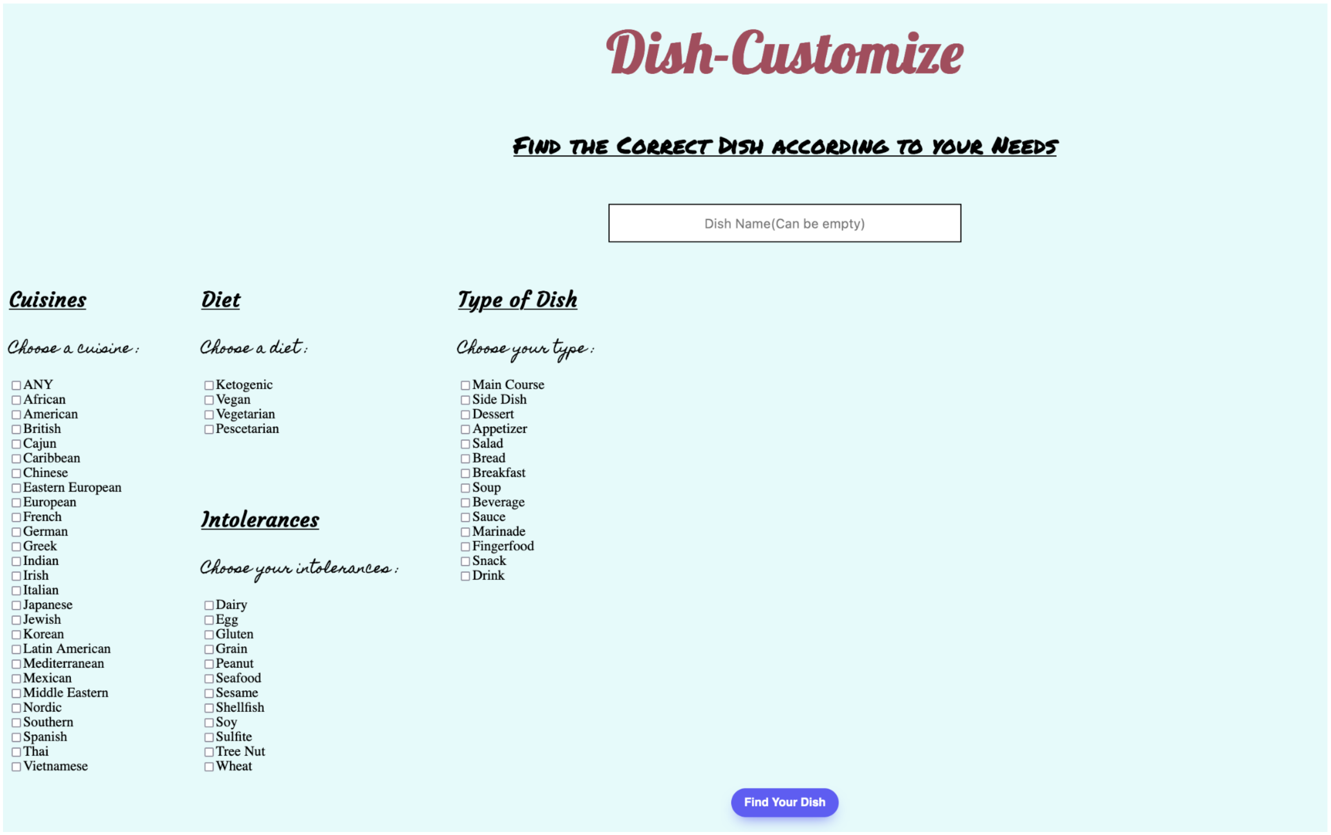

Dish-Customize

FIND THE CORRECT DISH ACCORDING TO YOUR NEEDS

Dish Name(Can be empty)

Cuisines

Choose a cuisine :

☐ ANY
☐ African
☐ American
☐ British
☐ Cajun
☐ Caribbean
☐ Chinese
☐ Eastern European
☐ European
☐ French
☐ German
☐ Greek
☐ Indian
☐ Irish
☐ Italian
☐ Japanese
☐ Jewish
☐ Korean
☐ Latin American
☐ Mediterranean
☐ Mexican
☐ Middle Eastern
☐ Nordic
☐ Southern
☐ Spanish
☐ Thai
☐ Vietnamese

Diet

Choose a diet :

☐ Ketogenic
☐ Vegan
☐ Vegetarian
☐ Pescetarian

Intolerances

Choose your intolerances :

☐ Dairy
☐ Egg
☐ Gluten
☐ Grain
☐ Peanut
☐ Seafood
☐ Sesame
☐ Shellfish
☐ Soy
☐ Sulfite
☐ Tree Nut
☐ Wheat

Type of Dish

Choose your type :

☐ Main Course
☐ Side Dish
☐ Dessert
☐ Appetizer
☐ Salad
☐ Bread
☐ Breakfast
☐ Soup
☐ Beverage
☐ Sauce
☐ Marinade
☐ Fingerfood
☐ Snack
☐ Drink

Dish-Customize

Zense Recruitment 2022

Vikas Kalyanapuram
IMT2021040

Objective

The dish-customize WebApp will help find dishes according to your requirement.

What does it do?

1. The website will first ask you for the name of a dish. This field can be left empty as well if you want any dish according to your requirements.
2. Then there are a list of fields which will help you describe the various parameters that need to be satisfied in your dish.
3. These fields include:
    a) Cuisine type
    b) Diet
    c) Intolerances
    d) Type of meal

Finally, based on the options chosen and the dish (if entered) the WebApp will return a list of dishes that satisfy all the fields.

The names of the dishes will have a hyperlink to a google search that has the name of the dish searched if the user wants additional info (e.g. recipe) about the dish.

Frameworks Used:

1. The front end uses EJS, CSS and JavaScript for rendering the website
2. The backend uses Express JS (Node JS) for
    a) Parsing data from the form (body parser)
    b) Sending data to the EJS files to render the html webpage
    c) Fetching data using the **spoonacular API**.
3. The npm modules used include: body-parser, ejs, express, node-fetch

Running the Website:

In order to run the website locally :

a) Make sure NodeJS is installed locally on the system

b) If it is, download the zip file of this project and then unzip it.

c) Open terminal in the folder and type 'node server.mjs' in terminal. A message "Listening on port 3000" must appear in the terminal

d) Open a browser and type localhost:3000/ in the URL bar.

e) The first image in the report must appear as the first webpage.

f) Now, the website is working.

In order to access the website without running it locally, go to the following URL:

- https://thawing-basin-91829.herokuapp.com/

Interesting aspects of the Project:

1. As the first webpage has a long list of options, it was impractical to type them all out in an HTML file. Hence, I had to learn EJS and pass the list as an array from the *server.mjs* file and run a for loop in the *index.ejs* file to display the list (within the form).
2. As I couldn't use *querySelector()*, I needed to find a way to find which checkboxes were clicked (in the ejs file) within the mjs file. This led me to learn about the *.hasOwnProperty()* method which can be used on the body that was obtained from the bodyParser.
3. After generating the apiURL (which wasn't much of a challenge after obtaining the ticked fields), I needed to GET info from the **spoonacular** website, using their API. This was the hardest part as *https.get()* was not working and kept giving a JSON parsing error. Hence, I needed to learn a new way of utilizing APIs using *await* and *fetch()*.
4. The final interesting aspect was rendering the resulting dishes in the *index2.ejs* file. This included aligning the images and the dish name as well as hyperlinking the dish name to google with the name already searched which all required some embedded JavaScript.

5. Learning git and version control was crucial as this was my first software project. Also learning how to deploy my server using heroku and the terminal was also an interesting aspect of the project.

* The aligning and styling using CSS was also challenging but as this is a WebApp, I kept the UI a little simple and paid more attention to the functionality of the website.

Potential Improvements:

1. Introduce a Sign up Method where users can sign in and store their favourite dishes and access them when signed in.
2. The spoonacular API is very limited in the number of results for a particular dish. Also, the number of results returned as the number of parameters increases, decreases a lot. Hence using a different API or some other method to find more dishes may need to be implemented.