# Architecture and Design Document

## Table of Contents


**Diagrams** **2**

System Diagrams 2

Component Diagrams 3

Use Case Diagrams 4

Activity Diagrams 5

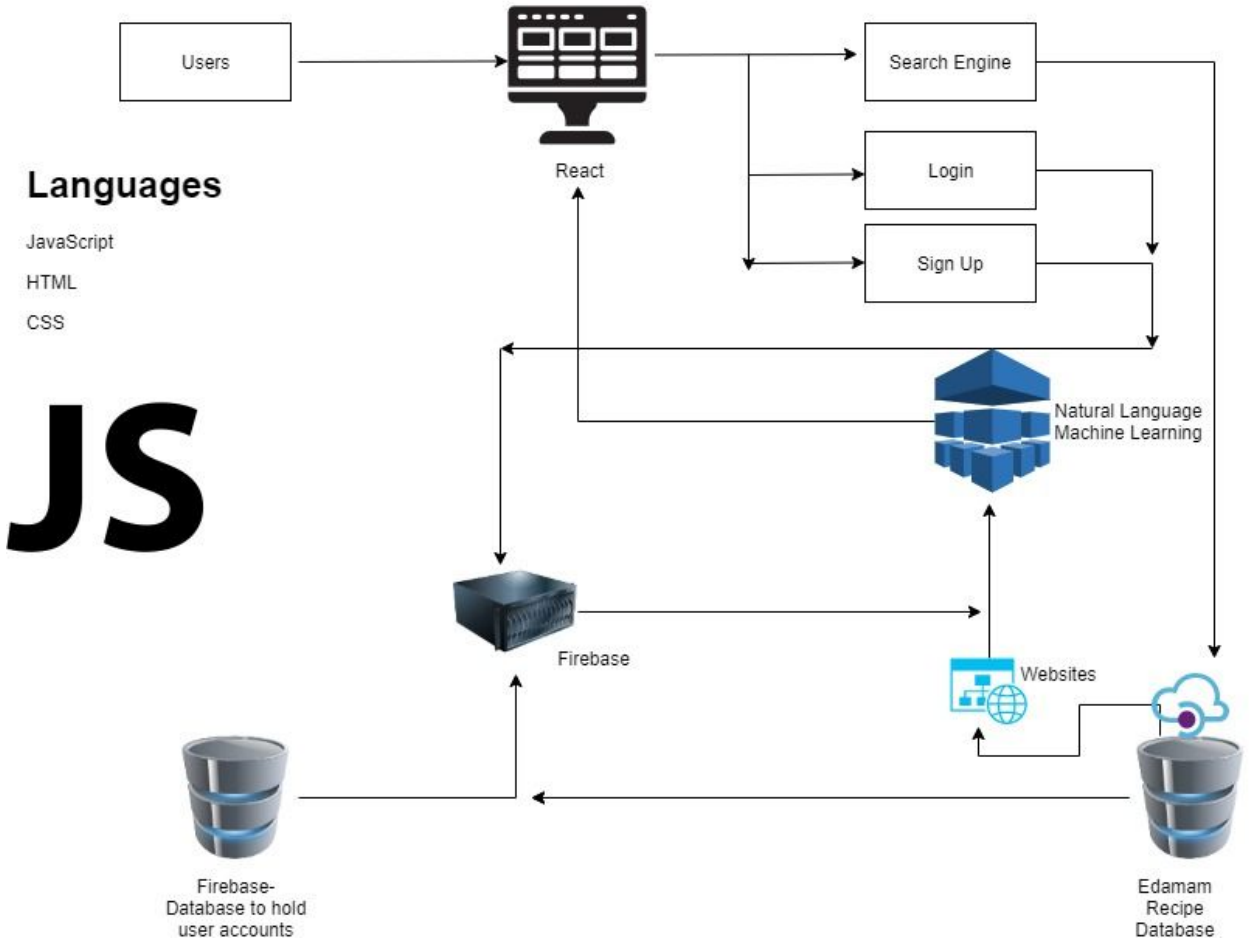
Sequence Diagrams 6

**Trade-Off Analysis** **7**

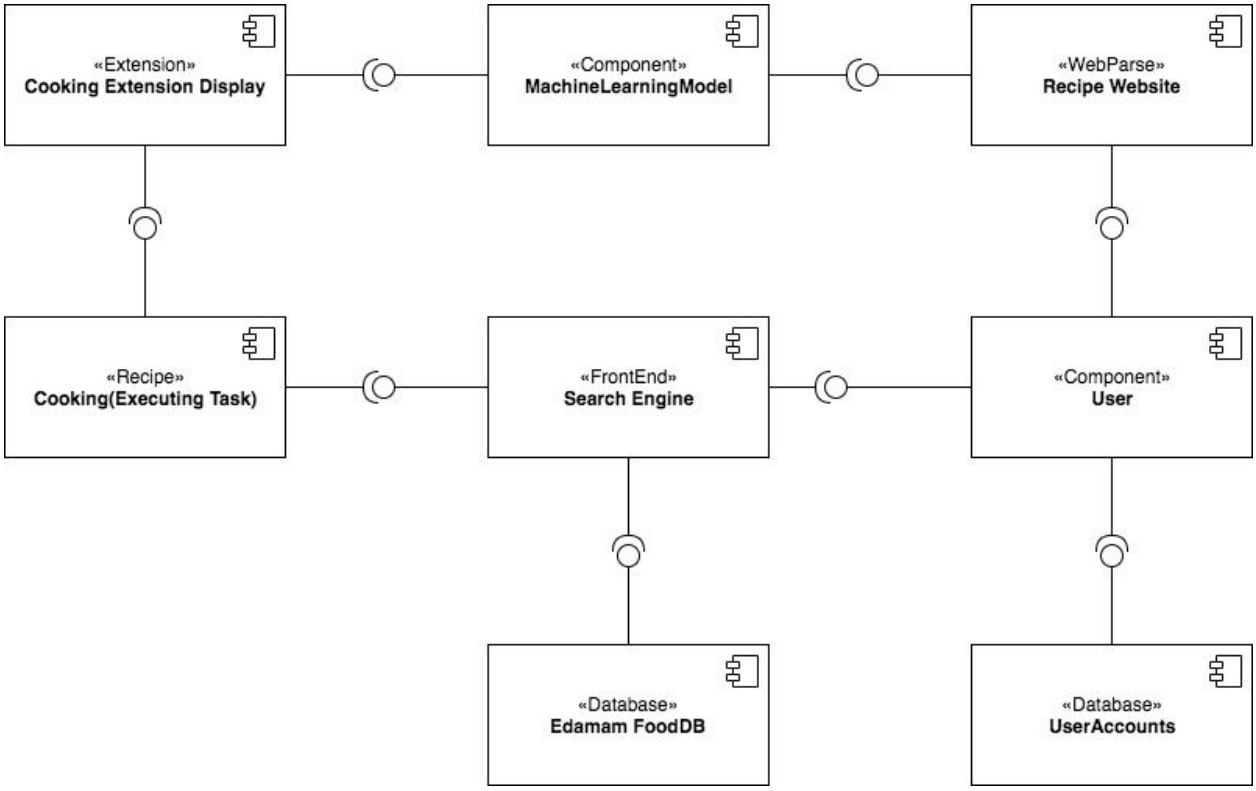
**Machine Learning Analysis** **12**

# Diagrams

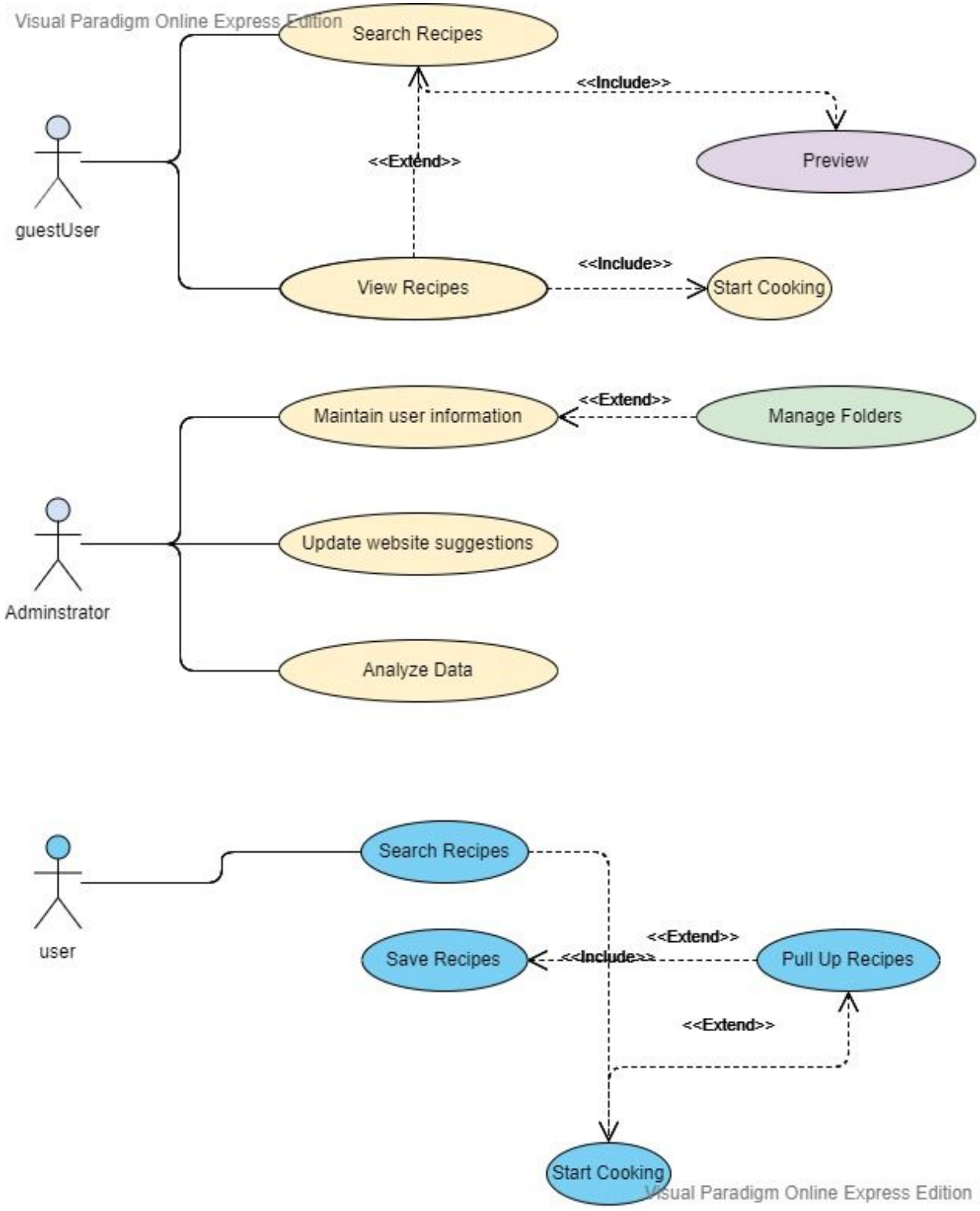
## System Diagrams



# Component Diagrams

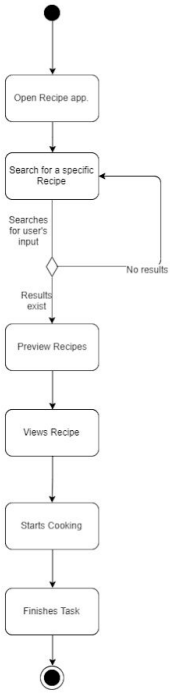


# Use Case Diagrams

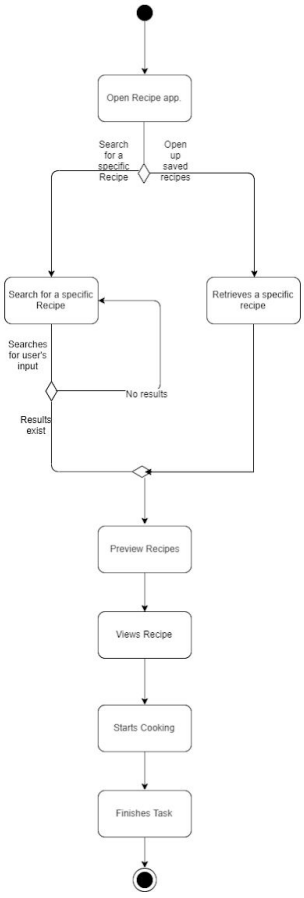


# Activity Diagrams

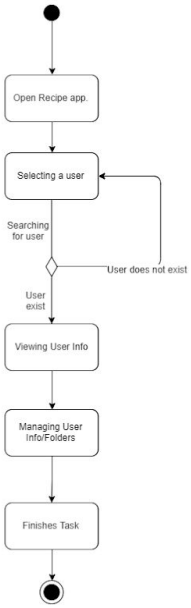
Guest User - Searches Recipe



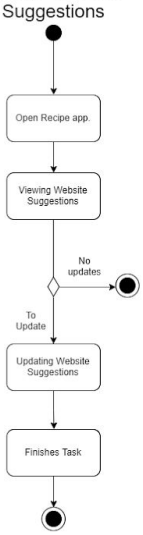

User - Searches Recipe



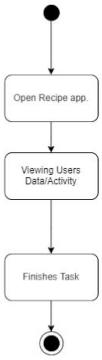

Admin - Maintain User Information



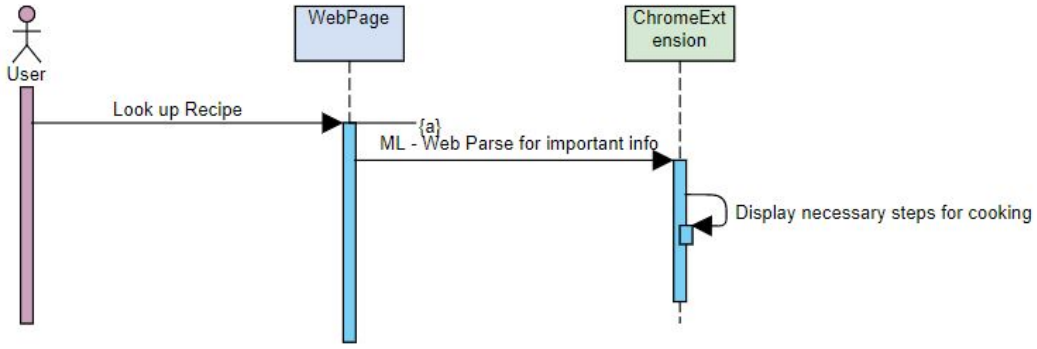

Admin - Update Website Suggestions

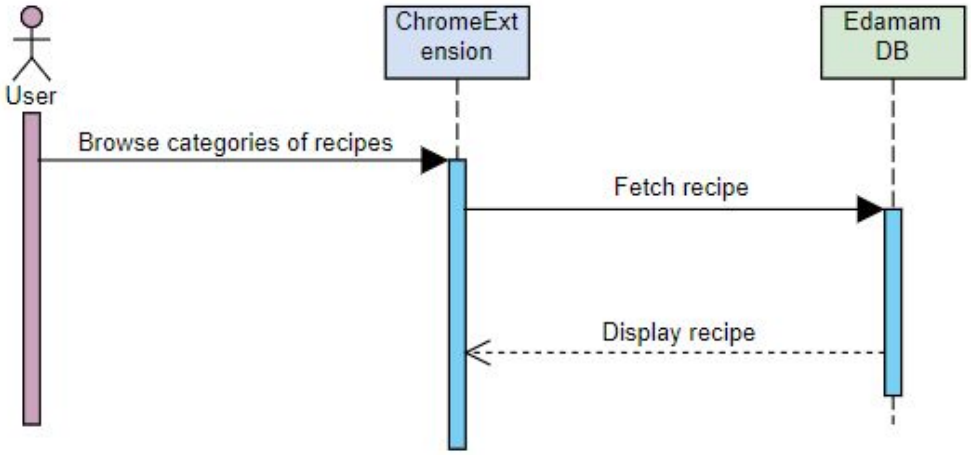



Admin - Analyze Data



# Sequence Diagrams



# 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> |
| - Better data sharing since data is all on a single server<br>- Easier maintenance and better security control access<br>- Resources are shared across different platforms<br>- Ability to log into the system despite location or technology of the processor<br>- 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<br>- If server fails, no user can use the application until servers are fixed |

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

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

# Trade-Off Analysis : Front-end

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

# Trade-Off Analysis : Back-end

| Pros | Cons |
| --- | --- |
| <br>• No data capacity - you pay for the amount you use on the cloud<br>• Very detailed documentation for ease of learning<br>• Array of tools ready to be deployed (ex. Database - Amazon Aurora, analytics) | • Price Packages (Developer, Business, Entreprise)<br>• No control over environment, dependent on Amazon |
| <br>• Document-oriented NoSQL database for ease of access of indexing<br>• Direct use of JSON and JavaScript frameworks<br>• Free of cost | • Less flexibility in queries (ex. No joins)<br>• 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 fit 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

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. Later on, we slightly altered the focus of the machine learning analysis into a model in which parses strings from web pages and generate a recommendation based on the strings. 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.