

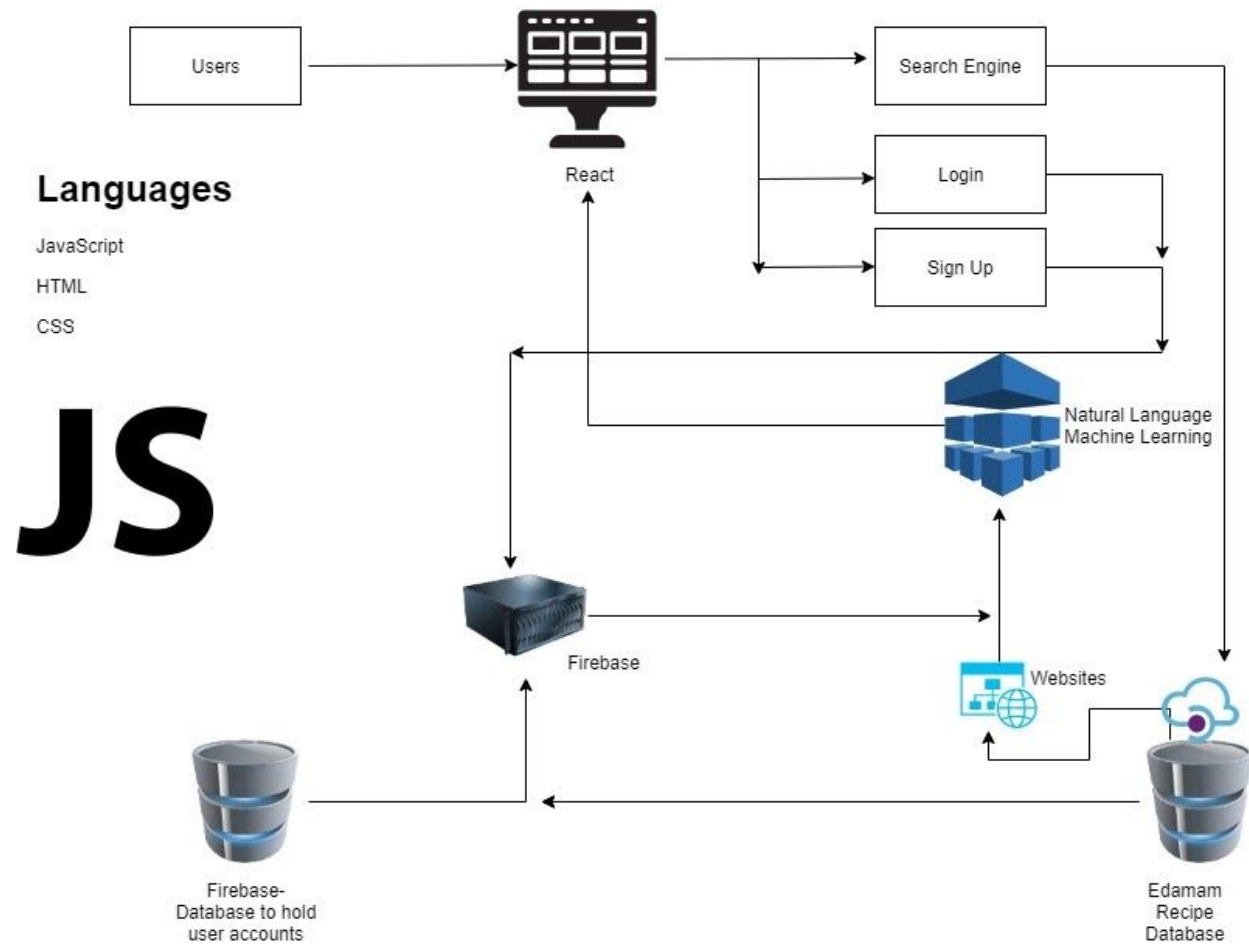
Architecture and Design Document

Table of Contents

Diagrams	2
System Diagrams	2
Component Diagrams	3
Use Case Diagrams	4
Activity Diagrams	5
Sequence Diagrams	6
Class Diagram	7
Front End Design	8
Storage Architecture	9
Recipe Explore Tab UI Example	9
Trade-Off Analysis	10
Machine Learning Analysis + 10 Steps	14
Technology Depth and Innovation	21

Diagrams

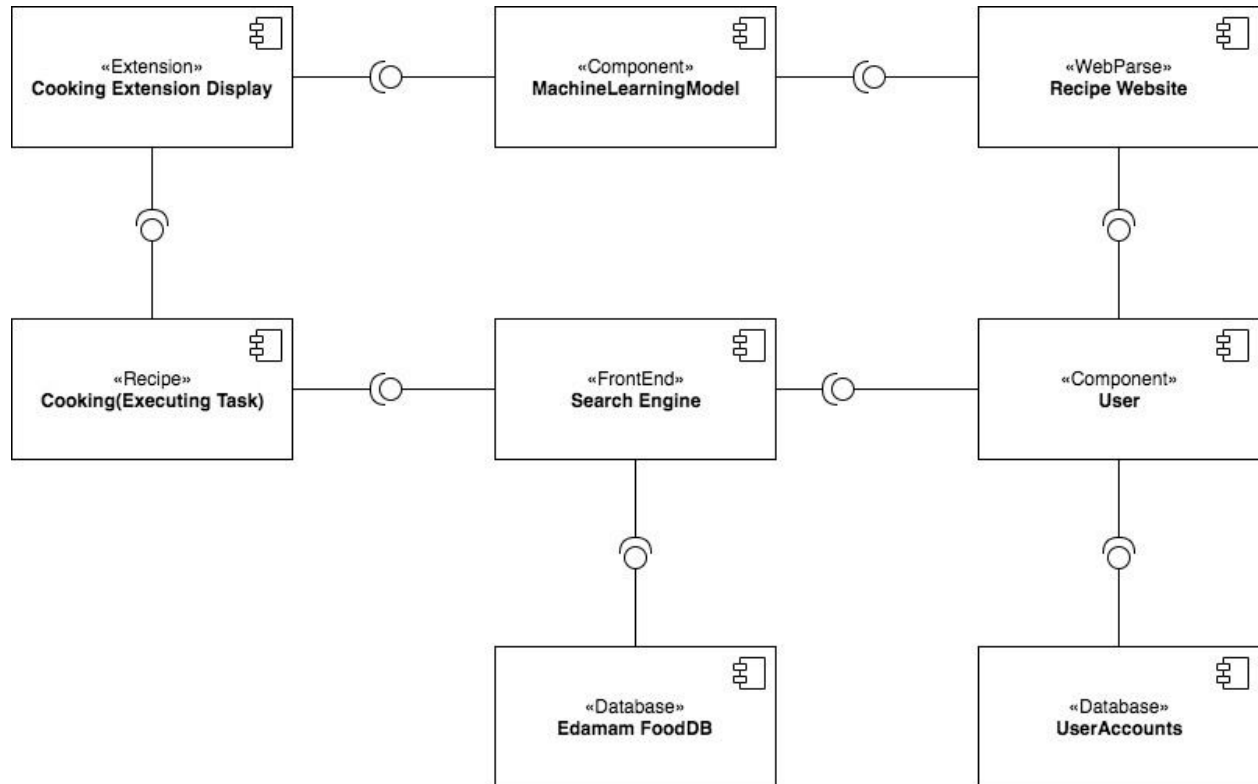
System Diagrams



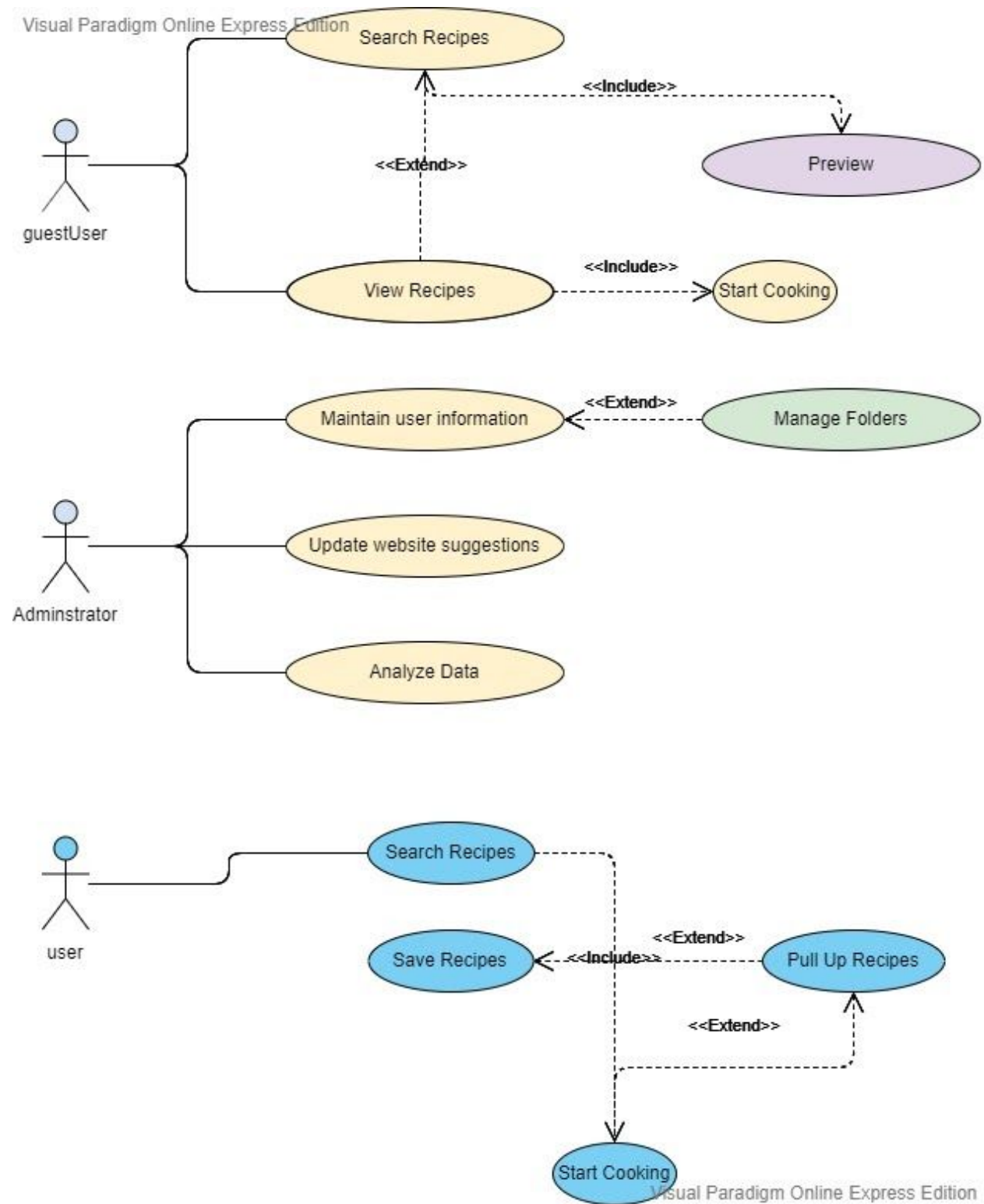
ODM



Component Diagrams

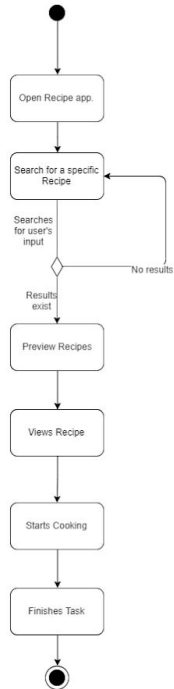


Use Case Diagrams

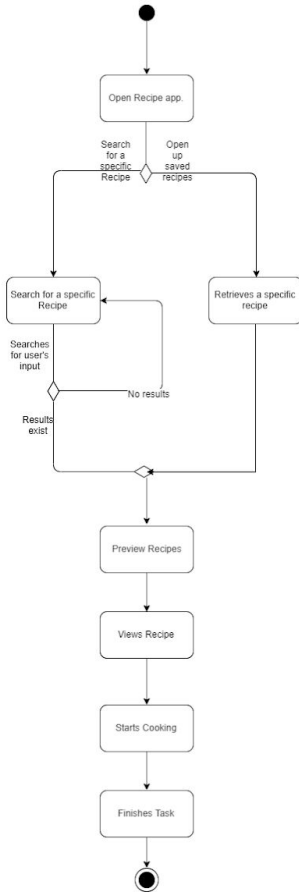


Activity Diagrams

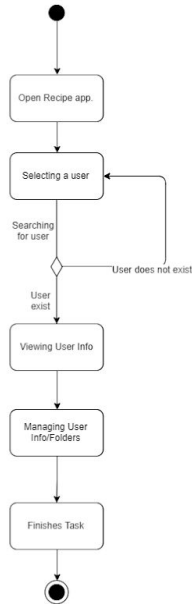
Guest User - Searches Recipe



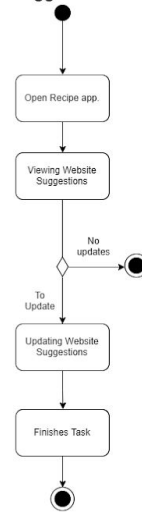
User - Searches Recipe



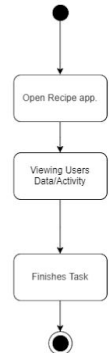
Admin - Maintain User Information



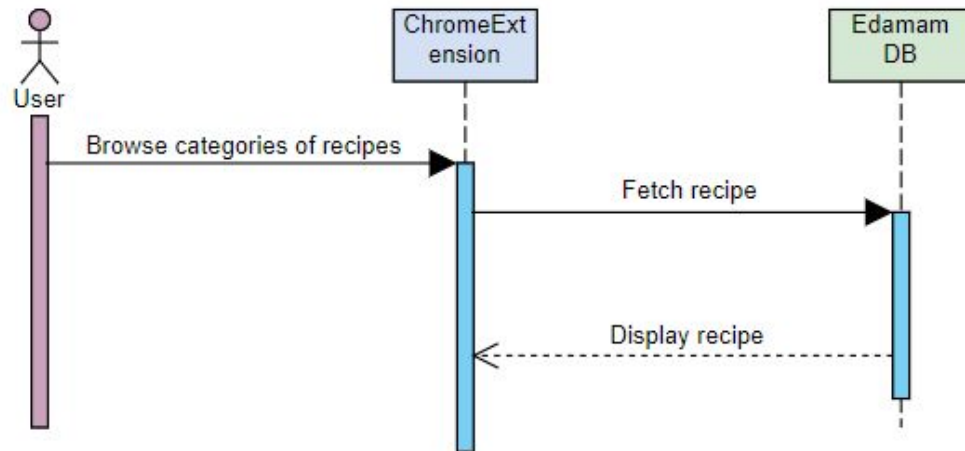
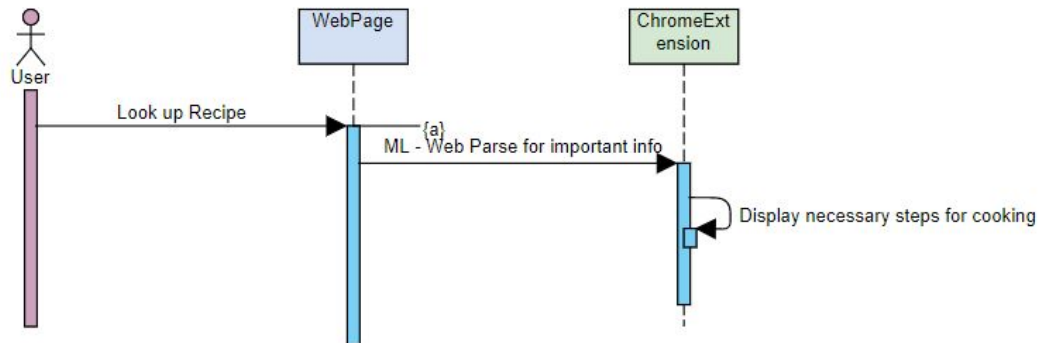
Admin - Update Website Suggestions



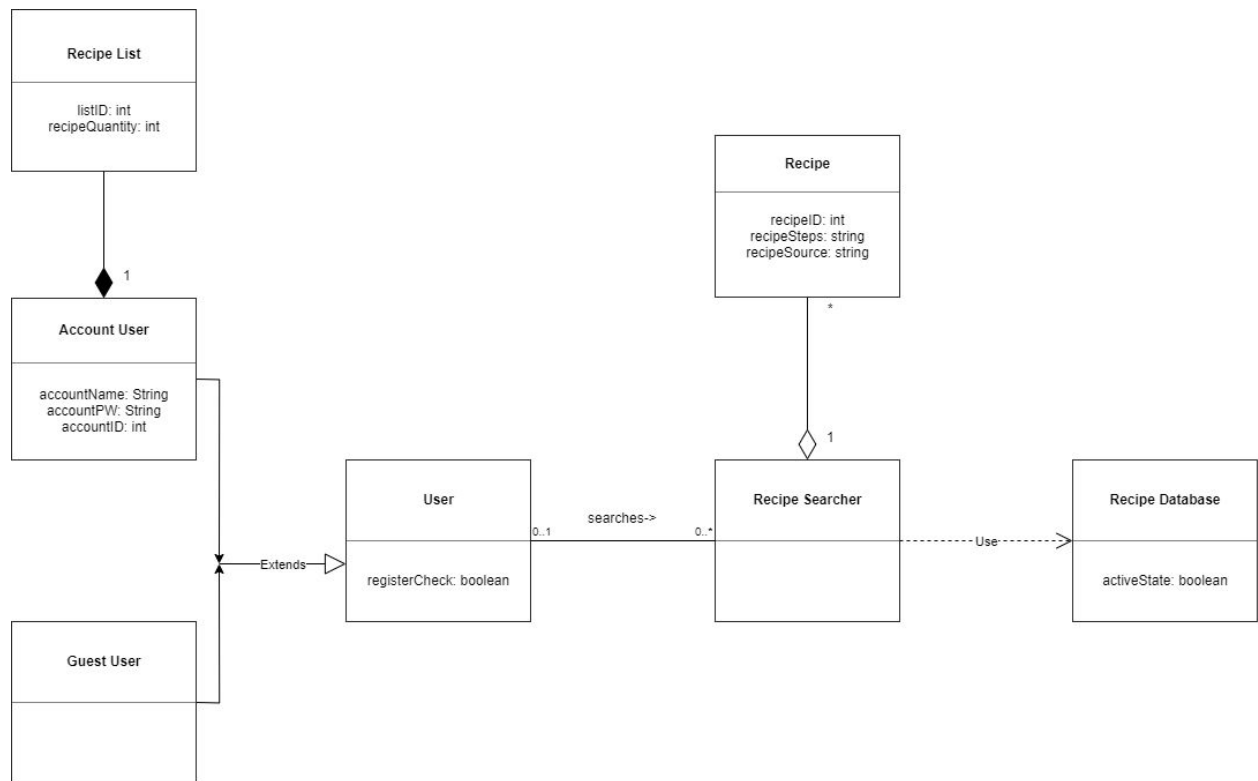
Admin - Analyze Data



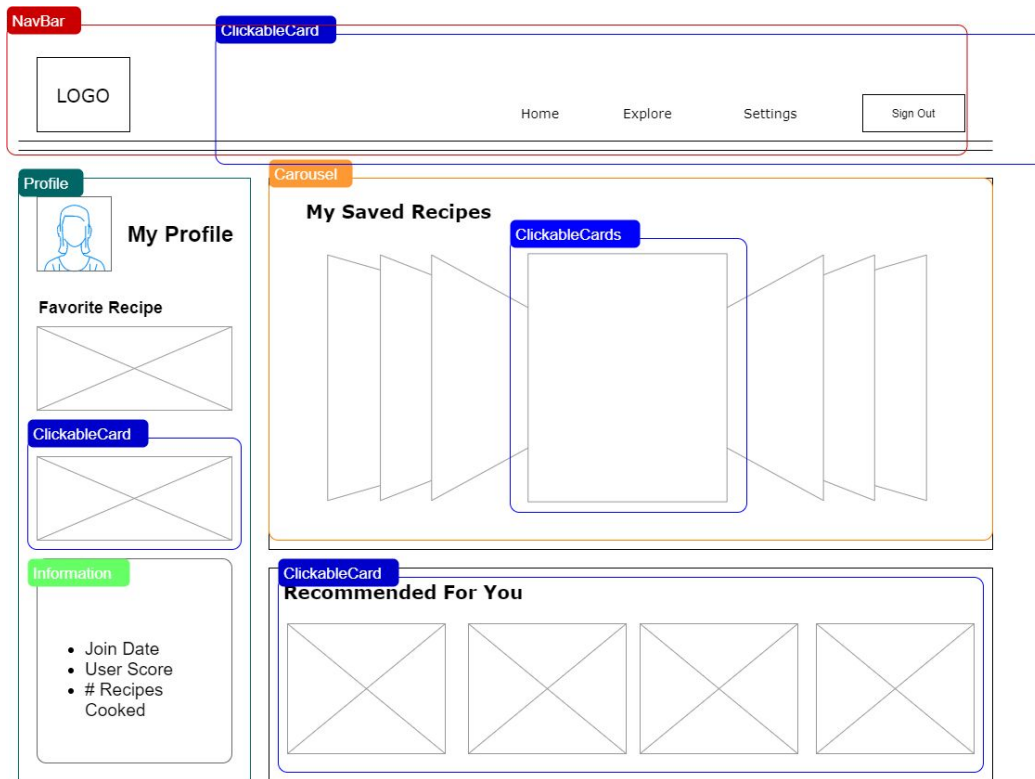
Sequence Diagrams



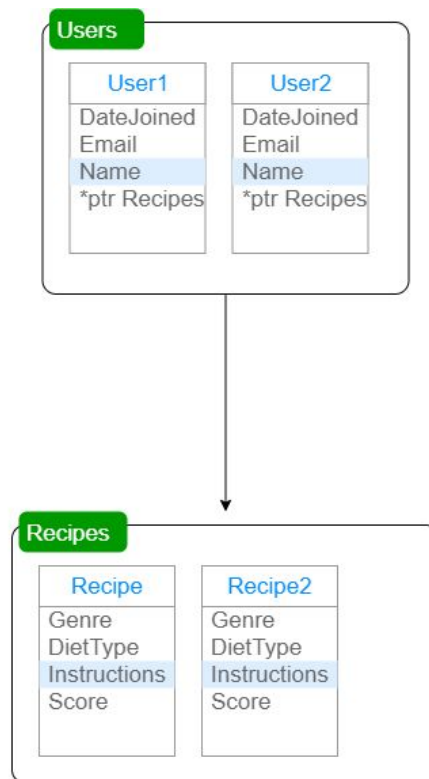
Class Diagram



Front End Design:





Storage Architecture



Recipe Explore Tab UI Example

The image shows a UI example for a "Recipe Finder" application. The interface has a teal background and a white search bar at the top with the placeholder text "Enter type of dish..." and a green "Search" button.

Below the search bar, there are two recipe cards displayed side-by-side:

- 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	
<u>Pros</u>	<u>Cons</u>
<ul style="list-style-type: none">- 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	<ul style="list-style-type: none">- 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)	
<u>Pros</u>	<u>Cons</u>
<ul style="list-style-type: none">- 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	<ul style="list-style-type: none">- 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)	
<u>Pros</u>	<u>Cons</u>
<ul style="list-style-type: none"> - High cohesion, low coupling - Easier to modify due to separation of concerns - Multiple developers can work components at the same time 	<ul style="list-style-type: none"> - 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

<div data-bbox="207 499 496 674"></div> <ul style="list-style-type: none">• 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)	<ul style="list-style-type: none">• Price Packages (Developer, Business, Enterprise)• No control over environment, dependent on Amazon
<div data-bbox="207 1031 496 1325"></div> <ul style="list-style-type: none">• Document-oriented NoSQL database for ease of access of indexing• Direct use of JSON and JavaScript frameworks• Free of cost	<ul style="list-style-type: none">• 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:** Spoonacular API
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. If we take a look at the prototype that we have it works on movies. This can be modified to have the same logical structure for a Recipe Recommendation System. For example, the input for our movie model is called “bag_of_words” which holds info related to the plot, actor/director names, and genre. If we apply the same logic to a recipe recommendation system our “bag_of_words” will hold info such as ingredients, equipment used, and the instructions of the recipe.

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/>
- viii. <https://spoonacular.com/food-api>

b. Inputs for Prototype Recommendation Model System

- i. $X = [\text{bag_of_words}]$; with respect to the model, X is a vector that holds words that it pulled out of the plot, Actor/Director names, and genre. (Movie that is chosen by user)
- ii. $Y = [\text{bag_of_words}]$; with respect to the model, Y is a vector that has the words related to plot, Actor/Director names, and genre. (This vector will hold information depending to the movie that it is comparing from the CSV file to the one that was inputted by the user)

c. Inputs for a Recipe Recommendation System

- i. $X = [\text{bag_of_words}]$; with respect to the model, X is a vector that holds words such as ingredients, equipment, and instructions. Something that can also be added could measurements for ingredients but as of right now we don't know how to incorporate the specified measurements with their ingredients. (Recipe that is chosen by user)
- ii. $Y = [\text{bag_of_words}]$; with respect to the model, Y is a vector that has the words related to ingredients, equipment, and instructions. (This vector will hold information depending to the recipe that it is comparing from the a file to the one that was inputted by the user)

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

```
recommendationModel.py 67 def recommendations(title, cosine_sim=cosine_sim):
68     # initializing the empty list of recommended movies
69     recommended_movies = []
70
71     # getting the index of the movie that matches the title
72     idx = indices[indices == title].index[0]
73
74     # creating a Series with the similarity scores in descending order
75     score_series = pd.Series(cosine_sim[idx]).sort_values(ascending=False)
76
77     # getting the indexes of the 10 most similar movies
78     top_10_indexes = list(score_series.iloc[1:11].index)
79
80     # populating the list with the titles of the best 10 matching movies
```

would match with the user. The raw inputs that are used are vectors that hold huge amounts of words that are related to the plot, actor/director names, and the genres of the movie. The inputs are then processed by the RAKE and Sklearn libraries which allows us to categorize the importance of each word in the vector. The way it inserts words into the vector is by using delimiters to separate words and be able to compare them. So if we apply the same logic of gathering data and the processing of it then we can realistically build a recipe recommendation system. Like mentioned before the only major difference would be what information is pushed through the model. Some of the setbacks is what information should be pushed and how to categorize the importance of certain words that pertain to cooking.

Raw inputs[bag_of_words]:

	bag_of_words
Title	
The Shawshank Redemption	crime drama frankdarabont timrobbins morganfr...
The Godfather	crime drama francisfordcoppola marlonbrando a...
The Godfather: Part II	crime drama francisfordcoppola alpacino rober...
The Dark Knight	action crime drama christophernolan christia...
12 Angry Men	crime drama sidneylumet martinbalsam johnfied...
Schindler's List	biography drama history stevenspielberg liam...
The Lord of the Rings: The Return of the King	adventure drama fantasy peterjackson noelapp...
Pulp Fiction	crime drama quentintarantino timroth amandapl...
Fight Club	drama davidfincher edwardnorton bradpitt meatl...
The Lord of the Rings: The Fellowship of the Ring	adventure drama fantasy peterjackson alanhow...
Forrest Gump	comedy drama romance robertzemeckis tomhanks...
Star Wars: Episode V - The Empire Strikes Back	action adventure fantasy irvinkershner markh...
Inception	action adventure sci-fi christophernolan leo...
The Lord of the Rings: The Two Towers	adventure drama fantasy peterjackson bruceal...
One Flew Over the Cuckoo's Nest	drama milosforman michaelberryman peterbrocco ...
Goodfellas	crime drama martinscorsese robertdeniro rayli...
The Matrix	action sci-fi lanawachowski,lillywachowski ke...
Star Wars: Episode IV - A New Hope	action adventure fantasy georgelucas markham...
Se7en	crime drama mystery davidfincher morganfreem...
It's a Wonderful Life	drama family fantasy frankcapra jamesstewart...
The Silence of the Lambs	crime drama thriller jonathandemme jodiefo...
The Usual Suspects	crime drama mystery bryansinger stephenbaldw...
Léon: The Professional	crime drama thriller lucbesson jeanreno gary...
Saving Private Ryan	drama war stevenspielberg tomhanks tomsizemor...
City Lights	comedy drama romance charleschaplin virginia...
Interstellar	adventure drama sci-fi christophernolan elle...
American History X	crime drama tonykaye edwardnorton edwardfurlo...
Modern Times	comedy drama family charleschaplin charlesch...
Casablanca	drama romance war michaelcurtiz humphreyboga...
The Green Mile	crime drama fantasy frankdarabont tomhanks d...

Dataset: <https://query.data.world/s/uikepcpffyo2nhig52xxeevdialfl7>

4. **Training Set/Test Set**

The beauty of our system is that we don't necessarily have to feed it a training set. Our recommendation model works on a scoring system. This scoring system that determines the similarity between movies(or recipes) is Cosine Similarity. So our model can just be fed a database with recipes and be compared to the recipe a user selects. As seen from the example of our movie recommendation model we can see that our model just iterates through the CSV file and determines its scoring matrix.


```

(0, 2174) 1
(0, 888) 1
(0, 1899) 1
(0, 2481) 1
(0, 969) 1
(0, 2950) 1
(0, 311) 1
(0, 1733) 1
(0, 1269) 1
(0, 2765) 1
(0, 59) 1
(0, 655) 1
(0, 519) 1
(0, 306) 1
(0, 1810) 1
(0, 2678) 1
(0, 1011) 1
(0, 768) 1
(0, 584) 1
(1, 1972) 1
(1, 82) 1
(1, 2492) 1
(1, 2193) 1
(1, 846) 1
(1, 475) 1
: :
(248, 1648) 1
(248, 352) 1
(248, 1920) 1
(248, 2188) 1
(248, 1645) 1
(248, 1462) 1
(248, 290) 1
(248, 768) 1
(249, 2869) 1
(249, 2825) 1
(249, 2801) 1
(249, 2176) 1
(249, 1828) 1
(249, 443) 1
(249, 2472) 1
(249, 146) 1
(249, 2340) 1
(249, 705) 1
(249, 1615) 1
(249, 1278) 1
(249, 1779) 1
(249, 621) 1
(249, 50) 1
(249, 2626) 1
(249, 768) 1
[[1. 0.15789474 0.13764944 ... 0.05263158 0.05263158 0.05564149]
 [0.15789474 1. 0.36706517 ... 0.05263158 0.05263158 0.05564149]
 [0.13764944 0.36706517 1. ... 0.04588315 0.04588315 0.04850713]
 ...
 [0.05263158 0.05263158 0.04588315 ... 1. 0.05263158 0.05564149]
 [0.05263158 0.05263158 0.04588315 ... 0.05263158 1. 0.05564149]
 [0.05564149 0.05564149 0.04850713 ... 0.05564149 0.05564149 1. ]]

```

5. 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.

6. Training the Model

As stated in our discussion regarding Training Data and Test Set, we determine that the model doesn't necessarily need to be trained. All the data that is needed should be in a CSV file that can be easily interpreted by the model. The way the model determines how close in similarity the recipes are is by applying the cosine similarity on both vectors. Unlike other Machine Learning Models we don't need to shove huge amounts of data so that our model can be tested.

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.

7. Evaluating Results

```
(0, 2174) 1
(0, 888) 1
(0, 1899) 1
(0, 2481) 1
(0, 969) 1
(0, 2950) 1
(0, 311) 1
(0, 1733) 1
(0, 1269) 1
(0, 2765) 1
(0, 59) 1
(0, 655) 1
(0, 519) 1
(0, 306) 1
(0, 1810) 1
(0, 2678) 1
(0, 1011) 1
(0, 768) 1
(0, 584) 1
(1, 1972) 1
(1, 82) 1
(1, 2492) 1
(1, 2193) 1
(1, 846) 1
(1, 475) 1
:
(248, 1648) 1
(248, 352) 1
(248, 1920) 1
(248, 2188) 1
(248, 1645) 1
(248, 1462) 1
(248, 290) 1
(248, 768) 1
(249, 2869) 1
(249, 2825) 1
(249, 2801) 1
(249, 2176) 1
(249, 1828) 1
(249, 443) 1
(249, 2472) 1
(249, 146) 1
(249, 2340) 1
(249, 705) 1
(249, 1615) 1
(249, 1278) 1
(249, 1779) 1
(249, 621) 1
(249, 50) 1
(249, 2626) 1
(249, 768) 1
[[1. 0.15789474 0.13764944 ... 0.05263158 0.05263158 0.05564149]
 [0.15789474 1. 0.36706517 ... 0.05263158 0.05263158 0.05564149]
 [0.13764944 0.36706517 1. ... 0.04588315 0.04588315 0.04850713]
 ...
 [0.05263158 0.05263158 0.04588315 ... 1. 0.05263158 0.05564149]
 [0.05263158 0.05263158 0.04588315 ... 0.05263158 1. 0.05564149]
 [0.05564149 0.05564149 0.04850713 ... 0.05564149 0.05564149 1.]]
```

These are our results when the model has finished running. This allows us to see the matrix for each movie and their similarity values.

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

    # getting 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
```

```
def main():
    userInput = input("Please enter a movie for recommendations: ")

    for i in recommendations(userInput):
        print("Movie: " + i)

    return 0

if __name__ == '__main__':
    main()
```

```
Please enter a movie for recommendations: The Godfather
Movie: The Godfather: Part II
Movie: Scarface
Movie: Fargo
Movie: Rope
Movie: On the Waterfront
Movie: Goodfellas
Movie: Cool Hand Luke
Movie: Baby Driver
Movie: Casino
Movie: A Clockwork Orange
```

```
103    1.000000
181    0.189525
90     0.189525
77     0.166091
165    0.161627
202    0.151620
145    0.145671
130    0.121566
242    0.118771
133    0.118771
73     0.113715
131    0.101710
37     0.098480
98     0.095893
```

The first result that is shown is the similarity score between The Godfather and The Godfather. The max score you can get from Cosine Similarity is 1. The next closest result is 0.189525 which represents The Godfather II. The same thought process would be applied to the recipe recommendation system.

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 occurrences 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.

8. **Predictions**

A few predictions we have when it comes to a recipe recommendation system is the complexity of retrieving the proper data either from an online recipe or from our API recipe calls. With the help of Alex's NLP Scraper we should be able to get all necessary information from said recipes. The accuracy of the recommendation model will heavily rely on the accuracy of the NLP model.

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.

9. **Deployment and Reuse**

Deploying our model wouldn't be too difficult, rather like mentioned earlier the difficulty will come with what information is pushed through the recommendation model. Our current model is a prototype of a movie recommendation system. This is a prime example of how we can reuse this prototype to work on a recipe recommendation system. The reuse of this prototype should be rather easy to apply on other areas such as video game recommendations or sporting recommendations.

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.

Technological Depth and Innovation - Natural Language Processing Model

The problem that we are trying to solve with this application is that people want to cook using recipes but the authors of the recipes tend to have a lot of text that people don't care about. To solve this problem, we are creating a recipe finder.

1. Finding data

The process of finding data is the toughest part of this solution. A python script was created for easy input of data. The data that we are looking for is the number of occurrences of a word in a recipe instruction. We only want recipe instructions and not just recipe sentences because the instructions are what we are trying to find. For example, let's take the sentence, "put the fish in the oven." The script will first take out any punctuation, in this case, there are none. Next, it will split by white spaces. Every word has its dictionary position with the number of occurrences next to it. In this case, it would be "put : 1", "the : 1", etc. A section of our datasheet looks like this:

```
1  bring 113
2  water 273
3  to 1402
4  boil 123
5  and 2074
6  add 603
7  the 2916
8  thai 36
9  tea 89
10 mix 114
11 sugar 141
12 gently 69
13 stir 205
14 completely 50
15 dissolve 10
16 for 539
17 about 277
18 3 112
19 minutes 557
```

To keep things consistent, we kept a record of all the foods that we looked up so that others in the group would not look over it again. In addition to this, we limited the number of recipes per food to 5. With these constraints in place, all of the data should be consistent.

2. Refining data

A big problem that you run across when doing this method, is a lot of random words as well as unwanted words. First, to solve this problem, we created a script that finds all the non-English words and takes them out of the datasheet. The problem with this method is that non-English words are common among English users like dijon in dijon mustard.

However, this problem is ignored and taken out of the dictionary anyways. Another important step is to take out words that although appear in the recipe instruction, are common in other sentences too. Pronouns, conjunctions, articles, and more were taken out. The last set of data that was taken out was biased opinions. We looked through the data line by line and considered if these words are of any importance to a cooking instruction along with words that only appear in the dictionary two times or less. With all of these words out, our data was a lot smaller to work with.

3. Creating a tier list

Another important feature of recipe instruction besides word occurrences is the type of word occurrences. One of the most common structures of instruction is that it starts with a verb such as "SET the oven to 350 degrees." With this in mind, another datasheet was constructed with only verbs. To do this, a simple good search for recipe verbs was conducted. Another method was to watch cooking videos and record what they said in case the google search was not clear enough. The tier below the verb is nouns. The same method of creating a datasheet is the same as the verb. In the end, we were able to create two datasheets.

4. Scoring sentences

The last part of this recipe finder is the algorithm itself to find recipes. The first step is to gather data. The data that we were looking for are sentences, however, we did not have enough time to complete this step so working with a couple of sentences is all we had. To create the scoring itself, we have to take in all of the past steps. A sentence is inputted into the script. Next, the script takes out the punctuation and splits it by whitespaces just like the input script. After this, the word that appears in the input sentence is compared against the entire datasheet. If the current word that is being compared is "cook", then the script will look into the datasheet and take out the score that is associated with it, which would be 274. This is done to all the words in the sentence. At the same time, it is also checking the word to see if it is a cooking verb or cooking noun. If it is a cooking verb, its score will be multiplied by two, and if a noun, it will multiply by 1.5. In this case, the word "cook" will have a score of 548. Once each word has its score, the numbers are added together and divided by the number of words in that sentence to get an average score. If the score is over 20, then it is considered a recipe sentence. At the moment, we are unable to distinguish between a cooking instruction from a regular cooking sentence. If we had more data, we would accomplish this. The code for the scoring looks like this:

```
for word in paragraph_list:
    word_score = 0
    if (word in master_data_dic):
        word_score = int(master_data_dic[word])
    else:
        word_score = 1
    if (word in verb_list):
        word_score = word_score * 2
    if (word in noun_list):
        word_score = word_score * 1.5
    current_score += word_score

score = current_score / total_words
```

Combine garlic, butter, and oil in a microwave safe dish or in a small saucepan. Heat garlic and butter and oil in microwave for 1 minute or in a small pot over moderate-low heat for 3 minutes.

@
97.91891891891892
YES

Please select the amount of RAM you would like to download. One cannot simply Download too much RAM.

@
9.333333333333334
NO