# Jessica – motivation, concept, & design

1. Motivation for development: Are you ever at a loss for what to make for dinner, yet have a bunch of leftover ingredients in your refrigerator?
2. Input ingredients with the option of selecting a cuisine and search for recipes.
3. Design process: since we decided that AJAX calls on API’s were the main function of the page, Alan started with AJAX calls (to make sure it worked) while Jason researched free API’s that would return recipes.
4. Problems: the first API, food2fook was not CORS-compatible, and would not return AJAX responses in the browser. Another API, Big Oven, charged a monthly fee for AJAX calls. Lastly, we had issues finding the search URL and key for Spoonacular. The documentation was difficult to read. After spending hours staring at a blank page, I choose to work on the front end, setting up the html structure and throwing on Bootstrap while Jason and Alan continued to research API documentation.

# Alan – AJAX call for dishes

1. Design process: First, input ingredient(s) and optionally, a cuisine type into the textbox. Clicking the “Find Recipes” button outputs a dish, the name of the dish, and a “Like” button. Based on the pictures, the user can decide which dish appeals to them. When clicking on the image, a step-by-step recipe displays.
2. We looked for free API’s with lots of recipes and images.
3. Problems: trying to figure out how to increment the “Likes” button of one recipe without affecting the others. Added the recipeID data attribute to each button. The number of “Likes” is pushed to Firebase and displays on the page in real time. Data is stored persistently so refreshing the page or calling the same dish at a later time does not affect the number of “Likes” for each dish.

# Jason – 2nd AJAX call for recipe ingredients and steps

1. Clicking on dish photo brings up a list of ingredients and step-by-step directions on how to prepare the dish. This required a second API call with Spoonacular. I looped through the JSON response to extract the ingredients from the array and display them to the page. Also, I pushed step-by-step recipe instructions to the page.
2. Problems: had to use a different query URL for this AJAX call (compared to the first one). Typos. Took the recipeID returned from first AJAX response to pass into the second AJAX call.

# Jessica – user submitted recipes

1. For the repeating element, we use a panel for users to submit their own recipe. The data is captured by firebase and automatically updates the recipe to the page in real time.
2. Problems: when you click the like button on the first page, it creates an empty row in the user submitted recipes table. This happens due to the way the “Likes” button is setup. When you click a recipe with 0 “Likes,” a new entry is added to the Firebase server. The user submitted recipes table looks for child\_added, so it appends what seems to be an empty row to the table.
3. Future development:
   1. Adjust image sizing so the dishes display more uniformly. Use Bootstrap JavaScript CDN so navBar works on smaller screen sizes (mobile).
   2. When clicking a dish to view the recipe, pop out the recipe in a new window (or new tab) so the user doesn’t have to scroll to the bottom (didn’t want the recipe to be posted to a new page).
   3. Call 4- and 5-star recipes with at least 100 reviews. This way, you don’t get recipes that have been rated 5 stars by 1 person. Alternatively, allow the user to choose to recipes of a certain rating (in case they *really* want Mongolian food with odd ingredients) in order to improve upon it.
   4. Replace our “Likes” system with a secondary 5-star rating system exclusive to this site.
   5. Nutritional data; Beverage pairing with each dish.
   6. User account authentication that lets users save their favorite dishes, view past search history, and personal recipe recommendations based on recipes user “Liked” or rated 5-stars.