CS373 IDB Technical Report: Group 15, WorldEats

Motivation:

Our goal is to connect food enthusiasts to recipes, restaurants, and cities where they can find delicious food from around the world.

App Overview:

Our website, worldeats.me, organizes information about recipes and restaurants into three models — recipes, restaurants, and recipes filtered by city. Data is gathered from multiple APIs and linked together through a webpage. Each model has a page of clickable instance cards, and each instance page contains attributes that could be useful to a customer.

User Stories:

User Story #1: Homepage — *"Would like to see a homepage with some relevant information. Maybe just a short description of what the site will be and how to use it. And also maybe an option to choose a random city/recipe/restaurant based on another input?"*

Response: We will address this problem in phase two of the project. At that point, we plan to have a homepage with information providing an overview of the purpose of our website and how to use it. The suggestion of a randomizer is intriguing. We will consider it as an extra feature to add in a later phase.

User Story #2: Connect the Models — *"Perhaps not in this phase but I would like to see some more interconnectedness between the various models. Maybe allow the user to designate a specific city and the restaurants will change based on the chosen city. Or the user can pick a certain recipe and then they can see restaurants those recipes are available in and what cities those restaurants are located in?"*

Response: Connecting the three models is our top priority moving forwards. We have already connected the restaurants displayed to those mentioned on the cities page and plan to do much more in the future.

User Story #3: More Information on the Primary Recipes Page — *"As a user, I would like some more information earlier when I first click the recipes page. Currently, other than the recipe name and picture, all I have is the number of upvotes. I am a little confused about how exactly those upvotes were determined or where they came from so maybe some clarification there. Ideally, I would like to see the cost, time to make, and total calories under the name so I can decide even faster which recipe I am interested in."*

Response: All of the information mentioned is scrapable from the API. Including more detail in the primary recipes page is a justifiable request. We are likely to heed this user suggestion in the next phase.

User Story #4: Cities Page — Add Ability to Sort + Little Bit More Relevant Information — *"As a user, I want to be able to sort the cities based on the ratings so I can ideally choose from the top-rated cities as an example. To add onto this, if I am using the website primarily for food and finding restaurants, I want the city ratings to be based on food and not overall (I'm not sure if you guys are using overall ratings or food ratings but if you guys are using food ratings then ignore this part)."*

Response: The city data is currently the weakest of the three models. We plan to have a recipes API feed into a cities API in later phases, a part of which would be either finding or calculating food ratings on our part.

User Story #5: Restaurants Page — Add Ability to Sort Based on a Variety of Factors — *"As a user, I would like to be able to sort the various restaurants on the page by a couple of different factors. These factors can include but are not limited to price, ratings, type of cuisine, and maybe also its distance from me."*

Response: This feature should naturally follow once complete dependence on the restaurant API is established. It is natural for a user to want to sort cards, so it is certainly on our to-do list.

RESTful API:

We designed a RESTful API that fetches data for recipes, restaurants, and cities from their respective APIs using Postman. The Models section contains further information about the specific APIs. The resultant design is at https://documenter.getpostman.com/view/25838982/2s93CExciz.

Models:

- **Restaurants:**
  - We have downloaded data from a few different APIs into src/mock-data/RestaurantData.js, src/mock-data/RestaurantReviews.js, and src/mock-data/RestDataHours.js.
  - We are using this downloaded data in src/pages/RestaurantPage.js and src/components/RestaurantCard.js. Within these two files, we are accessing the current restaurant and its data by searching for the id of that restaurant, which is passed in from src/pages/Restaurant.js in one of the three data files. Right now, IDs are hard coded since we have downloaded data.
  - All our CSS style for documents is in .css files with the same name as the .js file that they are styling or formatting. They are also located in the same directory as their corresponding .js file.
  - Restaurant.js is the main Restaurants page, which then connects to restaurant cards (which it has three of at the moment), and RestaurantCards links to RestaurantPages, saving us from creating a page for every instance.
  - We used three different calls to the Yelp API service to collect all of the data needed for this portion of the assignment.

- **Recipes:**
  - Sample API recipe data resides in src/mock-data/RecipeData.js, retrieved from the Spoonacular API.
  - The component for rendering the main recipe model page is src/mock-data/Recipes.js
    - On this page, recipes are displayed to the user using RecipeCard components. Users can click on the image of a RecipeCard to navigate to a recipe instance page.
  - RecipeCard.js can be found in src/components/RecipeCard.js. It is responsible for displaying a recipe preview to the user. This component takes in an ID for props, which is used to get the appropriate recipe data from RecipeData.js.
  - RecipePage.js is a component used to render a recipe instance, located at src/pages/RecipePage.js. RecipePage uses the useParams() hook from React Router, which allows it to retrieve its URL page id. Using this ID the appropriate recipe data is displayed from sample API data in RecipeData.js. This architecture enables us to display an unlimited number of recipes using a single component since it is adaptable based on the ID of the URL.
  - All component CSS files are in the same directory as their respective components.

- **Cities:**
    - The APIs for getting data about cities are pretty sparse. We are currently using the countries now API to load some mock data into /src/mock-data/cities.js and then manually adding some extra info to read into the CityPage instances, but for the final version, we will use a combination of APIs to get the information we want.
    - src/pages/Cities.js formats our city card pages. It displays cards of each city, showing its average rating and linking to the city's page. It uses a similar layout to the Recipes and Restaurants card pages. The city data structures in mock-data/cities have unique IDs, which are referenced to create city cards based on their other elements.
    - The formatting for the cards is found in Cities.css, a simple three-line class.
    - CityPage.js renders the city instances. It loads data from the mock data API to display the name of the city, its country, a picture, the country's flag, the popularity of the city, and popular dishes in that city. It fills in this information based on which city ID was provided.

Tools:

We used React and React Router to implement the front end and Bootstrap with CSS to style it. Using Namecheap, we claimed the worldeats.me domain. Our skeletal implementation of a backend server relies on AWS Amplify, which provides the RESTful API that the front end consumes. Using Postman, the RestFUL API was designed. Finally, we used Microsoft Teams and Zoom for communication and VSCode as our development environment.

Hosting:

We are currently hosting our website online using AWS through carangan@utexas.edu's profile. To host the website, we use AWS Amplify, a microservice provided by AWS. When given an ongoing Gitlab repository, it hosts the app we have online. The current root of the hosted website is currently at https://main.d3up18lducqmy8.amplifyapp.com/. This microservice issues an SSL certificate, keeping the site secured with HTTPS. Currently, there is an attempt to reconfigure the hostname to "worldeats.me." However, the CNAME record associated with the domain name has not been verified by AWS. This process can take up to forty-eight hours.