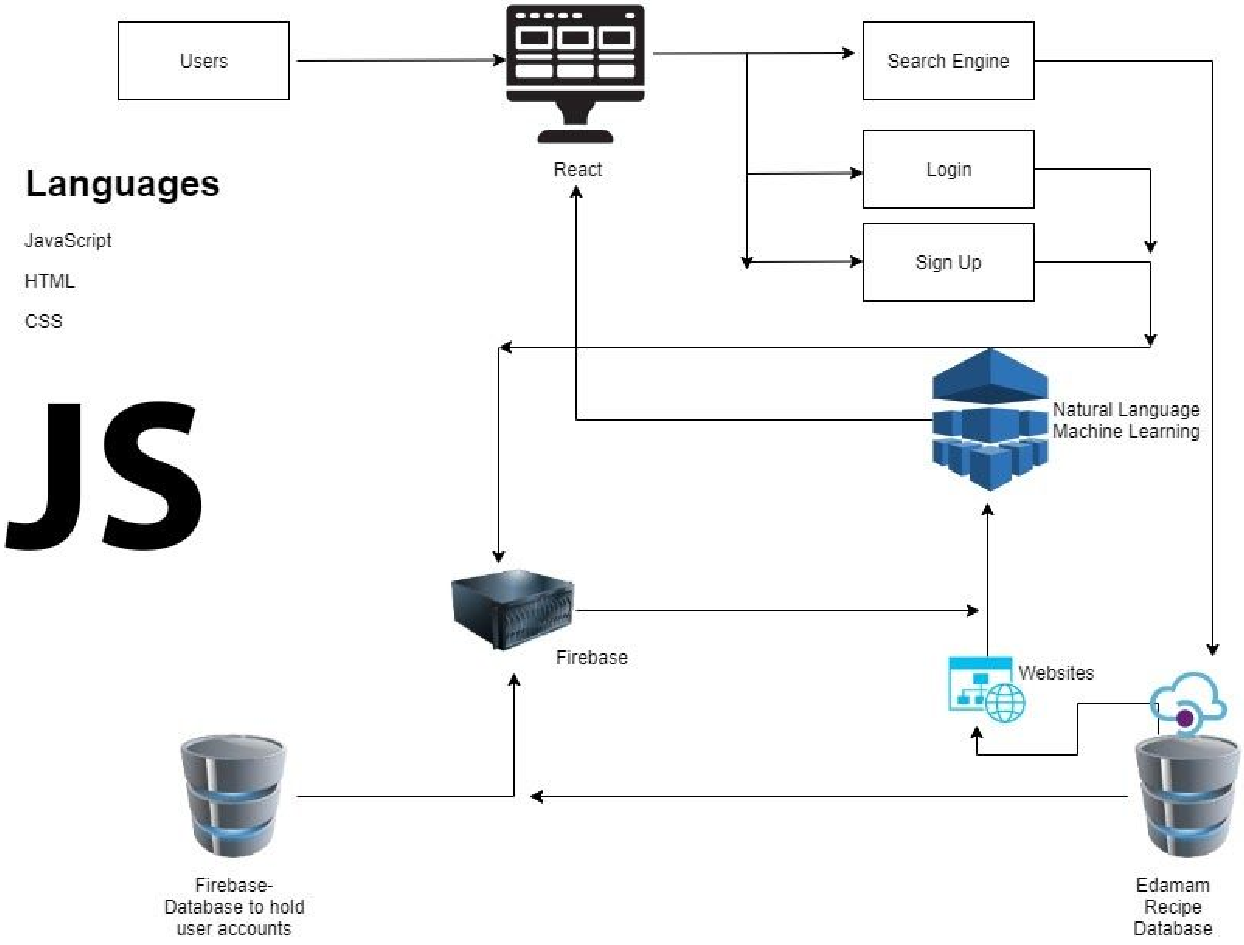
Architecture and Design Document

Table of Contents

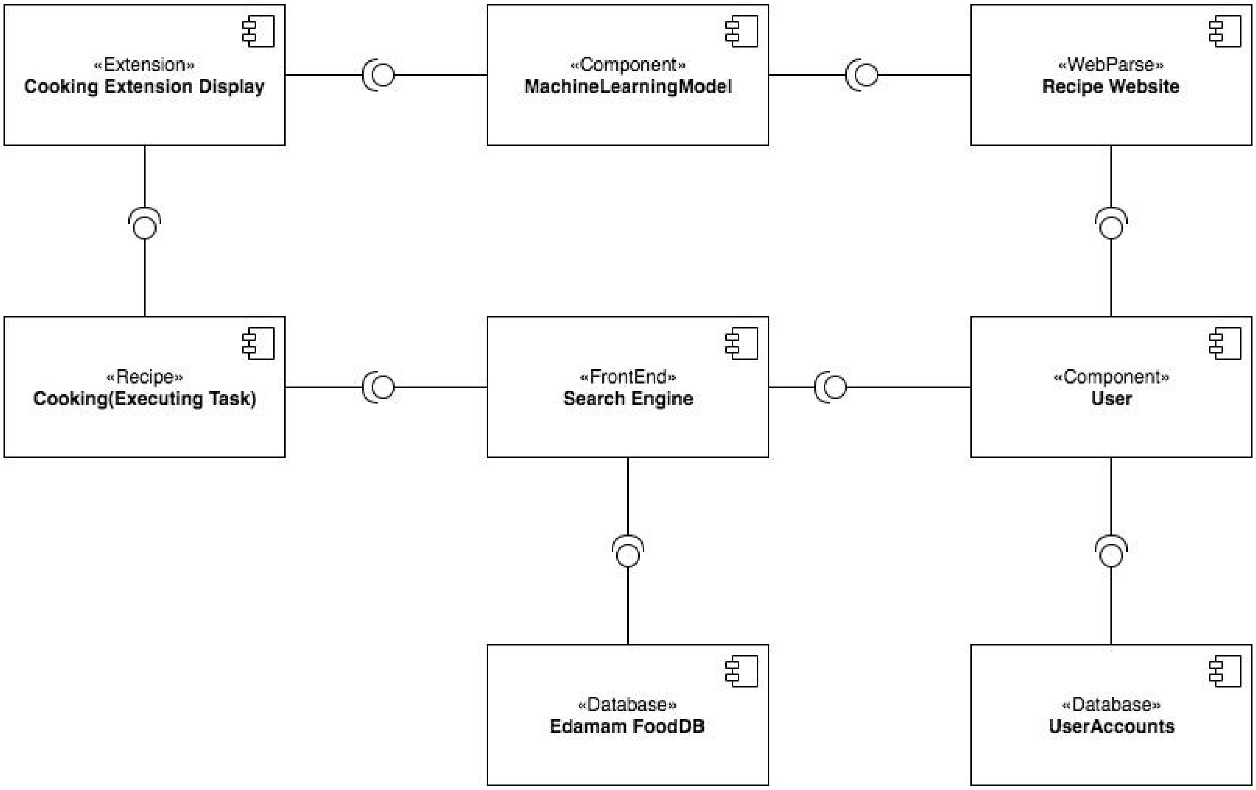
Diagrams

System Diagrams

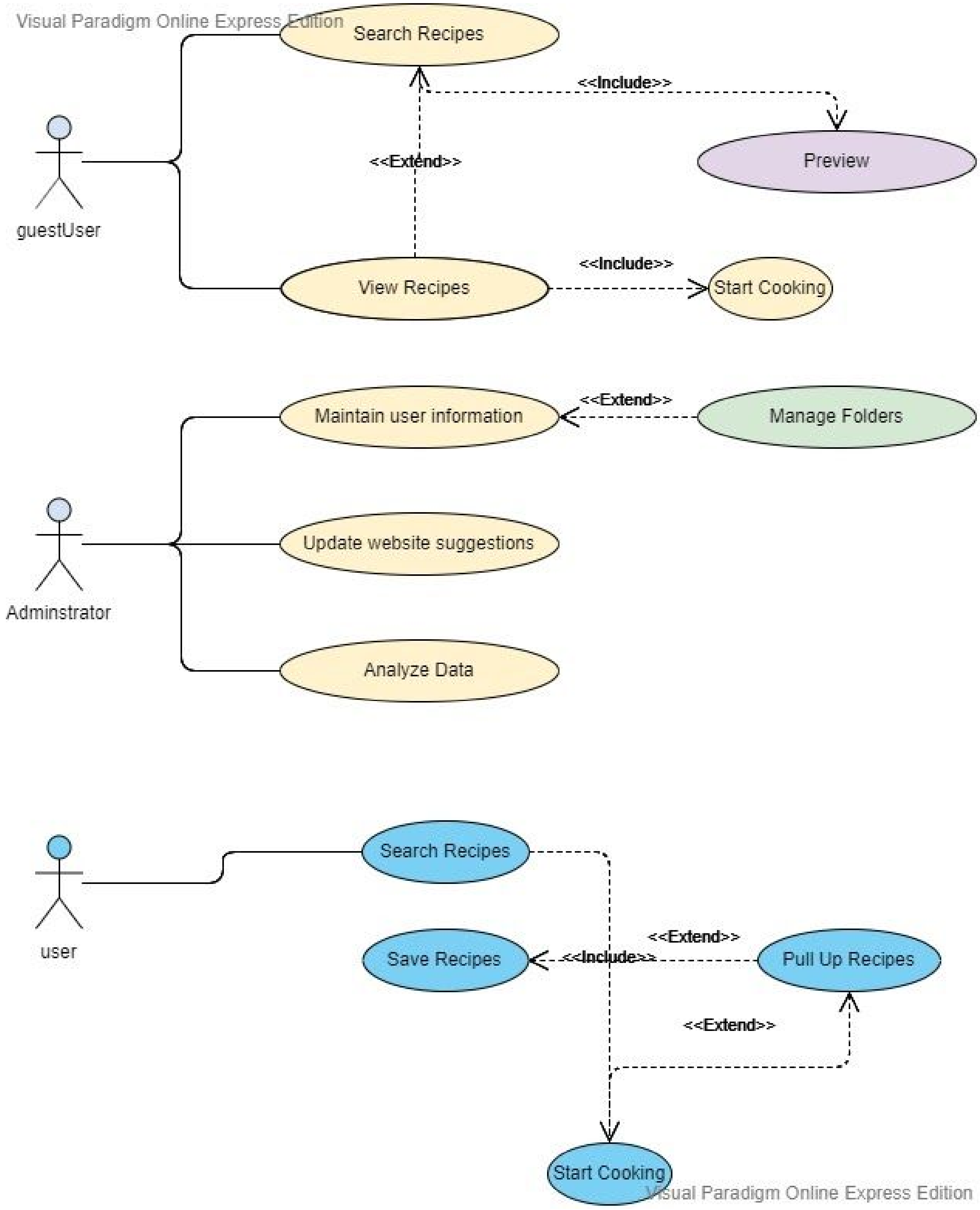


ODM

Component Diagrams

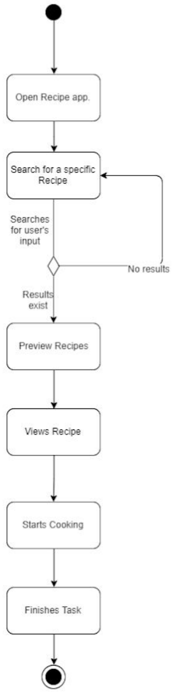


Use Case Diagrams

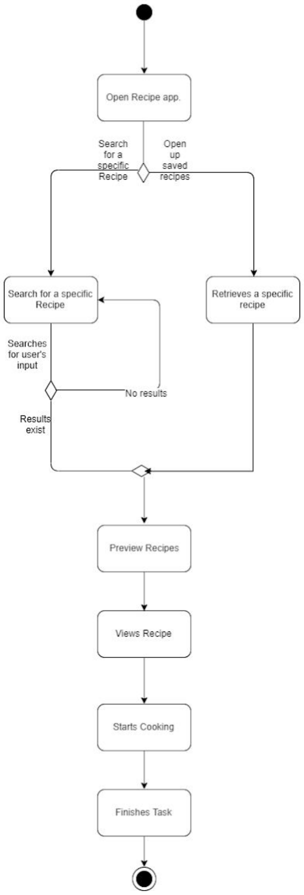


Activity Diagrams

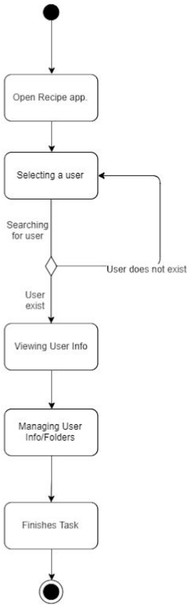
Guest User -
Searches Recipe



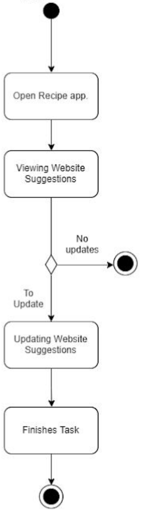
User -
Searches Recipe



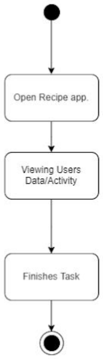
Admin -
Maintain User
Information



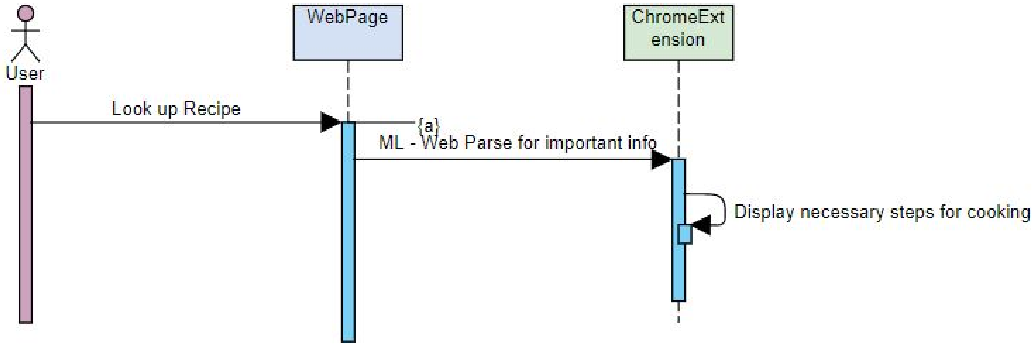
Admin -
Update Website
Suggestions

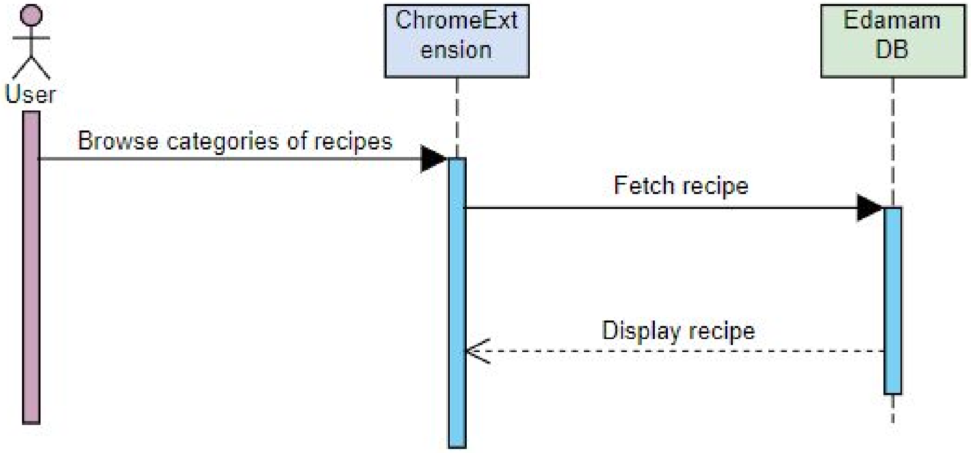


Admin -
Analyze Data

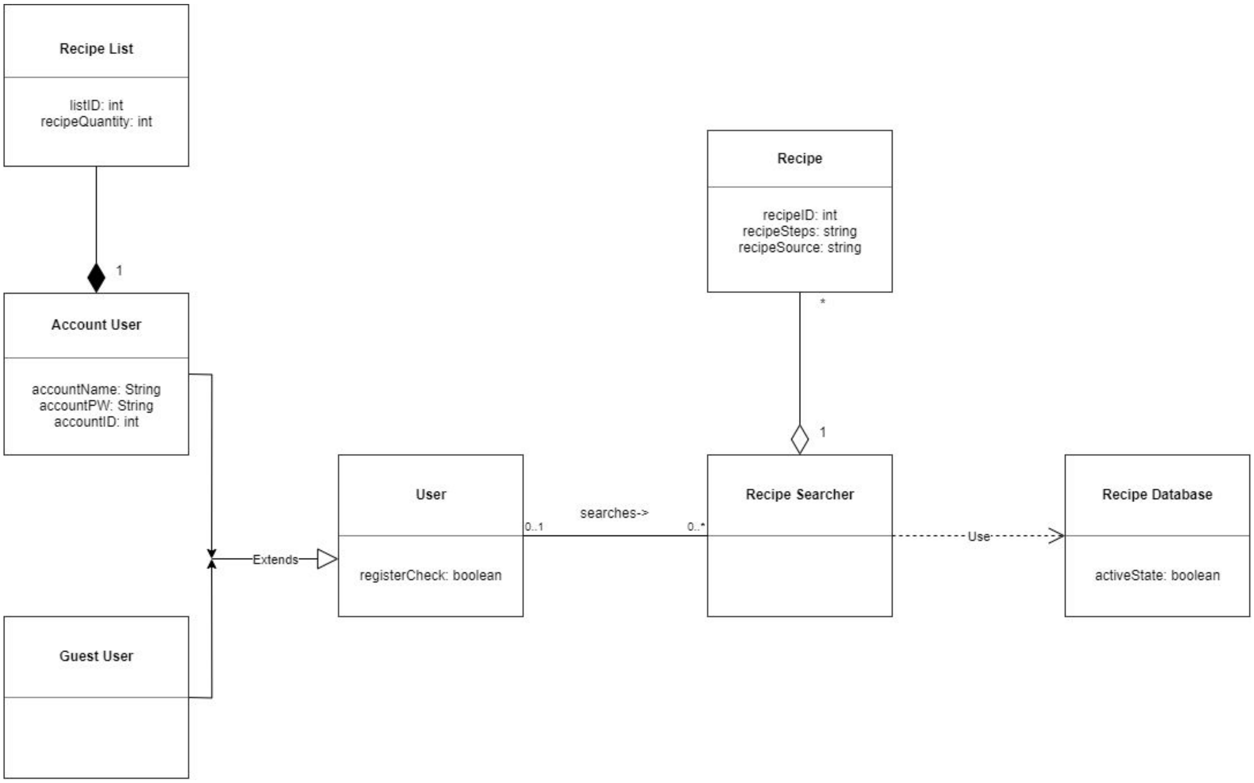


Sequence Diagrams

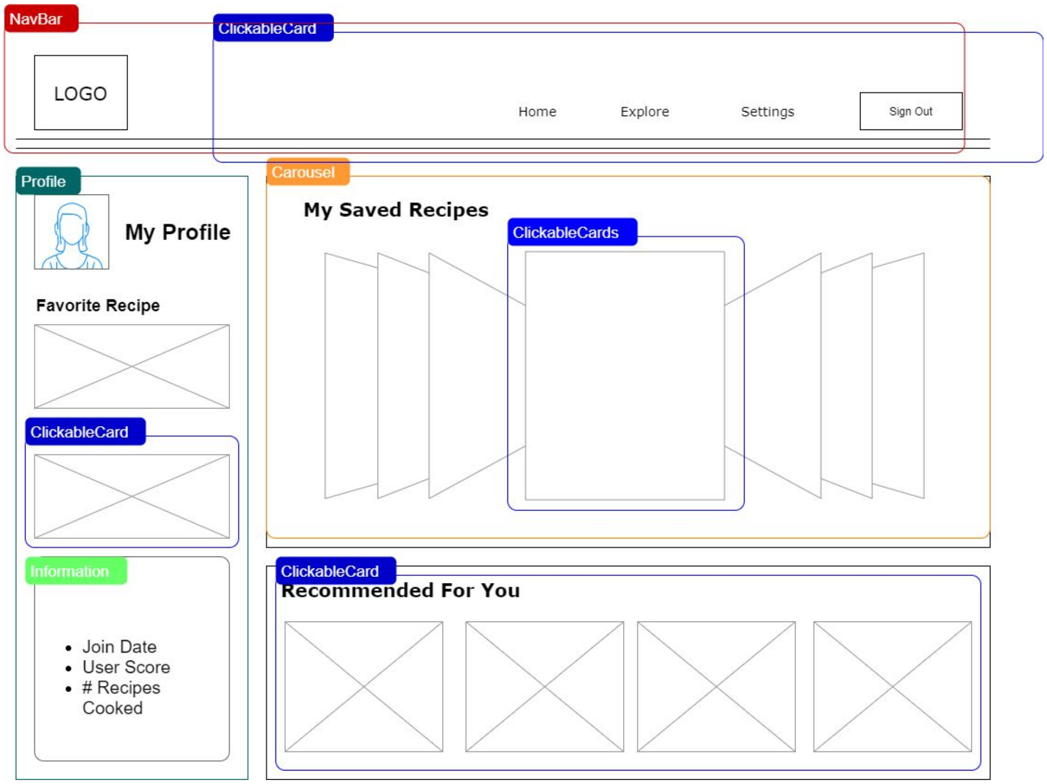

User
WebPage
ChromeExtension
Look up Recipe
{a}
ML - Web Parse for important info
Display necessary steps for cooking



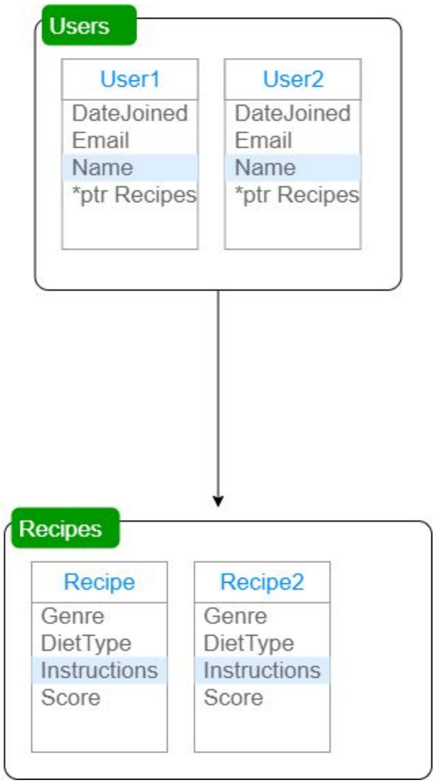
Class Diagram



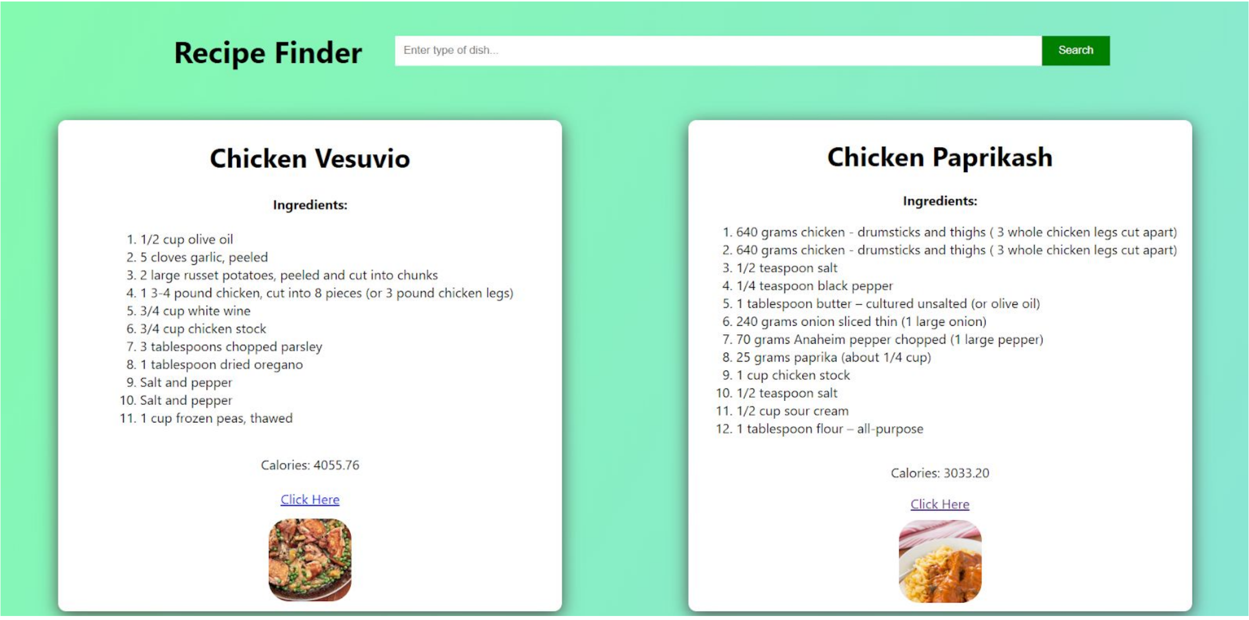
Front End Design:



Storage Architecture



Recipe Explore Tab UI Example



Chicken Vesuvio

Ingredients:

1. 1/2 cup olive oil
2. 5 cloves garlic, peeled
3. 2 large russet potatoes, peeled and cut into chunks
4. 1 3-4 pound chicken, cut into 8 pieces (or 3 pound chicken legs)
5. 3/4 cup white wine
6. 3/4 cup chicken stock
7. 3 tablespoons chopped parsley
8. 1 tablespoon dried oregano
9. Salt and pepper
10. Salt and pepper
11. 1 cup frozen peas, thawed

Calories: 4055.76

Click Here

Chicken Paprikash

Ingredients:

1. 640 grams chicken - drumsticks and thighs (3 whole chicken legs cut apart)
2. 640 grams chicken - drumsticks and thighs (3 whole chicken legs cut apart)
3. 1/2 teaspoon salt
4. 1/4 teaspoon black pepper
5. 1 tablespoon butter – cultured unsalted (or olive oil)
6. 240 grams onion sliced thin (1 large onion)
7. 70 grams Anaheim pepper chopped (1 large pepper)
8. 25 grams paprika (about 1/4 cup)
9. 1 cup chicken stock
10. 1/2 teaspoon salt
11. 1/2 cup sour cream
12. 1 tablespoon flour – all-purpose

Calories: 3033.20

Click Here

Trade-Off Analysis

The Client-Server Architectural Design was chosen as the architecture for the project. MVVM and MVC were other patterns considered for the project.

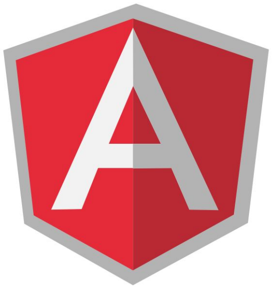
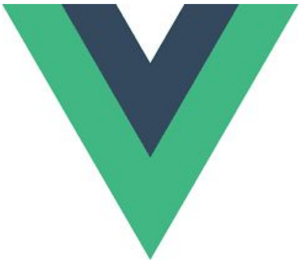
Client-Server Architecture	
Pros	Cons
- Better data sharing since data is all on a single server - Easier maintenance and better security control access - Resources are shared across different platforms - Ability to log into the system despite location or technology of the processor - Users can access the server through an interface rather than having to log into a terminal mode	- Multiple simultaneous clients can overload the server and cause slowdown - If server fails, no user can use the application until servers are fixed

Model-View-ViewModel (MVVM)	
Pros	Cons
- Separates UI and application logic to clearly define where certain code goes - Better unit testing, because it allows for testing of individual components without affecting the others - Developers can focus on either the UI or the application logic without worrying about the other, leading	- Adds complexity to Presentation Layer of the application, which adds a learning curve for some developers - Errors aren’t generated at compile time, but instead at run time, usually producing silent errors and making debugging harder

to safer coding	

Model-View-Controller (MVC)	
Pros	Cons
- High cohesion, low coupling - Easier to modify due to separation of concerns - Multiple developers can work components at the same time	- Adds complexity which adds a new learning curve for some developers - Developers need to maintain the consistency of multiple representations at once

Trade-Off Analysis : Front-end

Pros	Cons
 ● Component Reusability ● Google support ● Third-party integrations for better functionalities	● Steep Learning curve, difficult to learn
 ● Component Reusability ● Detailed documentation for ease of learning ● Very lightweight and easily integratable	● Small community due to being new, lack of support ● High pace of development

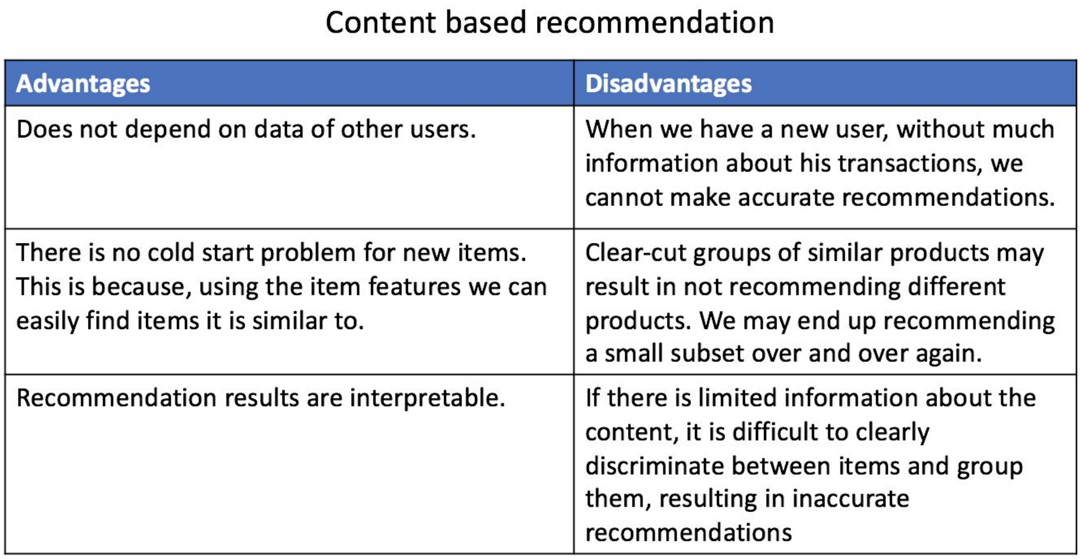
Trade-Off Analysis : Back-end

Pros	Cons
aws ● No data capacity - you pay for the amount you use on the cloud ● Very detailed documentation for ease of learning ● Array of tools ready to be deployed (ex. Database - Amazon Aurora, analytics)	● Price Packages (Developer, Business, Entreprise) ● No control over environment, dependent on Amazon
mongoDB ● Document-oriented NoSQL database for ease of access of indexing ● Direct use of JSON and JavaScript frameworks ● Free of cost	● Less flexibility in queries (ex. No joins) ● No support for transactions (updating documents/collections) → risk of duplication of data

1. **Major Architecture:** Client Server
2. **Component Choice:** Undecided.
3. **Language Choices:** JavaScript, HTML, CSS
4. **Framework Choices:** N/A
5. **Database Choices:** Firebase Firestore Database (NOSQL)

6. **Server vs Serverless Choices:** Firebase
7. **Front end Framework:** React
8. **API Choices:** Edaman Recipe Database
9. **Cloud Decisions:** Cloud Firestore
10. **Security Decisions:** Firebase Authentication
11. **Logs/Monitoring Choices:** Firebase Firestore Database (NOSQL)
12. **Process Decisions:** our program fits very well for the client server architecture due to the fact that users interact with our app, sends a request to the server, then the server returns the data back to the user. In this case, it would be a recipe request to the server.
13. **Future Additions:** N/A

Machine Learning Analysis + 10 Steps



Content based recommendation

Advantages	Disadvantages
Does not depend on data of other users.	When we have a new user, without much information about his transactions, we cannot make accurate recommendations.
There is no cold start problem for new items. This is because, using the item features we can easily find items it is similar to.	Clear-cut groups of similar products may result in not recommending different products. We may end up recommending a small subset over and over again.
Recommendation results are interpretable.	If there is limited information about the content, it is difficult to clearly discriminate between items and group them, resulting in inaccurate recommendations

1. **Analysis**
   When people search online for food recipes, many of those websites contain recipe instructions as well as unnecessary filler text such as a description about the history of the recipe. All people want to do is find the instructions to the recipe and begin cooking, but having that extra text makes it difficult for them to start cooking right away. To tackle this problem, we want to create an application that can parse through a recipe website and grab only the instructions for the recipe for people to use. We also want to create a recommendation system to recommend users other recipes based on what they've chosen to cook. In order to accomplish this, our initial idea of the type of Machine Learning that would be incorporated into our application would be a Natural Language Processing Model. This would allow the user to receive recommendations to other recipes based on the current search of their recipe as well as previous viewings of

recipes.

2. **Data Gathering**

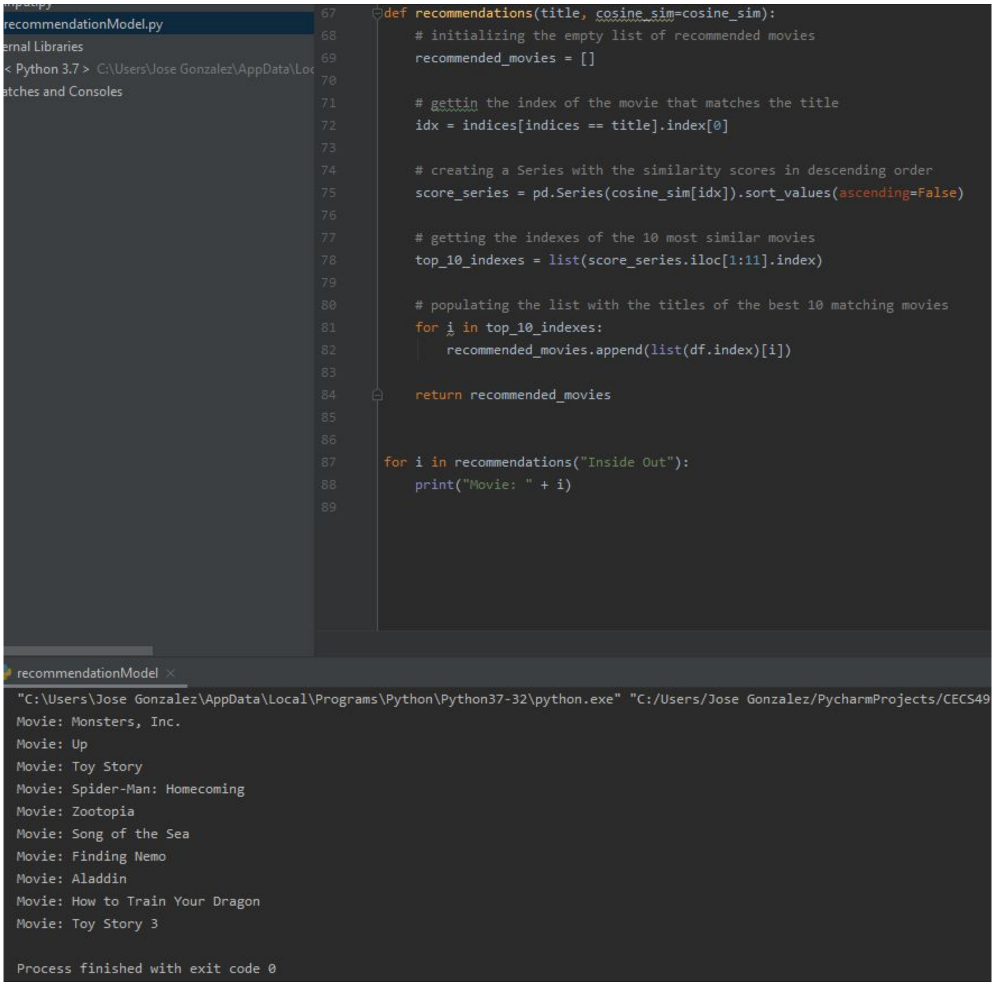
   Data was gathered by parsing various food websites and recipes gathered using APIs.

   a. Websites used:

      i. https://api2.bigoven.com/
      ii. https://developer.edamam.com/
      iii. https://foodnetwork.com/
      iv. https://delish.com/
      v. https://allrecipes.com/
      vi. https://simplyrecipes.com/
      vii. https://stackoverflow.com/

3. **Importation and Preprocessing of Data**

   If you see in our github folder under the recommendation system folder we have a csv file that was used to test out the model. It is a list of movies that we found and used that to test how the model works. The results would produce a list of the top 10 movies that would match with the user.



```
def recommendations(title, cosine_sim=cosine_sim):
    # initializing the empty list of recommended movies
    recommended_movies = []

    # gettin the index of the movie that matches the title
    idx = indices[indices == title].index[0]

    # creating a Series with the similarity scores in descending order
    score_series = pd.Series(cosine_sim[idx]).sort_values(ascending=False)

    # getting the indexes of the 10 most similar movies
    top_10_indexes = list(score_series.iloc[1:11].index)

    # populating the list with the titles of the best 10 matching movies
    for i in top_10_indexes:
        recommended_movies.append(list(df.index)[i])

    return recommended_movies


for i in recommendations("Inside Out"):
    print("Movie: " + i)
```

```
"C:\Users\Jose Gonzalez\AppData\Local\Programs\Python\Python37-32\python.exe" "C:/Users/Jose Gonzalez/PycharmProjects/CECS49
Movie: Monsters, Inc.
Movie: Up
Movie: Toy Story
Movie: Spider-Man: Homecoming
Movie: Zootopia
Movie: Song of the Sea
Movie: Finding Nemo
Movie: Aladdin
Movie: How to Train Your Dragon
Movie: Toy Story 3

Process finished with exit code 0
```

4. **Selection/Creation of a Machine Learning Model**
   We're focusing on applying a content-based recommendation model to our project. We have considered decision trees as an initial model to choose but it didn't really match the goal of our project. The goal is to train the computer to recognize human speech to the point where it can recognize cooking verbs and then depending on how it is used within a website, it would have the ability to decide whether or not that string of information should be fetched for the user to see and use for cooking. This would be much more effective than a decision tree.

5. **Training the Model**
   Our very first step would be to create an algorithm that differentiates strings that relate to cooking/recipe compared to unrelated strings. In order to do this, we would need to gather up a dictionary of common words used from recipe web pages and there on train our algorithm to recognize and differentiate words apart from each other.

6. **Evaluating Results**
   At this current standpoint, the script right now can parse a long amount of strings, break them down into individual words, acquire the verbs and count the occurences of those verbs. The higher number of occurrences would generally mean that specific verb is very common for recipes but not often the case. This is the reason why we would like to train the model and implement a point scoring system that would give emphasis to those specific verbs instead of relying on number of occurrences alone.

7. **Predictions**
   We predict that once the model has been trained, the model would be able to split and parse all the cooking-related verbs into point categories, ranking them from the most used verbs to the least used verbs in addition to verbs that are not used for cooking at all. Once the model has acquired these verbs, it'll split the sentences and present them in a numbered list of recipe steps for the user to see. As of right now, there is little evaluation for this model because the model is currently incomplete.

8. **Deployment and Reuse**
   As stated previously, the model is currently incomplete and we only have scripts that allow the parsing of data and splitting them up, counting the occurrences of those verbs and placing them onto different point categories. However, the code is currently uploaded on GitHub for public usage if the user desires so.