Mood

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:**	4/25/16

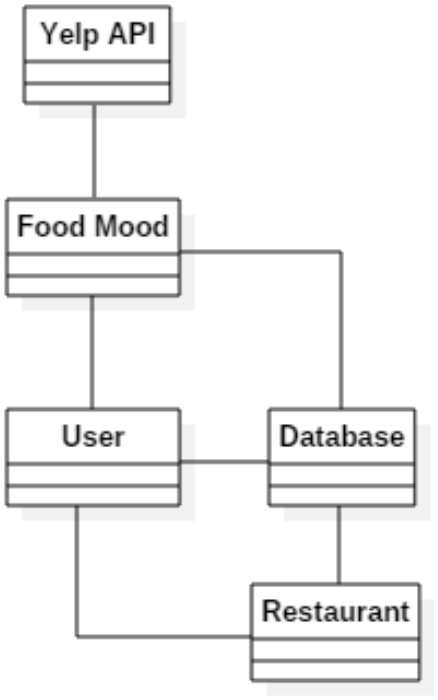
Actors:	App User
Description:	After either selecting an existing restaurant category ("Food Mood") or generating a new one, the system will return a recommendation based on the Food Mood.
Trigger:	User clicks button to generate a new recommendation
Pre-Conditions:	User has logged in and been authenticated. 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 list of relevant restaurant IDs from data store 3. System retrieves information of the next restaurant ID from Yelp 4. System displays restaurant recommendation
Alternate Flows:	UC-3.1 User does not select recommended restaurant *Begin after normal flow step 6* 1. User selects "next" 2. System accesses the next restaurant id on Yelp 3. Resume normal flow at step 4 UC-3.2 User saves the restaurant *Begin after normal flow step 6* 1. User selects "star" 2. Restaurant ID is saved to the user's wish list in the data store 3. Resume at normal flow step 4 UC-3.3 User does not select any restaurant *Begin at any time* 1. User exits the Food Mood
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 no restaurants
Includes:	UC-1 Log in process UC-2 Create Food Mood
Priority:	High
Frequency of Use:	2x per use of the app
Business Rules:	List of restaurants will repeat once user has gone through them all.

	The list of restaurants in the Food Mood will resume on the next restaurant ID on the list (based on the last restaurant the user viewed).
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:	Display error message if no restaurants are found 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

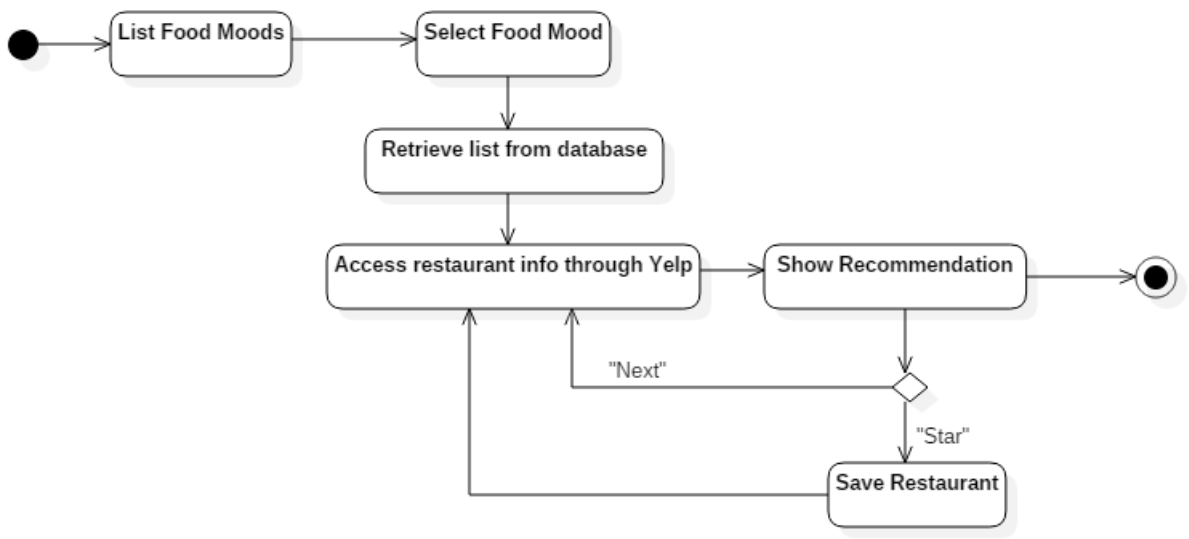
Use Case ID:	UC-4		
Use Case Name:	View wish list		
Created By:	Nathalie Langlois	**Last Updated By:**	Nathalie Langlois
Date Created:	4/13/16	**Date Last Updated:**	4/25/16

Actors:	App User
Description:	User can access the list of restaurants that they have “starred” in the past.
Trigger:	User selects “view wish list”
Pre-Conditions:	User has logged in and been authenticated. User has starred restaurants.
Post-Conditions:	User’s wish list has been updated.
Normal Flow:	UC-4.0 View wish list 1. System accesses data store 2. System retrieves list of the user’s saved restaurants 3. System displays list of restaurant
Alternate Flows:	UC-4.1 Remove item from wish list 1. User clicks to remove item from wish list 2. System updates user’s wish list in data store 3. System removes restaurant from the displayed list
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:	None

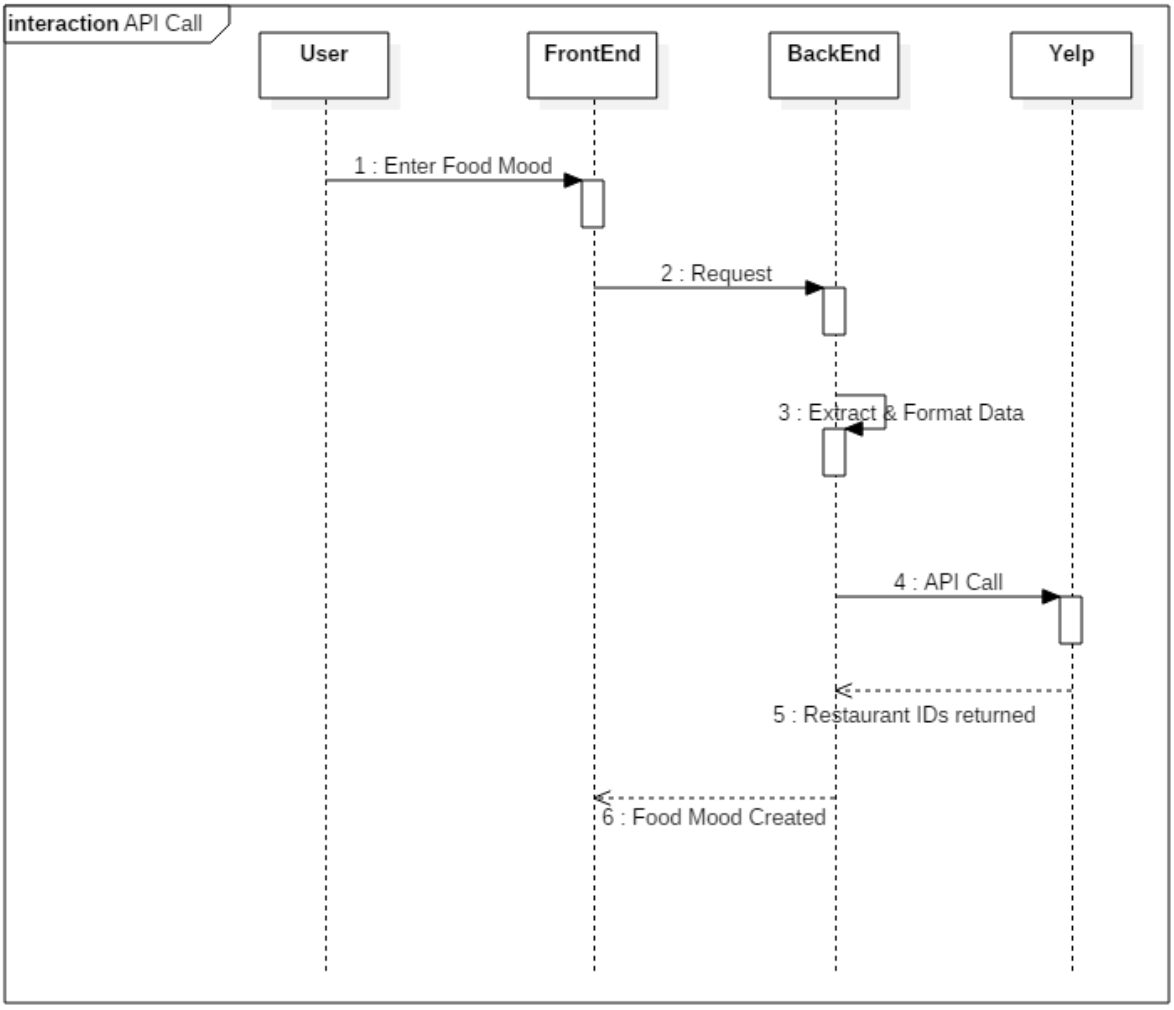
UC-3 Class Diagram



UC-3 Activity Diagram



API Call Sequence 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.

Version Log

Number	Changes
Version 1.0	Developed project proposal to documents
Version 1.1	Added UC-3
Version 2.0	Updated first use case with proper formatting Developed other three use cases Added activity diagram Added class diagram Added architecture choice
Version 2.5	Added data models to documentation Added sequence diagram for database API calls Made small revisions to all use cases to reflect implementation
Version 3.0	Made small revisions to UC-1, UC-2 and UC-3 to reflect app implementation Changed use case 4 from a restaurant history component to a wish list component Incorporated Facebook login to use cases Made small revisions to class diagram to reflect app implementation Made small revisions to activity diagram to reflect app implementation Created testing document Completed testing of system
Version 3.2	Added presentation to documentation Made minor revisions to use case 4 to reflect implementation of wish list Updated testing documentation