Final Documentation

SeeFood

Prepared by: Kathryn Kingsley, Hung-Jui Guo, Kwonyoung Lee, James Dao, Faris Nasoetion, Brendan Frost, Abhinav Kotra

Date: December 02. 2019

Table of Contents

Description

The main goal of SeeFood is to solve indecisiveness. More specifically, the indecisiveness one faces when selecting a restaurant. In order to help people decide on a restaurant, the team created a web application. This web app would have an easy-to-navigate home page, and would also be able to resize to any mobile device's resolution. In addition, the user would be able to utilize the homepage's drop-down menus to input the type of cuisine, price, and the distance they want, and upon clicking the submit button, get a result in the form of a restaurant's information. The information of the restaurant would be displayed in card form. The web application would also be able to handle POST and GET requests, as well as utilize an API. If the user does not like the suggestion the application gives them, they would be able to request another recommendation. Furthermore, the app will be able to display a recommendation within a certain mileage, as well as display information that wasn't on the initial recommendation card.

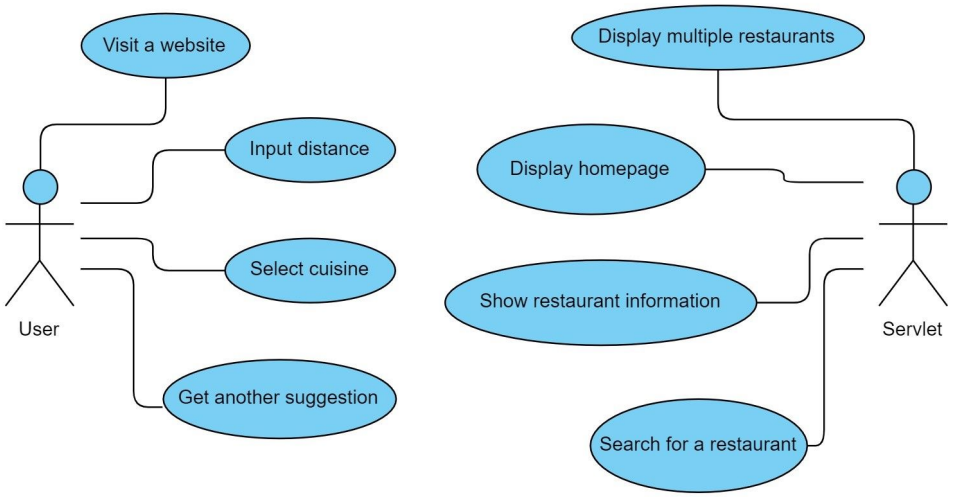

Figure 1. Use Case diagram of SeeFood which visualizes many of the project's goals

Architecture & Design

The SeeFood web app has a Java servlet called SeeFoodServlet.java. It is located in the "src" folder, which is in the SeeFood folder. The servlet handles GET and POST requests. The function that handles GET requests is called "doGet". It displays an HTML page for the user. The user can select a distance and cuisine and then click "Submit". Then, a POST request is sent to the servlet. POST requests are handled by the "doPost" function. The function reads the user input. The function has default values for cuisine and distance if the user does not make a selection. Then, the function determines the ID for the cuisine and converts distance from miles to meters. Next, a GET request is sent to the Zomato API with the cuisine ID and

distance in meters as query parameters. The API returns a list of restaurants in JSON format. A restaurant is randomly chosen and its information is gathered. Finally, the restaurant info is displayed through a jsp file.

To expand on jsp files, three jsp files were created for our web app. They are located at “SeeFood/WebContent/WEB-INF/jsp/”. “Homepage.jsp” is the first page shown to the user. “ResultsPage.jsp” is shown after the user submits a POST request. “NoRestaurant.jsp” is shown if a restaurant could not be found for the user. Bootstrap was used to design our web pages so that the app looks good on a phone, as well as on a desktop computer. This decision was also due to the fact that it was assumed this web app would be used mostly by people on their phones. The main purpose of the app is to help people quickly choose a restaurant when they are with family and friends. When a restaurant is selected, its information will be displayed in a Bootstrap card in the center of the screen. The card will include the restaurant’s photo, name, cuisine, address, phone numbers, and a link to more info. In addition, there will also be a button the user can click if they want another recommendation. When the button is clicked, another POST request is sent to “SeeFoodServlet.java”.

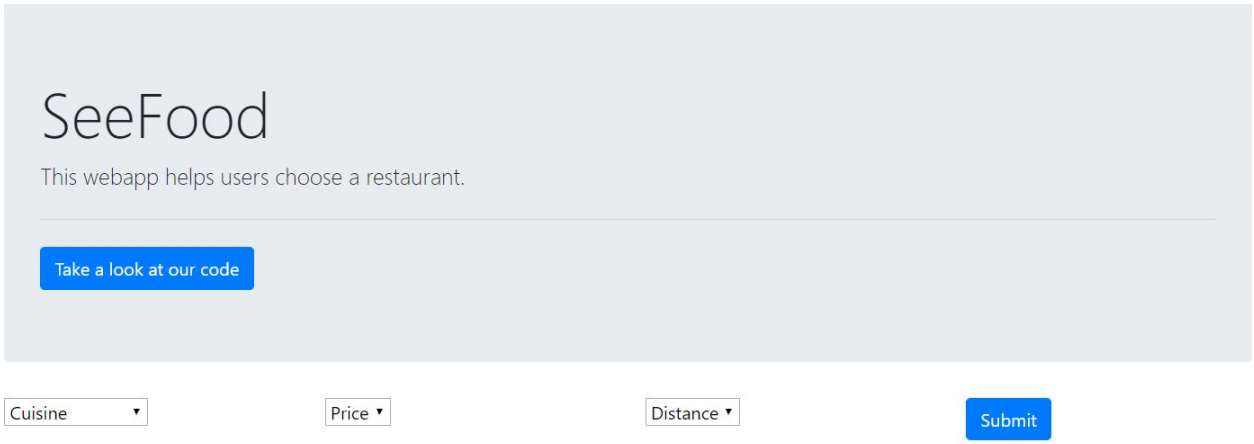

Figure 2. A picture of Homepage.jsp, which utilizes Bootstrap

Unfortunately, there was not enough time to write the code for determining a user's location. So, the coordinates are constant. The user's location is set up so that they are in Dallas. Dallas was chosen because it made testing easier. This was to ensure that the web app was suggesting real restaurants. Another constant in the program is the API token. The token is valid forever, and it was needed to use the Zomato API.

Another problem that had come up during development was that the Zomato API may not always have the information that was wanted by the user. What if the API does not have a restaurant's phone number? To resolve this it was decided that the application will have default values for all the variables that hold the chosen restaurant's info. The default value is "Not Found."

Furthermore, several tests were implemented before completing each of our user stories. The tests are in "SeeFoodTest.java" which is

in the same folder as “SeeFoodServlet.java”. JUnit testing was used for this project because the application used a Java Servlet. The following functions were tested: getCuisineID, convertMilesToMeters, and getRestaurant. Since getRestaurant randomly chooses a restaurant, a specific restaurant could not have been used for any of the test cases. The test cases made sure the function did not return null when there were restaurants that fit the user’s parameters. They also made sure no exceptions were thrown. For the getRestaurant function, every possible combination of cuisine and distance was used for testing.

The other test is in SeeFood Test folder, divided into HomePageTest and ResultPageTest different tests. These tests make sure that every button and drop-down menu are work and not broken during our project processing. Detail operation is in the appendix part.

Appendix

Appendix A: How to run tests

JUnit testing was one method to testing used for this project. This was chosen because the application had a Java servlet. To run the tests in Eclipse, right-click on the project in the Project Explorer tab. Then click on “Run as” in the drop-down menu and select the “JUnit test” from the list on the side.

AutoHotKey was another method to testing used for this project. This testing was a blackbox testing. Since our homepage had some buttons and drop-down menu, we used AutoHotKey to automatically press

these buttons and drop-down menus. Here is the instruction of how to use this testing method:

Put the test file (GithubClick.exe) on your desktop, then open the web app. Open/double-click GithubClick.exe with your web browser as the last visited program on your computer(Alt-Tab should bring you between your desktop and your web browser in this case). The .exe will automate the actions of selecting menus in the web app.

In HomePageTest:

AllChoose.exe -> Set all drop-down menu to the first choices, and click submit.

CuisineChoose.exe -> Choose first cuisine and click submit.

DistanceChoose.exe -> Choose first distance and click submit.

GithubClick.exe -> Click "Take a look at our code".

PriceChoose .exe -> Choose first price and click submit.

In ResultPageTest:

WebsiteClick .exe -> Auto press "Restaurant's website" button.

PickAnotherClick .exe -> Auto press "Pick another restaurant" button.