University of Arkansas – CSCE Department
Capstone I – Final Proposal – Fall 2020

FooDecisive: A Choice for You

Aneesh Komanduri, Adam Greenfield, Huy Mai, Tay Pham, Zhi Han Weng

Abstract

A significant flaw with people is that we tend to be incredibly indecisive. Food is one example. It may seem trivial, but choosing what to eat stumps more people than you may think. For instance, you are going out with a friend for dinner or on a date, and you are too nervous about what makes the other happy, so you can't think of a good option for both of you. Not knowing what to choose can lead you to an awkward scenario where you may be unhappy with your meal. As computer science students, we recognize this problem. We create FooDecisive to help save your time and make better decisions by effectively recommending a list of restaurants based on your preferences. One key feature in this app that will allow users to make better and quicker decisions is implementing a convincing display of recommended restaurants in a list of organized preferences and specify directions to restaurants on a Google Map. FooDecisive saves and learns users' search history and preferences to make a better recommender. To build FooDecisive, we use technologies such as ReactJS for frontend, PostgreSQL for database management system, and Python Flask for the backend. To construct the recommender system, we use distributed computing APIs such as PySpark on a Databricks notebook to perform machine learning and collaborative filtering for providing personalized food recommendations. Additionally, we implemented a conversational chatbot to boost user experience. The Yelp API is used to obtain information regarding restaurant information such as locations, reviews, ratings, genres, etc.

1.0 Problem

People tend to have difficulties when facing what to eat, where it could be deciding between different cuisines or deciding between a long list of items on the menu. This problem that people experience is called choice overload, where people have trouble deciding when faced with a large number of options [10]. Our proposed idea, FooDecisive, tries to help the users reduce the impact of this problem.

According to the US Bureau of Labor Statistics, the time people spend on eating and drinking, and food preparation and cleanups are around an average of two hours per day [11]. With this in mind, we should not spend more time than necessary when deciding what to eat, other than actually eating or preparing. All that time could be spent on other activities, such as doing homework if you are a student or caring for others in the household if you are an adult.

As mentioned above, people's time would be wasted just on the problem of deciding what to eat. If the problem is not solved, people will spend an additional thirty minutes to an hour to decide what to eat, whereas on the other hand if the decision were already made, they could finish eating and all of the other activities related in around two hours. Even though that additional thirty minutes to an hour does not sound like much, the time would accumulate. According to a news article from the New York Post, the "average American couple spends 132 hours a year deciding what to eat [12]."

2.0 Objective

This project aims to create an application that effectively provides a list of recommended restaurants to users based on their preferences. The application centers around a recommending system that builds on existing work to consider additional user preferences and display improved performance. The list of recommendations is complemented with a map containing pinned restaurants to help users consider distance. In addition, to remedy the muddling of factors when considering a restaurant to dine at, the application gives users the option to view and save multiple lists as a way of organizing groups of preferences. This feature requires users to log into accounts to save the lists under their accounts.

3.0 Background

3.1 Key Concepts

Frontend

ReactJS [1]: React is a framework of JavaScript that was designed by Facebook. It is a declarative interface for building interactive UI's. Similar to other frameworks of JavaScript, React has a component-based architecture, where each feature can be modularized into a component. These components can then be used and reused wherever appropriate. React is a stateful framework, which means that it can keep track of the state of specific attributes. So, whenever an attribute changes, the render() method in React will automatically change the state of the attribute in the Document Object Model (DOM). Another benefit of React is that it follows a concept known as 'Virtual DOM'. This means that the DOM of a webpage does not need to refresh to update changes. React only detects the changes that have been made, compares these

changes to the previous state that is stored and only changes those elements on the DOM that are different. This allows React components to run efficiently on the webpage without redundant changes. We want a user-friendly UI for our FooDecisive application, so React makes for a great choice. There will be features in the app that are similar, so modularity is key, which React provides.

Database

PostgreSQL [2]: PostgreSQL is a powerful, open source object-relational database system with over 30 years of active development that has earned it a strong reputation for reliability, feature robustness, and performance. Postgres is optimal for scalable applications where tables in the database may contain millions of rows. As such, we decided to use postgres, since realistically if we wanted to scale our application to millions of users, we can do so with minimal cost and optimal performance.

Backend

Flask [3]: Flask is a lightweight microframework of Python that is often used for web development. It has a Jinja template engine that creates HTML templates that are returned to the webpage. Flask keeps things simple and in that it is very flexible. For FooDecisive, we want to use Flask to develop the backend of the application. We will use Flask to build REST API's in the backend. Security is important when creating a web application that stores data. We want to keep user data secure. Fortunately, Flask protects against one of the most common security problems: cross-site scripting (XSS). So, authentication will be made easy with Flask. Additionally, since Flask is a framework of Python and most machine learning development is done with Python due to the plethora of libraries, developing machine learning models and returning data to the webpage will be very convenient.

Distributive Computing

Databricks [15]: The Databricks platform is a simple platform to store and manage all of your data for analytics workloads. For the purposes of this project, a community version of Databricks will be used to train our recommender system. Databricks requires users to create clusters to run a Jupyter Notebook on. This allows a distributive computing approach that can handle large amounts of data.

Apache Spark [4]: We will be utilizing this popular distributed computing framework to train our recommender system model. Apache Spark is a unified analytics engine for large-scale data processing. To process the user ratings data, we will utilize an API in python called PySpark. We will establish a JDBC connection to databricks and feed in our ratings table to be processed using

PySpark. Once the model is trained and the predictions are calculated, the results (in a pyspark dataframe) are committed back to the postgres database via JDBC.

Concepts

Collaborative Filtering [5]: Collaborative filtering uses similarities between users AND items simultaneously to provide recommendations. It is a much better alternative to Content-based filtering, which strictly uses historical item preference of the given user to recommend new items. This doesn't take into account how users may be alike and could have similar taste. Collaborative Filtering models can recommend an item to user A based on the interests of a similar user B. Each user and item get embeddings in a lower dimensional space for comparison. This is done by methods such as Matrix Factorizations to generate embeddings. The matrix for the user and item, respectively, embed the features of the user or item. The two embedding matrices are initially randomly initialized and updated after every iteration of optimization based on the loss so that the elements of the matrices represent accurate embeddings for users and items. This allows us to get a compact representation of users and their preferences and can allow us to predict that a given user would like a given item and ultimately recommend the top preferred items to a given user.

External APIs

Yelp API [6]: Yelp provides API's that developers can use to build their applications while integrating Yelp features and real-time data. For FooDecisive, we use the Yelp Fusion REST API and the Yelp Open Dataset for user reviews and restaurants. The Dataset will be used to develop the aforementioned recommender system and the REST API will be used to retrieve information regarding restaurants such as location, reviews, ratings, genre, etc.

Google Maps API [16]: The Google Maps API is a service offered by Google to developers who want to embed the Google Maps Platform into their applications. For our application, we would like to show users a Google Map to show them directions to the specific restaurant that a user is viewing. The API allows embedding the maps into a UI and even allows location services via the Geocoding API so that users can view restaurants in their region.

WitAI API [17]: To increase interactivity with users, we will be creating a conversational chatbot. The chatbot will be responsible for understanding user intents and based on queries, extract information that can be fed into the Yelp Fusion API to search for restaurants that the user would like to view. Wit AI can be used to build natural language experiences and chatbots seamlessly. Wit AI also provides an HTTP API to send queries where a response is generated

and sent back to the client. We will leverage this to send user queries to the API and use natural language generation to allow the chatbot to respond to the user.

Cloud Services Used

Heroku: Frontend (React) and Backend (Flask)

Amazon Relational Database Service (RDS): PostgreSQL

3.2 Related Work

Many people struggle to decide where to eat. As a result, there are several restaurant/food recommendation systems that aim to solve this problem. One basic implementation is known as the Wheel of Dinner [7]. This system does not recommend specific restaurants or take into account the user's location. It is merely a wheel that contains different types of food (e.g., Mexican, pizza, etc.) and allows the user to spin the wheel, landing on a random spot. This system is intended to be a quick and easy method to help people decide what to eat. To that end, the Wheel of Dinner is relatively effective. However, we can improve on this by taking into account the user's preferences and location, which will allow us to recommend actual restaurants near the user that they will have a higher probability of enjoying.

Perhaps the most popular recommendation system is Yelp [8]. Yelp allows users to review not only restaurants but also tradesmen (e.g. plumbers, mechanics) and a variety of other services. The website then recommends these restaurants/tradespeople/services based on users' reviews. Yelp is an effective service and is widely used for a reason, but the filtering can be improved. The user's preferences can be used to predict the rating a user would give to a restaurant, even if they have never visited the restaurant before. A more robust recommendation system can be built based on these predicted ratings, combined with community reviews.

Another notably similar system is TripAdvisor [9]. Like Yelp, restaurant recommendations are just one part of what TripAdvisor does. The website allows the user to enter a variety of preferences. However, there are some potentially useful preferences that are missing (e.g., distance). Again, we can use the user's preferences to predict their unvisited restaurants' ratings, which will result in more accurate recommendations.

The overall goal of FooDecisive is to build a recommendation system that is more accurate than similar implementations. We can take collaborative filtering a step further by asking the user the right questions and processing their answers and community reviews to recommend restaurants that they have the highest probability of enjoying. This will result in higher customer satisfaction and retention.

4.0 Design

4.1 Requirements and/or Use Cases and/or Design Goals

Requirements

- Register/Login Page and authenticate credentials
- Provide a search bar to search restaurants
- User should be able to enter their preferences
- Provide a list of top-k recommended restaurants personalized to user
- Should be able to view location, ratings, genre, contact info, covid-19 protocols
- Integrate Google Maps to provide directions to a given restaurant (shortest path)
- Integrate Chatbot feature to request for restaurant information

Original Design

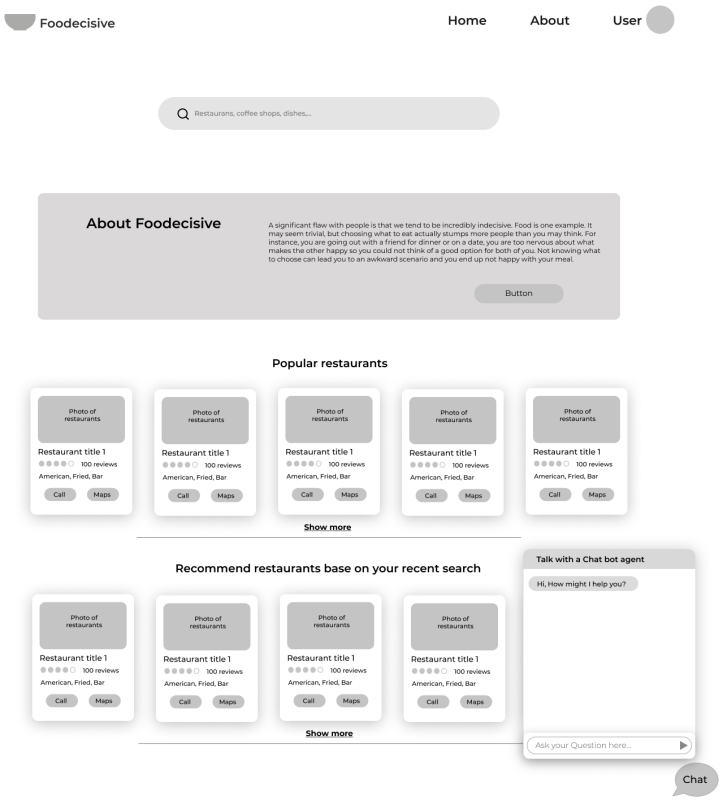

Figure 1: Original Home page design

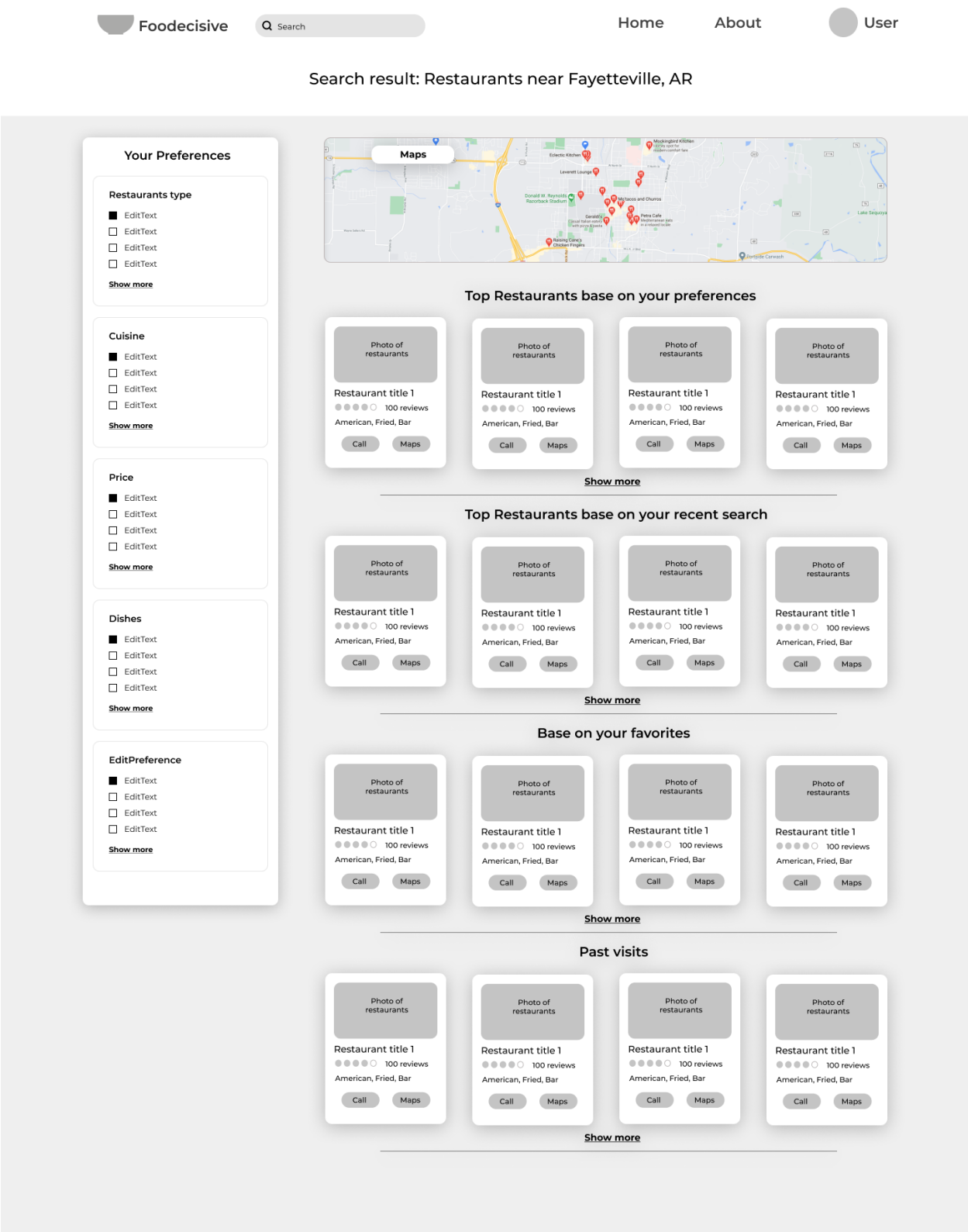

Figure 2: Original Restaurant Details page

4.2 New Detailed Architecture

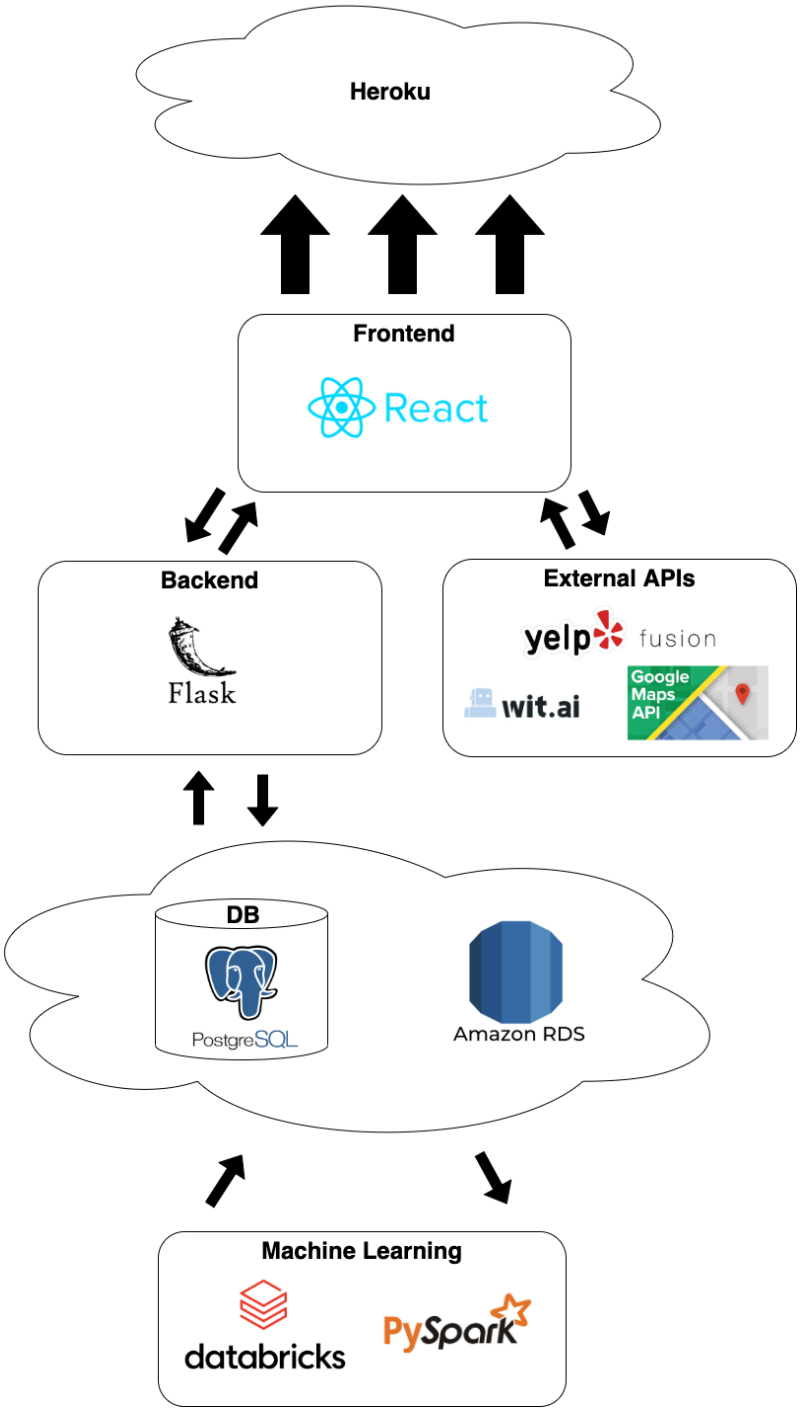

Figure 3: Overall High Level Design

Home Page

Homepage is an important part of our application because it shows an introduction to the application. We decided to design our homepage in a way to guide our user through all the features of Foodecisive. It first shows a welcome message and brief description to introduce our application. There is an option for users to login and signup. Then, we introduce all our main features by showing a brief description and a photo of the UI including search, Chatbot, Rate and Favorite and Personalized Recommendation. Foodecisive's homepage is built using a React UI framework Material-UI. From the homepage, users can navigate to other features by clicking a button on each feature.

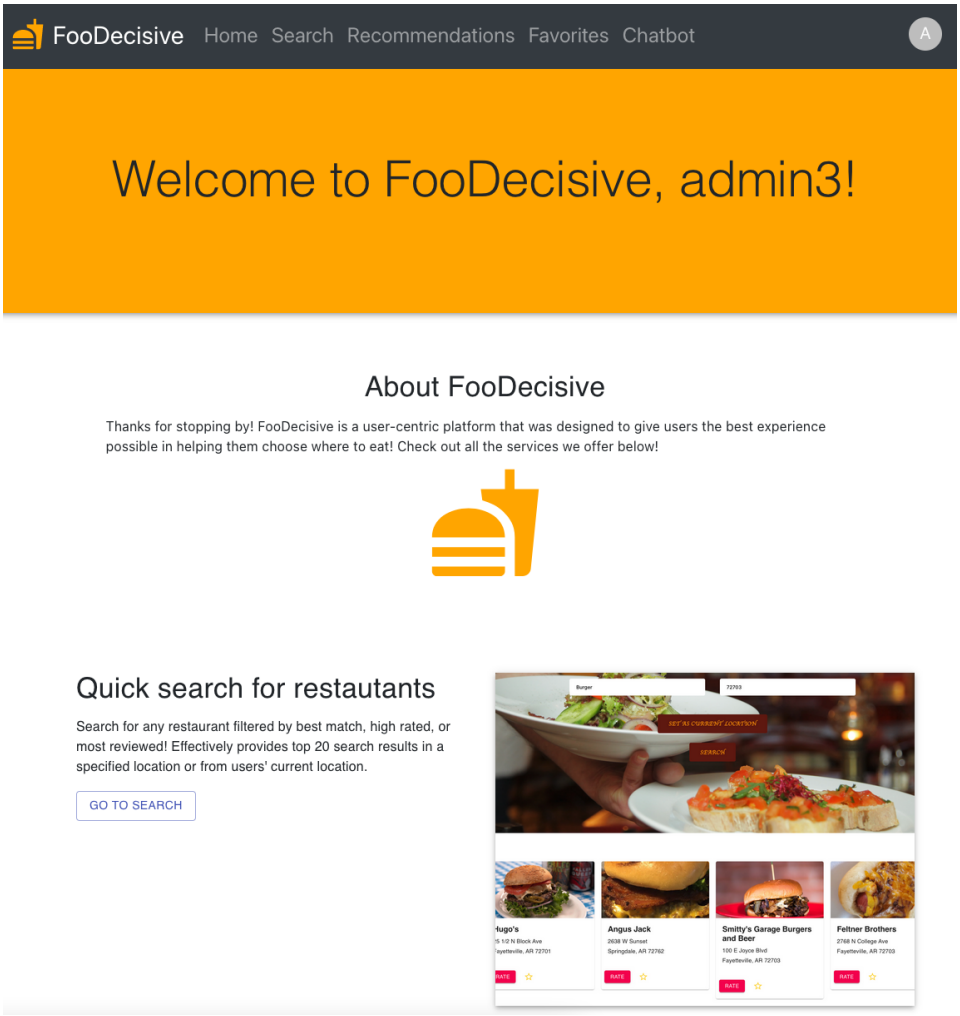

Figure 4: Home page

Login and Registration Pages

Before the user is logged in, the only viewable pages are the home page, the search page, the login page, and the register page. The login and register pages look nearly identical. They consist of a simple form with two text inputs: a username and a password. If the user does not already have an account with FooDecisive, they would first visit the register page. After the user enters a username and password, which must be at least 8 characters and contain an uppercase and a special character, they can press "Submit" and the entered information will be persisted to the database. Once the user's username and password are successfully stored in the database, they can then log in to this account from the login page.

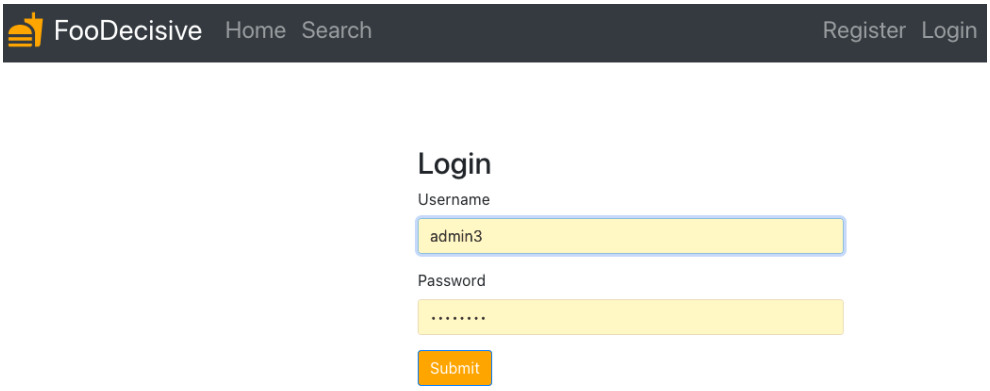

Figure 5: Login and Register page

Search Functionality

The user, with or without an account, can search up restaurants in the Search page. In order for the search functionality to work, the user is required to enter both a search term (e.g. Asian, tacos, fast food, etc.) and a location, which can be a city name or a specific address. Using Yelp Fusion REST API, 20 search results are obtained for each search based on one of three factors: Best Match, which matches restaurants with search terms specified by the user, Highly Rated, where the user views a list of top rated restaurants based on specified search terms, and Most Reviewed, where the list of search results are sorted based on the number of reviews given by users as reflected in the Yelp Fusion REST API. We note that these three factors are modes provided by the API to sort the search results.

One other feature to highlight with regards to the search functionality is the option to set a user's current location as the location entry for the search function. We include this in order for the user to conveniently view restaurants that are near his or her location, which is a common choice to consider when looking for restaurants to order food from.

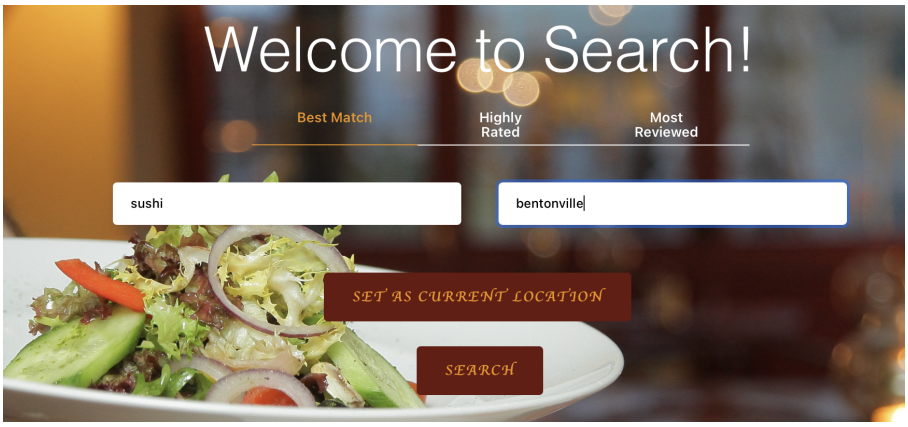

Figure 6: Example search entry

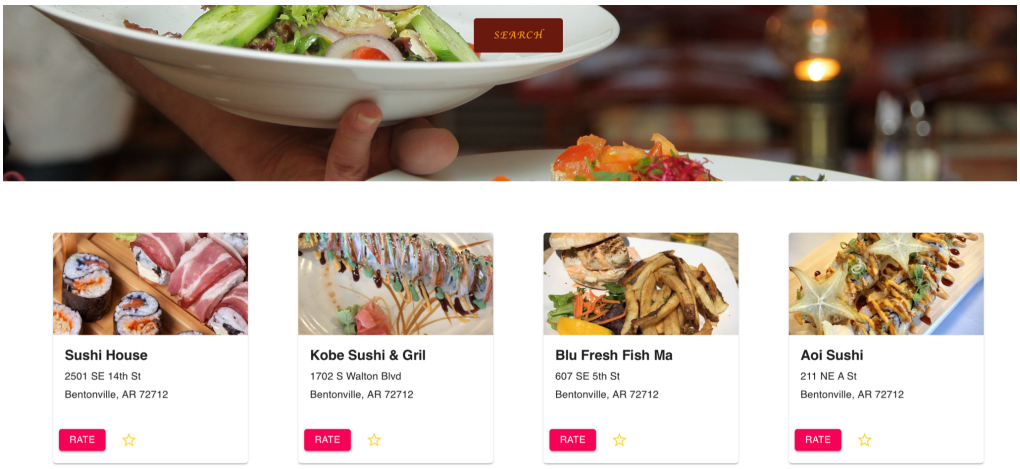

Figure 7: Search results

Profile Page

With an account, the user has access to his or her Profile page (shown in Figure 7), which primarily consists of an avatar containing the first letter of the user's username and a review history section. The review history lists every review form that is submitted, whether through the Search page or the Favorites page, where the user views the following for each review: the title of the restaurant that the user reviewed, the rating given by the user towards the restaurant, which is displayed in stars using React's Material UI library, and the user's comment about the restaurant.

Moreover, the user can perform two actions pertaining to each review. First, the user can edit the review, which takes the user to the original form that he or she filled out. In order to confirm to the user that any changes are made to the form, a condition on the activation of the submit button is added. Once the user makes changes, the form window is closed, and the review history is updated with the edited review as the last entry in the list. Second, the user can delete the review,

which takes the user to a Material UI pop-up that asks whether or not the user is sure about deleting the review. The user confirms the deletion by pressing the YES button in the pop-up, causing the review history to update with the deleted review no longer on the page.

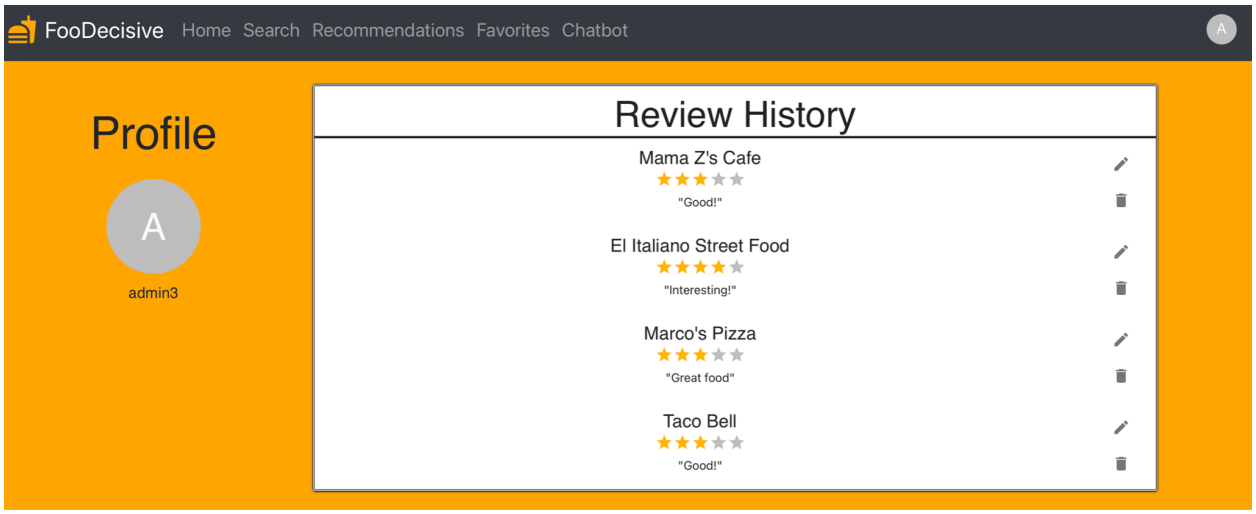

Figure 8: Profile page

Restaurant Detail Popup and Google Maps API

In FooDecisive, one of the useful components is the Restaurant Detail card because this is where the user can see all the information about the restaurant such as restaurant address, rating and location. This is a pop up window that shows all of this information when the user clicks on the restaurant card. In order to do this, we use react library Material UI to build the UI and Yelp Fusion API to retrieve restaurant detail information such as address, rating, and a photo of the restaurant. We also connected it to the backend API to send and retrieve user reviews and favorites to and from our database, which utilizes PostgreSQL. In order to show the location of the restaurant, we include a map that shows the location of the restaurant using Google Maps Javascript API. Google Maps Javascript API uses the latitude and longitude from the restaurant information from the Yelp Fusion data to put the marker on the restaurant's location. The Restaurant Detail window also shows the option to add the restaurant as a favorite and give the restaurant a rating and review.

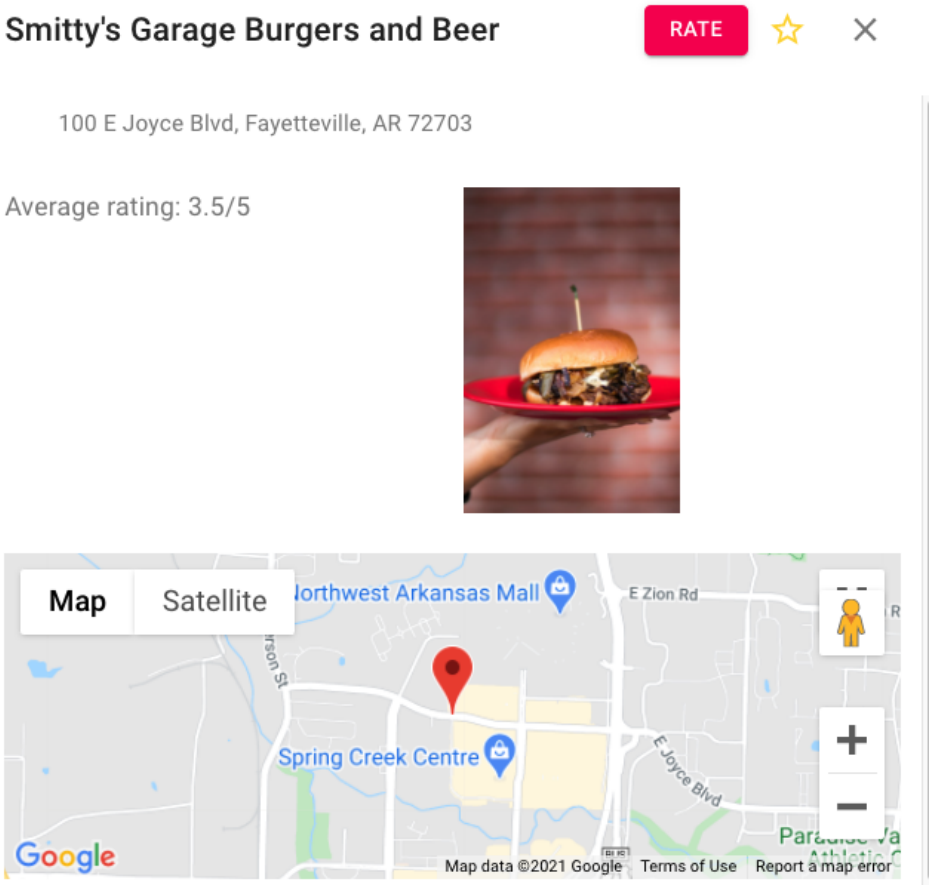

Figure 9: Detail page

Favorites Page

After the user is logged in, three new pages appear for the user to access: Recommendations, Favorites, and Chatbot. The favorites page presents a heading to suggest the correct destination and the restaurants that were previously starred by the user, where the user selected the star icon on the restaurant card and indicates the restaurant is one of their preferences. With the star icon being selected, the restaurant information would be retrieved using the yelp fusion API and be sent to the favorites table in the PostgreSQL database. Therefore, the preferences are saved in the database and would remain there until the user decided to remove that restaurant. The starred restaurants are presented in a card fashion, where an image of the restaurant appears on the top, followed by the name and the address of the restaurant in the middle, and a rate bottom and a star icon on the bottom of the card. A click on the card would introduce a pop-up window showing more details regarding the restaurant with an embedded map using the google map API. The rate bottom would launch a new window with the rate functionality, and finally the star icon would delete the restaurant from the favorite table and remove the card from the favorites page if the icon is deselected.

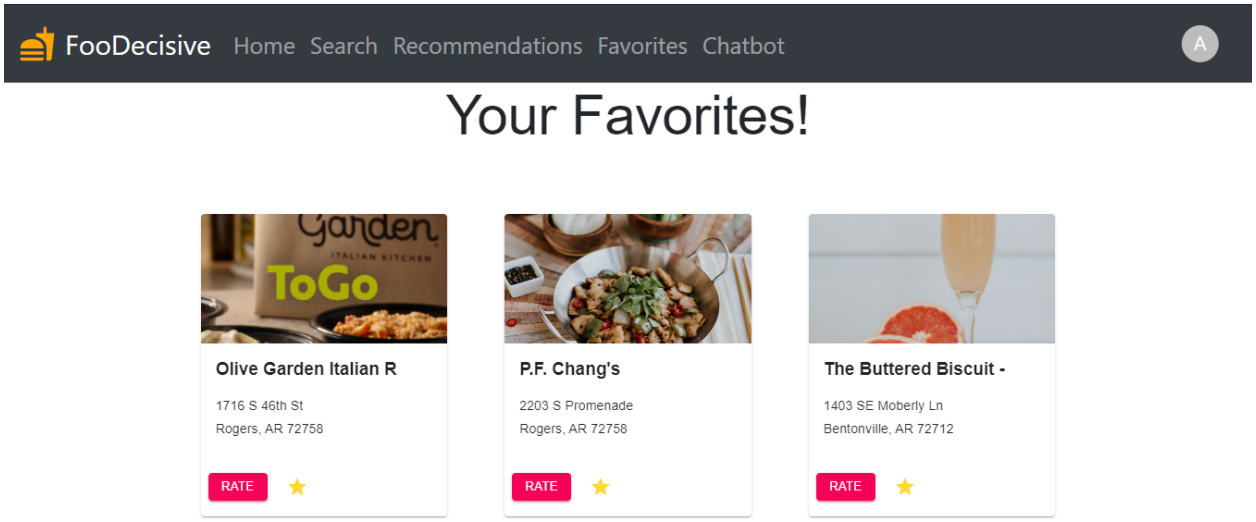

Figure 10: Favorites page

Rate/Review Functionality

One of the core functionalities in our application that is critical to developing our recommendation engine is our rating functionality. Once a user searches for a restaurant and opens up the details in the restaurant detail popup as discussed above, the user has an option to rate and leave a review of the restaurant. This is a form that consists of a rating scale from 1-5 and a text input field where users can write a review. The rating, review, user information, and restaurant ids are all stored in the reviews database relation upon submission. This is simply done by creating an endpoint in the backend to commit the information to the reviews table.

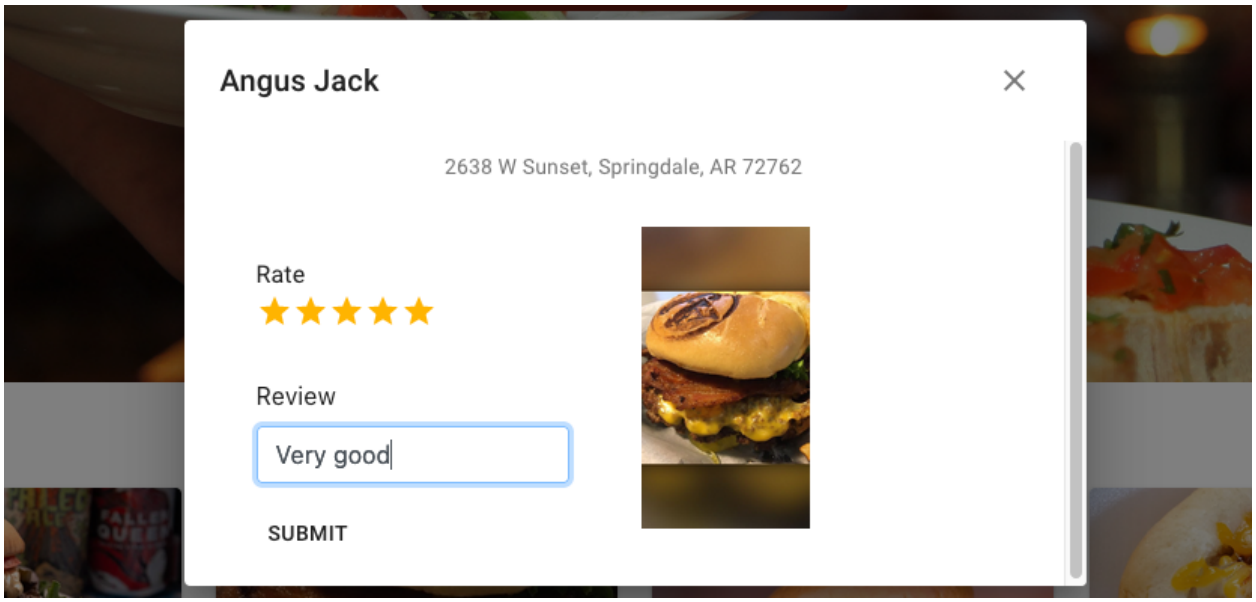

Figure 11: Rate/review functionality

Recommendation Algorithm and Page

What is it and why do we need it?: Recommendation systems have been used in a variety of applications to improve customer experience by providing personalized recommendations to users. We decided to take advantage of these systems and embed them into our application as a way of improving user experience. We develop a personalized restaurant recommendation system using collaborative filtering. The idea behind these types of recommender systems is that they take into account users' interactions with each other as well as with items. Accordingly, these algorithms are able to compute a user matrix and an item matrix and fill in missing ratings between users and items. Generally, the data we are given is incredibly sparse to begin with. That is, users may only have rated an incredibly small subset of all the restaurants we have. For instance, in the Yelp Open dataset, we have 10,000 users that have rated over 64,000 restaurants. However, the ratings between users and restaurants are incredibly sparse and the majority of elements in our matrix is 0. Now, the goal is to factorize this matrix in such a way that we uncover hidden latent factors and are able to predict what a user would rate a specific restaurant. Now, we can take the top K of the restaurants (perhaps the ones that the user would have rated the highest) and serve those as the recommendations to that user. We employ a nonnegative matrix factorization algorithm known as Alternating Least Squares (ALS), which is an algorithm that aims to factorize a nonnegative matrix into a product of two nonnegative matrices represented as the user and restaurant matrix, respectively. This is an optimization problem that is solved by the ALS algorithm from the pySpark API. We use parallel computing to speed up our model training process. To optimize our model, we have a number of hyperparameters that we tune accordingly to obtain the best possible recommendations for a given user.

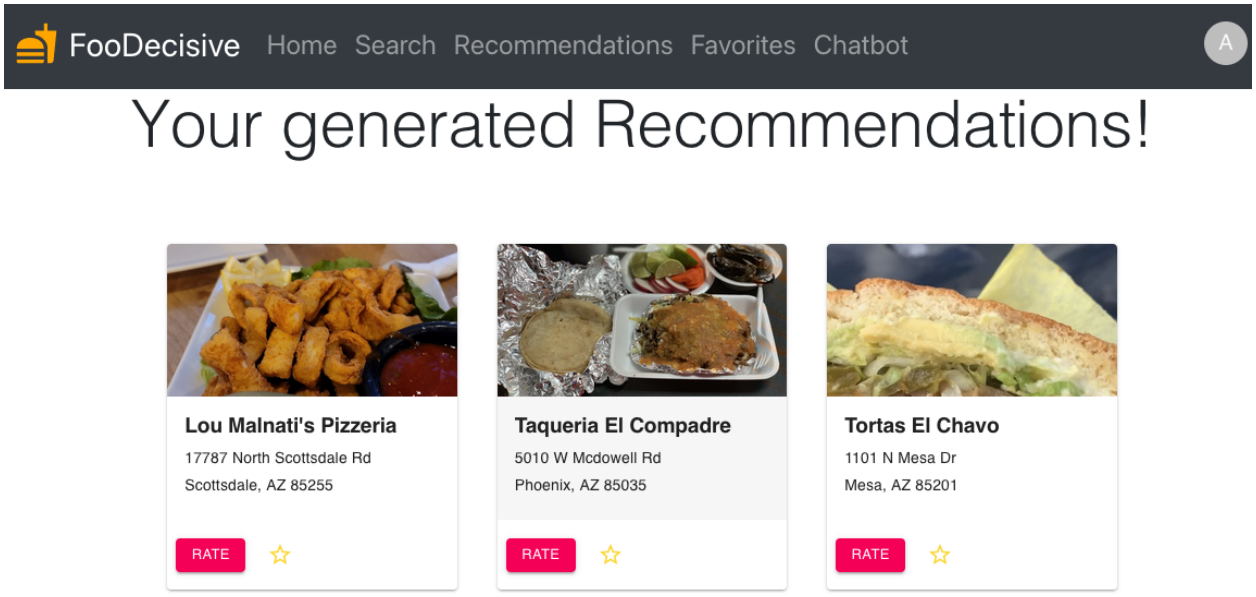

Figure 12: Recommendation page

Implementation: The recommendation algorithm was developed on the Databricks platform using Apache Spark. First, we establish a JDBC connection to the remote postgres database hosted on AWS RDS. We obtain the ratings for each user from the review table in the postgreSQL database as a pySpark dataframe and use these ratings as the basis to compute our factorization. We split the data into 60% training, 20% for validation with hyperparameter tuning, and 20% testing. We tune the following hyperparameters: number of iterations, rank of the factorization, and the regularization parameter. We find that 2 iterations, rank 50 factorization, and 0.2 as the regularization parameter yields the optimal results for recommendations. We take the top 10 recommendations and filter those that have a rating greater than 2.5 to recommend to the user. Finally, we establish a JDBC SSL connection back to the postgres database and commit the recommendations along with the restaurant ids to the recs table in the postgres database. This is now rendered in each user's recommendation feed based on their user id.

Recommendations Page: We simply develop an endpoint in the backend to retrieve the recommendations from the recs table in the postgres database based on the user id. So, different users will have a different recommendation feed, but their ratings affect the restaurants that users with similar tastes get recommended.

Chatbot UI and Functionality

An important part of our application is our conversational chatbot, Jessie. The intent of this chatbot is to increase interactivity to help engage users. Jessie can be accessed from any page in the application if the user is logged in. The UI of the chatbot page consists of a messaging component, where the user can send messages to and read messages from Jessie, and a restaurant list component, which is initially empty until the user asks Jessie to search for restaurants. Messages from the user are interpreted by wit.ai, a website that allows users to train a bot to recognize certain intents and entities. For example, if the user types, "Show me the best restaurants in Fayetteville, AR," then wit.ai will recognize that message as a *search* intent, with two entities: a *search query*, which is "restaurants" in this case, and a *location*, which is "Fayetteville, AR" in this case. User inputs are sent to wit.ai via their API, and the returned data is parsed and interpreted in our application. Our bot on wit.ai can recognize various intents: example, meaning the user is requesting an example of what kind of questions they can ask Jessie, goodbye, greeting, search, and thanks. Jessie will send a response based on the intent that wit.ai recognized. For our previous message example, Jessie could respond with, "Showing restaurants in Fayetteville, AR." Each intent has three possible responses, one of which is randomly chosen every time the user sends a message in order to give the bot a more "human" feel. Our chatbot also responds with relevant gifs, which are also randomized, again to make the bot seem more human. When the user sends a message with a search intent, Jessie will use Yelp's API to search for the entered search query and display the results in the restaurant list

component. If the user does not provide a location in the search message, Jessie will, by default, use the user's current location.

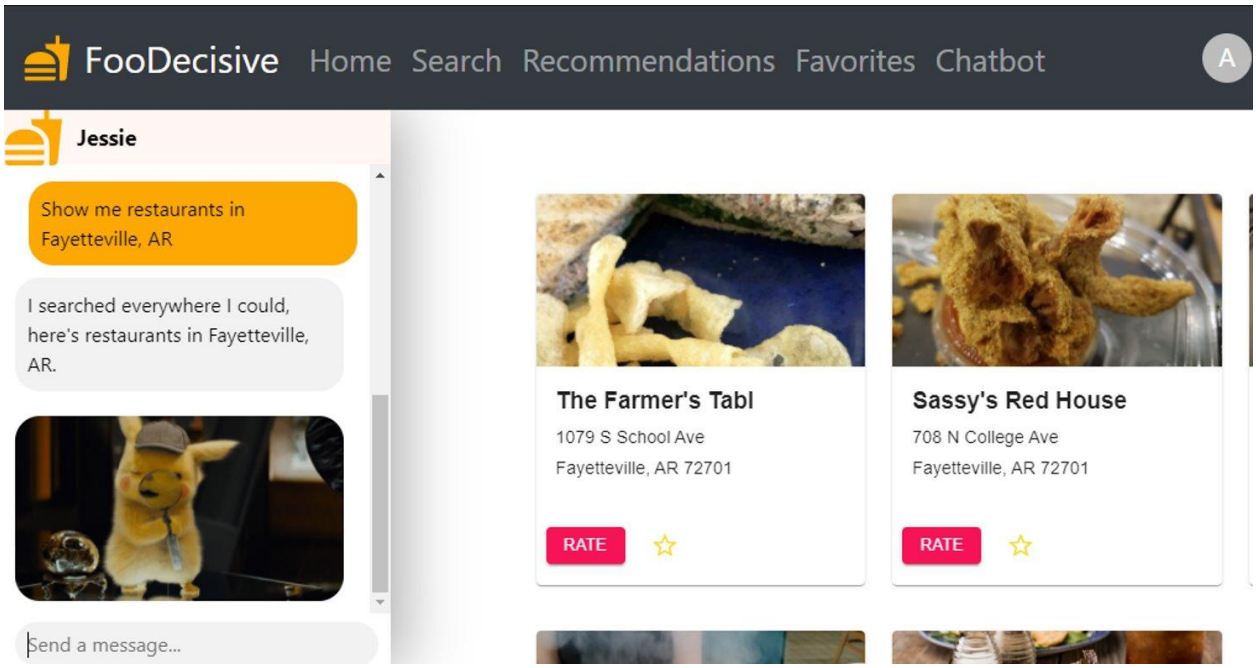

Figure 13: Chatbot page

4.3 Risks

Risk	Risk Reduction
User Data Security	Flask protects against cross-site scripting (XSS)
Data Sparsity/Cold Start	Get multiple users to use the application to help get more accurate recommendations to users
Dimensionality & Complexity of Data	Use PostgreSQL for scalability and query optimization/performance

4.4 Tasks

1. Design of sketches/views of the application, flowchart
2. Create skeleton webpages for each view
3. Setup basic backend CRUD operations in Flask
4. Setup PostgreSQL, establishing database connectivity, and develop basic schema

5. Develop a User-Authentication system to validate user credentials
6. Research Collaborative Filtering Recommendation Systems
7. Preprocess Yelp data to include only restaurants, etc.
8. Populate Yelp ratings data into PostgreSQL database
9. Test Retrieval of restaurant data using CRUD operations
10. Use Yelp Fusion API to retrieve restaurant ratings, location, etc. (Testing) and integrate Google Maps API
11. Implement conversational chatbot feature for restaurant information retrieval
12. Begin development of recommender system in Python using PySpark and Databricks
13. Implement recommender system and deploy to interact with incoming data from frontend
14. Polish up frontend and add clean and aesthetically pleasing design elements
15. Connect frontend and backend through middleware for communication

4.5 Schedule

Tasks	Dates
1. Research recommender systems and design sketches of UI and architecture (All members)	11/14-12/7
2. Exploratory Data Analysis of Yelp Dataset in Jupyter Notebook (Aneesh)	12/7-1/11
3. Establish database schema and connectivity to backend (Aneesh)	1/11-1/25
4. Develop Search functionality using Yelp Fusion API and frontend view (Huy)	1/25-2/8
5. Conduct feature engineering on Yelp data (Aneesh)	2/8-2/22
6. Implement login and registration frontend pages and routing (Adam)	2/8-2/22

7. Implement token based user authentication for login/registration in backend (Aneesh)	2/8-2/22
8. Implement restaurant detail popup in frontend view (Huy)	2/8-2/22
9. Implement rating/review form in frontend (Adam)	2/22-3/8
10. Develop backend endpoint for ratings and reviews commit to DB (Aneesh)	2/22-3/8
11. Embed Google Maps API into restaurant detail popup (Tay)	3/8-3/22
12. Develop Favorite functionality frontend view and backend endpoint (Huy)	3/8-3/22
13. Implement Favorites page in frontend to view list of favorites (Huy and Zhi)	3/22-4/5
14. Utilize HPC GPU to finely preprocess Yelp data and filter data based on requirements (Aneesh and Adam)	3/22-4/5
15. Develop Recommendation System using Apache Spark (Aneesh)	3/22-4/5
16. Implement conversation chatbot frontend UI (Adam and Aneesh)	4/5-4/19
17. Implement conversational chatbot functionality and develop	4/5-4/19

intents using wit.ai and incorporating responses (Adam)	
18. Develop profile page with history of user reviews/ratings (Huy)	4/5-4/19
19. Design homepage of application (Tay and Zhi)	4/5-4/19
20. Polish up aesthetic look throughout application and add transitions as needed. (All members)	4/5-4/19

4.6 Deliverables

- Design Document: Explains what our code does.
- UI drawings and flowcharts
- Database schema and initial data
- React and Flask code
- Jupyter Notebook for Recommender System (Python/PySpark)
- Final Report

4.7 Future Work

One issue that may be glaring about the search functionality is the use of precise addresses when a user chooses to set his or her current location as the location term. If we were to have more time, this would be one of the very first issues to address. One way we could work to ensure the confidentiality of the user's current location is by simply hiding the address as it is entered into the location text field. The address could be hidden in the same way passwords are hidden when users log into their accounts. Another alternative can be to hide the whole location text field altogether and include a message to the user saying, "Here are some restaurants near where you are at." This will improve user interaction for the website.

The Profile page is a simplistic page compared to other pages such as Home, Search, and Chatbot, so the design of the page can easily be more elaborate for future work. One way to improve the look of the profile page is to have an animated background, say a moving orange-white checkerboard, that is covered in the area where the review history is. Furthermore,

we could also allow the users to choose an icon to be their avatars. This would mean that as developers we would pick a set of 12 icons for the users to choose from (either from Material UI or by graphic design). Finally, the review history can be improved by including a timestamp for each review.

Due to time constraints, we were not able to productionize the recommendation system model itself to keep running in intervals. In the future, we plan on deploying our model to AWS and use Databricks cron jobs to run the model every hour or so to recompute recommendations based on the updated information of users and ratings. For this project, we employed the Databricks community edition, which only allows us to run our model on one cluster. However, with a paid subscription, we can use multiple clusters to speed up model training. Another flaw is that our model is trained on the Yelp Open dataset, which has restaurant reviews from the US and Canada over a variety of states and provinces, so recommendations (although effective) are not guaranteed to be from Arkansas. If we were to scrape ratings data from the Arkansas location only, our model would be able to recommend restaurants in Arkansas to users in Arkansas. This is a simple task, but would take a significant amount of time to obtain enough data to train on, so we decided it was out of the scope of our project timeline.

5.0 Key Personnel

Adam Greenfield – Greenfield is a junior/senior Computer Science/Computer Engineering major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has taken relevant courses such as Artificial Intelligence, Software Engineering, and Database Management Systems. He has completed relevant work in these courses. Greenfield is responsible for backend, chatbot development, and assisting with data preprocessing on the FooDecisive project.

Aneesh Komanduri – Komanduri is a senior Computer Engineering, Computer Science, and Mathematics major at the University of Arkansas. He has completed relevant coursework in Software Engineering, Database Management Systems, Numerical Linear Algebra, and Data Mining. He has internship experience as a Digital Security & Cloud Engineering Intern at Phillips 66 and research experience as a Research Assistant in the Data Science Lab. Komanduri is responsible for development of the recommender system in Spark, backend/database development, and cloud deployment of the application.

Huy Mai – Mai is a senior Computer Science and Mathematics major at the University of Arkansas. He has completed relevant courses such as Software Engineering, Data Mining, Symbolic Logic I, and Abstract Algebra. He has internship experience as an Application Development Intern at J.B. Hunt Transport Services, Inc. and as a summer research intern at the Autonomous Technology Research Center in Dayton, Ohio. Mai is responsible for frontend, backend, and database development.

Tay Pham – Pham is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed relevant courses such as Software Engineering, Database Management Systems and Operating Systems. He has job experience as a teaching assistant at University of Arkansas in Fort Smith. Pham is responsible for assisting with frontend and backend development on this project.

Zhi Han Weng – Weng is a senior Computer Engineering major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed relevant courses such as System Synthesis & Modeling, Operating Systems, Circuits & Electronics, Low Power Digital Systems, and Software Engineering. He is currently working in a research lab for Dr. Jia Di in preparation for attending graduate school. Weng is responsible for assisting with frontend and backend development.

6.0 Facilities and Equipment

We used the Pinnacle Cluster at the Arkansas High Performance Computing Center to preprocess and conduct feature engineering on our Yelp training data.

7.0 References

[1] "React - A JavaScript library for building user interfaces", React, https://reactjs.org/

[2] "PostgreSQL: The World's Most Advanced Open Source Relational Database", https://www.postgresql.org/

[3] "Flask - web development, one drop at a time", https://flask.palletsprojects.com/en/1.1.x/

[4] "Apache Spark: Lightning-fast unified analytics engine", https://spark.apache.org/

[5] "Collaborative Filtering | Recommendation Systems" https://developers.google.com/machine-learning/recommendation/collaborative/basics

[6] "Yelp Developers", https://www.yelp.com/developers

[7] "Wheel of Dinner", https://wheeldecide.com/wheels/food-drink/wheel-of-dinner/

[8] "Yelp", https://www.yelp.com/

[9] "Tripadvisor: Read Reviews, Compare Prices & Book", https://www.tripadvisor.com/

[10] "Scientists Uncover Why You Can't Decide What to Order for Lunch." California Institute of Technology, https://www.caltech.edu/about/news/scientists-uncover-why-you-cant-decide-what-order-lunch-83881

[11] "Time spent in primary activities and percent of the civilian population engaging in each activity, averages per day by sex, 2019 annual averages." U.S. BUREAU OF LABOR STATISTICS, https://www.bls.gov/news.release/atus.t01.htm

[12] “American couples spend 5.5 days a year deciding what to eat.” New York Post, https://nypost.com/2017/11/17/american-couples-spend-5-5-days-a-year-deciding-what-to-eat/

[13] “Maps Javascript API”, https://developers.google.com/maps/documentation/javascript/overview

[14] “Collaborative Filtering Recommender Systems”, http://files.grouplens.org/papers/FnT%20CF%20Recsys%20Survey.pdf

[15] “Databricks”, https://databricks.com/

[16] “Google Maps Platform”, https://developers.google.com/maps

[17] “WitAI”, https://wit.ai/