**Software Requirements and Design Document**

**For**

**Group 8**

Version 1.0

**Authors**:

Amelia Sieg

Reagan Bourne

Julia Sveen

Madalina Bulat

# Overview (5 points)

A website with a restaurant chat bot that gives the user nearby restaurants based on the cuisine type that they enter and the cost that they are willing to pay. It also provides the user with options based on their location. Once the user has acquired multiple options, it then sorts it based on highest rating, and closest location, based on the preferences. First, the website will ask you to enter the preferences, and then it will provide the user with one restaurant at a time. The user either has the option to answer “yes” or “no” to the recommendation, and if the user says “no” then the restaurant bot will return another answer.

We are focusing on programming the back end in Python and connecting it with the front end with React. This project is based on the Yelp Fusion API. The Yelp Fusion API allows the user to get the best local content and user reviews from millions of businesses around the world.

* *Top of Form*

*Bottom of Form*

# Functional Requirements (10 points)

1. Restaurant Search (High Priority)

* The system allows users to input a search term and a location to find relevant restaurant options.

1. Display Search Results (High Priority)

* The system displays the search results from the Yelp API in a nice, readable format, including details like the restaurant's name, rating, address, and phone number.

1. Responsive Design (Medium Priority)

* The system will be a website and will be optimal on a PC or desktop. However, it will also be accessible on mobile devices, ensuring its use across different screen sizes and devices.

1. Error Handling (High Priority)

* The system will display error messages if there is an issue with the search, like if the user inputs invalid search terms/phrases.

1. Input Validation (Medium Priority)

* The system will validate user inputs to ensure they are not empty and fit the correct format before making an API request.

6. Filtering Options (Low Priority)

* The system will allow users to filter search results based on criteria like ratings, price, or distance to them.

# Non-functional Requirements (10 points)

1. Performance (High Priority)

* The system will return search results within 3 seconds of the user initiating a search.

1. Scalability (Medium Priority)

* The system will support up to 10,000 concurrent users without a degradation in performance.
  1. Availability (High Priority)
* The system shall have an uptime of 99.9% excluding planned maintenance.
  1. Security (High Priority)
* All data exchanges between the system and Yelp's API shall be encrypted using HTTPS.
  1. User Interface Responsiveness (Medium Priority)
* The system's user interface shall adapt to different device screen sizes, ensuring a consistent and user-friendly experience on both mobile and desktop devices.
  1. Maintainability (Medium Priority)
* The system's codebase shall be written following best practices, making it modular and well-documented, ensuring that developers can efficiently update or fix the system.
  1. Usability (High Priority)
* New users shall be able to initiate a search within 30 seconds of their first visit without any guidance.
  1. Reliability (High Priority)
* In the case of a failed search query, the system shall provide clear error messages and automatically retry the request up to 3 times before notifying the user of a failure.
  1. Compatibility (Medium Priority)
* The system shall be compatible with the latest versions of major browsers, including Chrome, Firefox, Safari, and Edge.

# Use Case Diagram (10 points)

* Use Cases:
  + Search for Restaurants
  + View Restaurant Details
  + Filter Restaurant Results
  + Sort Restaurant Results
* Relationships:
  + "Search for Restaurants" can include "Filter Restaurant Results" and "Sort Restaurant Results"
  + "Search for Restaurants" leads to "View Restaurant Details".
* Use Case 1: Search for Restaurants
  + Description: User inputs search criteria (like location, cuisine, etc.) to search for restaurants.
  + Precondition: User has access to the restaurant search system.
  + Postcondition: A list of restaurants that match the search criteria is displayed.
  + Primary Flow: User accesses the search page. User inputs search criteria. System fetches and displays the search results.
* Use Case 2: View Restaurant Details
  + Description: User selects a restaurant from the search results to view detailed information about it.
  + Precondition: User has searched for restaurants and has a list of results.
  + Postcondition: Detailed information about the selected restaurant is displayed.
  + Primary Flow: From the search results, the user selects a restaurant. System fetches detailed information about the restaurant. System displays the detailed information.
* Use Case 3: Filter Restaurant Results
  + Description: User applies filters like price range, rating, etc., to narrow down the search results.
  + Precondition: User has searched for restaurants and has a list of results.
  + Postcondition: The search results are updated based on the applied filters.
  + Primary Flow: User selects filters from the available options. System updates the search results based on the selected filters.
* Use Case 4: Sort Restaurant Results
  + Description: User sorts the search results based on criteria like rating, distance, etc.
  + Precondition: User has searched for restaurants and has a list of results.
  + Postcondition: The search results are sorted based on the selected criteria.
  + Primary Flow: User selects a sorting criterion from the available options. System sorts and displays the search results accordingly.Bottom of Form

# Class Diagram and/or Sequence Diagrams (15 points)

Sequence Diagram:

* Viewing the App: User opens the app and sees the LEFT NO CRUMBS title, the dancing robot, and the question ARE YOU READY TO CLEAN YOUR PLATE with "Yes" and "No" buttons.
* Searching for Food: User clicks "Yes", which reveals the search input fields and button. The user fills in a food term and location, clicks "Search", and sees results.
* Showing results for Food: Based on what the food and location that the user entered, the API will print out a randomized restaurant in the area.

a) Class Diagram:

* Classes/Components:
  + App: The main component that stores everything.
    - Character: Represents the visual of the dancing robot
    - Robot: Represents the input fields and the search button for the Yelp API. It's more of a functional component that also handles states related to search and results.
* Attributes:
  + App:
    - showSearch (State that toggles the display of the search functionality)
      * Robot:
        + term (State for the food term)
        + location (State for the location)
        + results (State for the Yelp API results)
* Methods/Functions:
  + App:
    - handleYesClick (Handles the "Yes" button click, scrolls and triggers the search display)
      * handleNoClick (Handles the "No" button click, scrolls)
        + Robot:
        + handleSearch (Triggers the Yelp API call)
* Relationships:
  + App uses both Character and Robot. This is a composition relationship since App is composed of Character and Robot.

# Operating Environment (5 points)

Operating Environment Description Hardware Platform:

Standard Personal Computers and Laptops, Mobile Devices

Operating System:

Windows, macOS, Linux, Mobile OS

Web Browsers:

Google Chrome, Mozilla Firefox, Microsoft Edge, Safari

Other Software Components:

Node.js, npm

Applications with which it must coexist:

Backend API Server, Yelp Fusion API

# Assumptions and Dependencies (5 points)

1. Assumptions:
   * Yelp API Stability: It's assumed that the Yelp Fusion API will remain stable, with consistent endpoints and data structures. Significant changes or deprecations could necessitate code modifications.
   * Web Browser Compatibility: The application assumes that users will access the platform through updated and modern web browsers, which support the latest web technologies.
   * Constant Internet Connection: The system's functionalities, especially interactions with the Yelp API, assume users have a reliable and continuous internet connection.
   * Backend API Stability: If there's a backend server, it's assumed that the endpoints, like /search\_yelp, will remain consistent throughout the development and operational phases.
   * Third-party Packages: The application may depend on third-party npm packages. It's assumed that these packages will remain maintained and compatible with newer versions of React and Node.js.
   * Responsive Design: For mobile device accessibility, it's assumed that the application has been designed responsively, ensuring functionality across various screen sizes.
2. Dependencies:
   * Yelp Fusion API: The project depends on the Yelp API for fetching restaurant data. If Yelp changes its API terms, rate limits, or structures, the project may need adjustments.
   * Node.js & npm: The React application's development, build, and deployment processes are dependent on Node.js and npm.
   * Backend Server (if applicable): The project may depend on a backend server to handle specific tasks like interfacing with databases or proxying API requests. If there's a shift in backend technologies or architectures, it might necessitate frontend changes.
   * React & React Libraries: The project heavily relies on the React library and potentially other React-related libraries (like React Router). Updates or major changes in these libraries might impact the project.
   * Database Systems (if applicable): If the backend interfaces with a database, then the project is dependent on that database's availability, structure, and performance.
   * Reusable Software Components: If components or functionalities from previous projects are integrated, the project becomes dependent on the continued compatibility and functionality of those components.