Function-Oriented Design

We chose a function-oriented design for our prototype as our project was centered around manipulating data for a result. The user would be able to select different filters such as any allergies or specific types of food so that our search function could narrow down the results. Our prototype focuses on the result and not necessarily how we got there. Each component is specifically tailored towards achieving a result. The search component or the “app.js” is only goal to display results the user wants by filtering in and out the specific aspects of a meal that they want. Our interface component is only tailored for the user to select filters and apply a search. The functional oriented design was best suited for this project as the overall goal is display the result of manipulated user input. The main function of our project 3 is pulling from an already established library of data and manipulating it for the user. The components act as if they were a math problem; results equal “this and this, but not this”. The user inputs data and the search function take that data and tells the result to include and not include specific meals aspects. Though our design takes elements from a data-structured design, it is still primarily functional because we are manipulating the database rather than only calling on the data. Our design takes on an input and output shape but still differs from data-structure centered design because the results that are being displayed are user inputted implications being used to functionally manipulate the results.

Iterator, Builder, & Bridge

In our project 3, we used a series of design patterns to build our prototype. The design patterns that were used were an iterator, a builder, and a bridge. An iterator was used to cycle through each recipe result to make its personalized recipe card to be displayed. The iterator is a very useful design pattern that makes it easier for doing a specific task on a set of data without having to write out the code base for each and ever element of an array. The next design pattern used was the builder. The builder was important for displaying our recipe search results. The cards’ function was our builder function for our results. It would be called upon each time to build the personalized recipe card for each of resulted recipes found. The builder design pattern is important because it saves time for having to build a specific element multiple times. Being able to call a method or methods that build a specific element without having to write out the entire code base for that element each time its needed drastically saves time. The last design pattern we used was the bridge. The bridge was used for connecting our API and our user interface. The API was used to search websites for recipes, and it was bridged with our user interface where the user could input their ingredients and filters to find specific recipes. The bridge design pattern is important for separating a complex larger class into separate parts such as separating our user interface from the API.