Cultured Foodies Technical Report

Authored by:

Vishal Tak

Lucinda Nguyen

Amy Ouyang

Tim Nguyen

Joshua Arrojado

Siddhesh Krishnan

Mission

Cultured Foodies was designed to bring the world of cuisine, restaurants, and cities to life in one website. Our website provides information about restaurants that have cultural cuisines in cities throughout the United States. With popping visuals and a streamlined UI, we hope to give people rich and cultural food for thought.

❖ User Stories

Phase 1 User Stories

- **Phase 1: User Story 1 (Make page titles more descriptive)**
  - As a user, it would be really helpful to have a descriptive title name for when I have multiple websites open.
  - **Estimated time:** 40 minutes
  - **Actual time:** 30 minutes
  - **Assumptions:**
    We need to edit the Tab name of the web page depending on the page we're currently on. The 'About Page' would be titled 'About'.
  - **Implementation:**
    We looked at React-dom documentation and just had to edit the Tab name in a *useEffect* webhook. The main challenge was finding documentation for this.
- **Phase 1: User Story 2 (Add calorie count per serving to recipe instance page)**
  - As someone who is trying to lose weight, it would be nice to know the calorie information for recipes.
  - **Estimated time:** 3 hours
  - **Actual time:** 5 hours
  - **Assumptions/Thoughts:**
    This task would involve using the 'Dish' API we found to get dishes data for 3 specific instances. It would require setting up an instance page that's redirected from the 'Dishes' grid page. We then need a decently formatted 'Dish' page with the calories count displayed.

- **Phase 1: User Story 3 (Add pictures to the about page)**
  - As a visual learner, it would be nice to be able to see pictures of all the developers on the About page. It would allow me to connect with them better.
  - **Estimated Time:** 1 hour
  - **Actual time:** 3 hours
  - **Assumptions**:
    This take would require us to align 6 grids for all members and display profile pictures.
  - **Comments:**
    It took much more time than we thought to format 6 grids with a circular profile picture. We went through many styling configurations and Row/Cols documentation to get this done.
- **Phase 1: User Story 4 (Add flags to country page)**
  - As someone interested in exploring recipes from different countries, it would be helpful to have a picture of each flag for the countries listed, to help distinguish between them
  - **Estimated time:** 2 hours
  - **Actual time:** 1 hour
  - **Assumptions/Thoughts:**
    This task would involve using the ‘Country’ API we found. We made the assumption our API would have a URL to a flag image (which it luckily did). This was the same process as User Story #2 (but display an image instead of text), so we got to reuse most of the code from ‘Dishes/Dish’ files.
- **Phase 1: User Story 5 (Add picture for recipe)**
  - As a food blogger, I want to make sure that what I am eating will look nice on my Instagram page. A picture of what the recipe will look like will be helpful.
  - **Estimated Time**: 2 hours
  - **Actual time:** 1.5 hours
  - **Assumptions/Comments:**

This story would be User Story #4 but replicated for ‘Recipe’ models. We would just need to retrieve the ‘Recipe’ data and change the data we are displaying.

Phase 1 Customer Stories

- **Navigation Bar**
  - As a user of the website, I would like to see a navigation bar to surf the different models and instance pages on the website. It would be more appealing for the navigation bar to be space-themed (in terms of font and color). Make sure the navigation is in a user-friendly location.
- **Logos**
  - As a user, I would like to see the logos of the tools that were used in the process of building the website. It would make your ‘About’ page more UI presentable and eye-catching. Also, as a user, one would be able to quickly recognize the tools by the logos.
- **As a user of the website, I would like to see a title of the website on the home page**
  - It would be beneficial to get a general idea of what the website is about when clicking on the landing/home page. You could have some text displaying a summary of the website (space theme) or display the website's name in a more eye-catching way. Also, I would like to see the Splash page when first clicking on aboveearth.me and see the About Us page in a different tab (would need a navbar).
- **Images for expeditions**
  - As a user, I would like an image on an expedition page such as the space-craft. It would help me identify the aspects of an expedition. Also, it would be more visually appealing in terms of representing the data related to the space-craft.
- **As a user of your website, I would like to click on a logo of an agency to see more details about that agency**
  - As a user of the website, I would like to see the agency logo on the expedition page and be able to click on the agency logo to see more

detailed information about that agency. This would be a fun way to access your instance page about each agency.

Phase 2 User Stories

- **Phase 2: User Story 1 (Map of City)**
  - It would be useful if you put a map of the city on the associated city page. This way, users can instantly recognize where the city they are browsing food for is located in the world.
  - **Estimated time:** 2h
  - **Actual time:** 1h
  - **Assumptions:**
    We can implement the user story by using the Google Maps API.
  - **Implementation:**
    We embed the Google Maps API for each city instance page. It is also interactable and includes directions.
- **Phase 2: User Story 2 (Insert card images for city dishes)**
  - On the city pages, it would be helpful if you could insert images of the dishes as cards underneath a "Dishes from this city:" section. I would like to be able to see the examples of the different dishes offered in this city.
  - **Estimated time:** 3h
  - **Actual time:** 2h
  - **Assumptions:**
    We think we can get a list of dishes for each cuisine associated with the city. Then, we just get the first dish name and its image.
  - **Implementation:**
    We included a dish and its image on the bottom of the page by referencing the cuisine's first dish.
- **Phase 2: User Story 3 (Cuisine on restaurants table)**
  - On your restaurants table, it would be useful to include the type of cuisine as a column, so users scrolling through the site can browse based on what cuisine they are in the mood for.
  - **Estimated time:** 10m
  - **Actual time:** 2m

    - **Assumptions:**
      We can get the restaurant's cuisine type using the same data to populate the restaurant information.
    - **Implementation:**
      We added a new column for cuisine type by getting the restaurant's cuisine types and filling it in the table.
- **Phase 2: User Story 4 (Restaurant Directions)**
    - As someone looking to try one of the restaurants listed on the restaurant's page, it would be convenient to link directions from something such as google maps on how to get there. It would be helpful so that way I can directly navigate to the restaurant without having to change websites.
    - **Estimated time:** 2h
    - **Actual time:** 1h
    - **Assumptions:**
      It would be satisfactory to the user if we embed the Google Map code from our city instance page and place them in the restaurant instance page since it includes the directions.
    - **Implementation:**
      We placed the embed Google Maps code at the bottom of the page for now using the code from the city instance page. The only difference to the code, of course, is the restaurant address
- **Phase 2: User Story 5 (Improve about page aesthetics)**
    - I struggle to read websites where the information isn't well laid out and well-formatted. This reduces my attention span. For the bottom half of the about page, the cards would look good with increased margins and headings. Starting at APIs and going downwards, add space between the different cards and add whitespace in between sections to break up the page. It would also look nice if the white backgrounds for the cards were removed to make them look more transparent. Another thing you could do is expand each team member's card to better fit the page.
    - **Estimated time:** 10h
    - **Actual time:** 8h
    - **Assumptions:**

We can meet the user’s needs by changing the parameters for our card sizes, etc. until it “looks good”.

- **Implementation:**
  We implemented what the user requested and they said it looks good. However, we felt that if we switched the white background for the cards to be transparent, it would not look very good. We contacted the user and they said it was fine.

Phase 2 Customer Stories

- **Splash Page**
  - As a picky user, I would like to see the bottom text of the page to be a little smaller. Also, clicking on the about button, when not on the top of the page, maintains the spot where I am in the side-scroll bar. I would like to see it fixed. Finally, I would like to see the descriptions of your models on the splash page to reflect the models themselves, instead of "The technic-utopia is awakening. What is imagined?"
- **Add more links to buttons and cards**
  - As a picky user, I would like to be able to click on a button or card and have it direct me to a link. The "Visit Postman API" button doesn't work. Also, it would be useful for me to be able to click on your tools card and direct it to the documentation for each one. Also, "Go to Article" button in your instance pages redirects to the API instead of the actual article.
- **Not enough Sortable/Searchable instances**
  - As a user, I would like to see more information about each expedition/agency/news story to fully gain a grasp on the scope of the particular expedition/agency/news story. It would also be more convenient for the user to have more attributes to search/sort through as well. Having more information available would help the user get a better connection mentally of what instance connects to what other instance (for a different model).
- **Links to other model instances**
  - As a user, I would like to see the connection between the three models to make my understanding of the expedition/agency/news stories better. It

    - would be convenient to get information about the agency from the expedition/news stories instance pages or vice-versa for the other models. This would also make it easier to explore all instance pages without relying on the navbar or the individual model pages.
- **As a user of your website, I would like to click on a logo of an agency to see more details about that agency**
    - As a user of the website, I would like to see the agency logo on the expedition page and be able to click on the agency logo to see more detailed information about that agency. This would be a fun way to access your instance page about each agency.

Phase 3 User Stories

- **Phase 3: User Story 1 (City instance page format)**
    - We think it would make the city instance pages more readable, and less lengthy if you could put the recipe cards in a grid. Furthermore you could use columns when displaying your text information. This way not everything will be vertically aligned on the page!
    - **Estimated time:** 3h
    - **Actual time:** 4h
    - **Assumptions:** We assumed that we can use tables and carousels to make the longer texts scrollable and the images of the cuisines more visually appealing.
    - **Implementation:** We used tables on top of a card element and a carousel to display the city instance information and the images for the cuisines. We utilized the default carousel and table classes from bootstrap and added CSS styling to meet the requirements of the user.
- **Phase 3: User Story 2 (Rounding decimals on City model/instance pages)**
    - We think it would look more pleasing if you could round decimals throughout your City model and instance pages to a standardized place (either hundredths or thousandths etc).
    - **Estimated time:** 10m
    - **Actual time:** 5m
    - **Assumptions:** We think we can use the function toFixed() to round our decimals into a pleasing number. Typescript should have this function.

- ➢ **Implementation:** We used the toFixed() function wrapped around the parseFloat() function to implement our user story. The end result should satisfy the user.

- ❖ **Phase 3: User Story 3 (Clickable Cards on model pages)**
  - ➢ It would be better UX if the full card on your model pages (recipe and city) would be clickable/route to the instance page.
  - ➢ **Estimated time:** 10m
  - ➢ **Actual time:** 5m
  - ➢ **Assumptions:** We were initially considering wrapping the whole card into an <a> tag, and changing the css styling to prevent its default blue colour from being displayed.
  - ➢ **Implementation:** On each of our model pages cards, we wrapped the entire thing with an <a> tag that routed to that specific instance when clicked.
- ❖ **Phase 3: User Story 4 (Flexible Card Sizes)**
  - ➢ When looking at your second page in the Cuisines' model page, there's a lengthy card underneath Caribbean. The shorter cards next to it look awkward with trailing whitespace. This can be avoided by making card sizes flexible to their content.
  - ➢ **Estimated time:** 4h
  - ➢ **Actual time:** 3h
  - ➢ **Assumptions:** We assumed that we could utilize some form of CSS to make the Caribbean and Scandinavian cards flexible.
  - ➢ **Implementation:** We used the collapse class from bootstrap to make the Caribbean and Scandinavian cards flexible. In order to collapse, we needed to import JQuery to make the “Show More” and “Show Less” links work in expanding and shrinking the card size.
- ❖ **Phase 3: User Story 5 (Pagination: Choosing how many items to display per model page)**
  - ➢ As a user we would like to choose how many items per page to view. Perhaps this could be completed with a dropdown option where we can choose to display either 5, 10, 15... items a page.
  - ➢ **Estimated time:** 30m
  - ➢ **Actual time:** 20m

- ➢ **Assumptions:** We thought of using a useState/setState web hook to keep track of the variable numItemsPerPage.
- ➢ **Implementation:**
- ➢ For the city and cuisine models, we keep a numItemsPerPage state variable (that will be either 6,12, or 15). We added a dropdown menu at the top of the cities and cuisines models pages so that they can choose between 6, 12, and 15 number of items to display on each page (the menu calls an onClick that updates the numItemsPerPage). The frontend relies on this variable to know how many instances to show per page.

  For the restaurant model, we used the MUIDataTable and it has a rowsPerPageOptions. This option allows us to display the number of rows per page at the bottom of the table (10, 20, or 50).

Phase 3 Customer Stories

- ❖ **Inconsistent card sizes**
  - ➢ As a user, I want to see that your cards are sized properly. For your Expeditions and News Stories model pages, the cards on the last page are not sized properly. They are much wider than the previous cards. I would be satisfied as a user if you made all the card sizes in your model pages to be the same.
- ❖ **Fix Splash Page**
  - ➢ As a visiting user, it would be nice to have the splash page a little bit more refined. For some reason, I am able to drag and drop the image that is being animated on the page as shown below. Another thing is that the width of the page is too wide. I am able to scroll horizontally as you can see below and it shows the page cut off. My assumption is that if you have a user working on another monitor, then it will look different for a user that is using a laptop.
- ❖ **Missing data for Agencies**
  - ➢ As a user, I want to go to your website and get all the information I need for every agency. Some of your data is missing (i.e.

https://www.aboveearth.me/agencies/205) and in the Agencies model pages. Fill in the missing data from other sources and I will be satisfied.

❖ **Interactive Maps in Expedition Instance Page**
  ➢ As a user, I'd like to be able to interact with the map under your Expedition instance pages. Having just an image doesn't give me anything, except for showing me the surrounding areas. If I want to visit the site, I would want to be able to get directions from this site. Embedding a Google Maps API would do justice here if you guys prefer

❖ **Hovering over Cards**
  ➢ As a user, it would be more visually pleasing if the cards weren't underlined when having my cursor hover over the card. On the Expeditions model page, when I hover over a card, all of the text is underlined. It would look nicer if you removed the underlining with CSS and maybe add some other type of styling like shadows, animation, or however you would like to do it.

❖ Restful API

Postman Link:
https://documenter.getpostman.com/view/15165948/TzJrBeeW
Each model has the option to GET ALL instances of that model and the option to GET ONE INSTANCE by the object id. Our three models are cuisine, cities, and restaurants. Cuisine has 'Country' information under it. Since some cuisines have the same country, we added a 'Country' endpoint as well.

Cuisine Endpoints

- GET All Cuisines
  https://www.culturedfoodies.me/api/cuisines

- GET details about a cuisine
  https://www.culturedfoodies.me/api/cuisines/id=<id>

Countries Endpoints

- GET All Countries
  https://www.culturedfoodies.me/api/countries

- GET details about a country
  https://www.culturedfoodies.me/api/countries/id=<id>

Cities Endpoints
- GET All Cities
  https://www.culturedfoodies.me/api/cities

- GET details about a city
  https://www.culturedfoodies.me/api/cities/id=<id>

Restaurant Endpoints
- GET All Restaurants
  https://www.culturedfoodies.me/api/restaurants

- GET details about a restaurant
  https://www.culturedfoodies.me/api/restaurants/id=<id>

❖ Models

Our three models are cuisine, cities, and restaurants. Detailed country information is contained within the Cuisine model.

City:
- Leisure & Culture
- Cost of Living
- Environmental Quality
- Safety
- Travel and Connectivity
- Population
- Summary
- Name

- State
- Timezone
- Latitude/Longitude
- Business Freedom
- Economy
- Housing
- Commute
- Education
- Healthcare
- Outdoors
- Taxation
- Internet Access
- Startups
- Venture Capital
- Tolerance

Cuisine (which includes country data):

- Name
- Country
- Capital
- Population
- Latitude/Longitude
- Area(sq km)
- Region
- Time Zones
- Translations
- Borders
- Dishes
- Subregion
- Alpha 3 Code

Restaurant:

- City
- Aggregate rating

        - Price range
        - Zip code
        - Average cost for two
        - Timings
        - Address
        - Cuisines
        - Name
        - Highlights
        - Phone number
        - Menu

Filterable/Sortable Attributes:

- **City:**
  State, Leisure & Culture, Cost of Living, Environmental Quality, Travel connectivity, Population
- **Cuisine:**
  Name, Country, Capital, Region, Subregion, Population, Area size
- **Restaurant:**
  Name, City, Aggregate Rating, Price Range, Average Cost For Two, Rating

Searchable Attributes:

- **City:**
  Summary, Name, State, Timezone, Latitude/Longitude
- **Cuisine:**
  Region, Time Zones, Population, Translations, Borders, Dishes
- **Restaurant:**
  Timings, Address, Cuisines, Name, Highlights, Phone Number

Media:

- **City:**
    - Image of city, maps location of city
- **Cuisine:**
    - Flag of cuisine’s country, map of cuisine’s country
- **Restaurant:**

    - Image of restaurant, maps of restaurant

Connections:

- **Cities:** Connects to cuisines because each city has cuisines available there. Connects to restaurants because each restaurant is located in a specific city.
- **Cuisines:** Connects to cities because each city has certain cuisines available. Connects to restaurants because restaurants have certain cuisine types they serve.
- **Restaurant:** Connects to cities because each restaurant is located in a city. Connects to cuisines because each restaurant has cuisine types.

❖ Phase II Features

Database:

We used AWS Relational Database Service to create our database using PostgreSQL as the engine. Using pgAdmin as our open source administration and development platform for PostgreSQL, we connected to our AWS RDS database through its endpoint, username, and password. In the backend, we connected to our AWS RDS database using Flask, allowing us to query our database using SQLAlchemy. Our API routes return formatted data using schemas for each of our tables in the database using Flask-Marshmallow.

Pagination:

We used Material UI's Pagination component for the model pages. We used React's useState webhooks to keep track of the *current page (*page_num, setPageNum). On the model page, we do an API GET <model> call (which returns a list of instances). Based on the *current page*, we know which instances to display by slicing the list of instances. When the user clicks on a new page, the pagination component updates the current *page value (*setPageNum).

❖ Phase III Features

The main features for Phase III were searching, sorting, and filtering. We chose to implement these features on the frontend for all of our three models. For the Cities and Cuisines model pages (which are displayed in grid format), we used state and effect webhooks to update the data being displayed. For the Restaurants model page (which is displayed as a table), we used MUI datatables.

On the Models Page: Cuisines and Cities

We keep two state variables to keep track of the data (*cities/displayed Cities*). *Cities* are all the cities from our API call (get all cities call). *DisplayedCities* keeps track of the data to actually display (the instances that match the filter/search query). The cuisines model follows the same implementation.

For cities and cuisines, we have a dropdown menu with all sortable attributes. For sorting, we keep a state variable **(SortingVar)** that keeps the sortingName (String) and ascending (boolean) for the model. When the user clicks on the sortable attribute on the dropdown, we update the **SortingVar** to have the new sorting field.

For filtering, we keep a state variable for each filterable attribute (filteringRegions, filteringSubregions) which is a list of strings (i.e. [“Asia”, “Europe”]). When the user clicks on a filtering menu, we update the filterableVariable for that attribute (update filteringRegions when they click on “Asia”).

For searching, we keep a state variable **searchQuery**, whenever the user types into the searchBar (detected by onChange), we update the **searchQuery.**

To support the combinations of search/sort/filter, we have a **useEffect** webhook that gets called whenever there is a change in the filtering/sorting/searching state variables. In this method, we iterate through *cities* (all instances of cities) while keeping a list of **correctCities** (subset of cities to now display)**.**

If the search query is not blank, we check if the city's searchable attributes include the search query. If there's a filterable attribute applied, we check if that cityInstance's filterable attribute (region is "Asia") matches a list of applied options (i.e. ["Asia", "Europe"]). If a cityInstance matches the search query and applied filters, then it gets pushed to **correctCities.** We used React's Highlight to highlight search results.

Afterwards, if a sort is applied, we sort the correctCities based on the state variable sortingVar. We then update the displayed data (state variable) to be **correctCities.**

On the Models Page: Restaurants

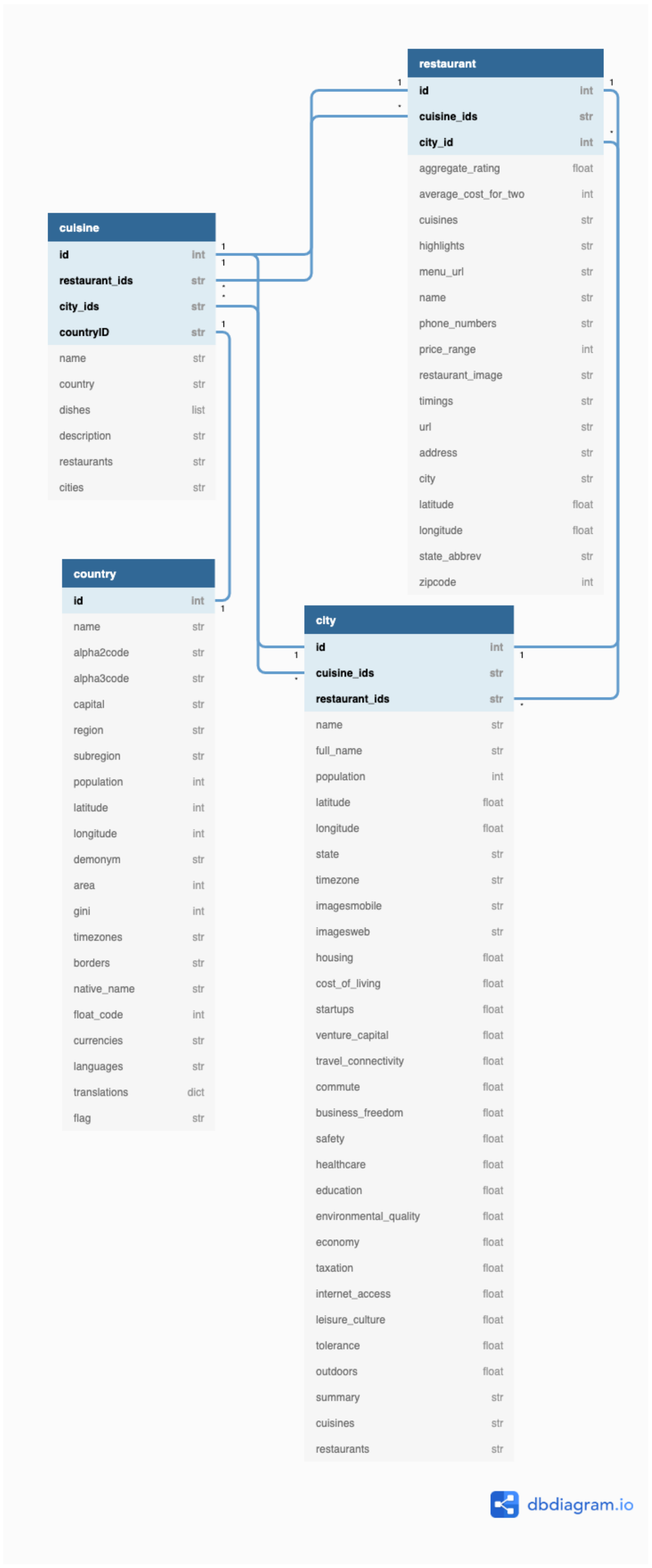
Restaurants are a table so we used MUIDataTable which handles pagination/search/filter/sort. The MUIDataTable accepts these properties: title, data, columns, and options. We set the title property to "Restaurants", linked the data property to our restaurants data, defined each field of restaurants as a column and handled its sorting and filtering using the columns property, and defined the options that our MUIDataTable can handle - such as sorting, filtering, and displaying a certain number of rows per page - using the the options property.

For searching, we keep a state variable **searchText**; whenever the user types into the searchBar and clicks the search button (detected by onClick), we update the **setSearchText.** In the options of our MUIDataTable, we set the option **searchText**, which is the search text for the table, to our state variable **searchText**. Each column's option has a **customBodyRender** where it returns a **<Highlighter>** of the **searchText** from the function **restaurantCustomBodyRender**.

On sitewide search

For sitewide search, we used Algolia and react-instant-search-dom's Instant Search to handle all of our results. On algolia, we have a cuisine index (which has cuisines and nested country data), cities index, and restaurants index with all of our data. We pass react-instant-search's Instant search component a search query and render the city/cuisines/restaurant components for the hits (based on which index it is).

❖ Database



restaurant	
id	int
cuisine_ids	str
city_id	int
aggregate_rating	float
average_cost_for_two	int
cuisines	str
highlights	str
menu_url	str
name	str
phone_numbers	str
price_range	int
restaurant_image	str
timings	str
url	str
address	str
city	str
latitude	float
longitude	float
state_abbrev	str
zipcode	int

cuisine	
id	int
restaurant_ids	str
city_ids	str
countryID	str
name	str
country	str
dishes	list
description	str
restaurants	str
cities	str

country	
id	int
name	str
alpha2code	str
alpha3code	str
capital	str
region	str
subregion	str
population	int
latitude	int
longitude	int
demonym	str
area	int
gini	int
timezones	str
borders	str
native_name	str
float_code	int
currencies	str
languages	str
translations	dict
flag	str

city	
id	int
cuisine_ids	str
restaurant_ids	str
name	str
full_name	str
population	int
latitude	float
longitude	float
state	str
timezone	str
imagesmobile	str
imagesweb	str
housing	float
cost_of_living	float
startups	float
venture_capital	float
travel_connectivity	float
commute	float
business_freedom	float
safety	float
healthcare	float
education	float
environmental_quality	float
economy	float
taxation	float
internet_access	float
leisure_culture	float
tolerance	float
outdoors	float
summary	str
cuisines	str
restaurants	str

Our database is based on our three models: Cuisine including their Country, City, and Restaurant. Each table has an id. Each Cuisine links to its respective Country id (countryID) and links to Cities (city_ids) and Restaurants (restaurant_ids) that have that Cuisine. Each City links to Cuisines (cuisine_ids) that it has and Restaurants (restaurant_ids) that are in the City. Each Restaurant links to Cuisines (cuisine_ids) it serves and the city (city_id) that they are in.

❖ Testing

Postman Tests:

For the Postman unit testing, we used the built-in Postman API testing feature within our workspace for each of the API endpoints. The main response attributes that were tested were whether the endpoint returned a valid status code (200) and whether all attributes returned in the response followed the same schema as defined on the database. We then exported tests from the Postman workspace into a JSON format file and then ran the file through Newman to retrieve the results.

Python Tests:

For the Python Tests, we used the unittest library to figure out whether data from our API endpoints were being pulled correctly. This test was added to our CI/CD pipeline and we used asserts to determine whether values are equal or not.

Jest Tests:

The Jest and Enzyme libraries were used to test our typescript code that runs the frontend of the site. We specifically use Jest's Snapshot testing that stores a returned instance of a specified page and stores it under the __snapshots__ folder. When we then run tests in the future, we compare our pages to the snapshots and make sure they are the same.

Selenium Tests:

For the Selenium Tests, we used Selenium and the associated Chromedriver to access our website directly from the driver. We directly used the HTML XPath attribute of the particular element and used those values to determine what particular element from the page we are retrieving. We also used the click() function to check whether links present on our pages were working. This test was added to our CI/CD pipeline as well.

❖ Tools

Frontend

- AWS S3- Used to host our static website
- AWS CloudFront - content delivery network
- React - A JavaScript library for building user interfaces
- BootStrap - Front-end development
- MaterialUI - React UI Framework
- Jest - Tests React and Typescript components and functions
- Algolia Search - Search UI library for site wide search
- Selenium - Tests GUI of website

Backend

- GitLab - GitLab Repository and CI/CI platform
- Postman - API and streamline collaboration platform
- NameCheap - Domain name registrar
- AWS ECR - Contains the docker image of our backend
- AWS Elastic Beanstalk - Used to deploy our backend
- AWS RDS - Storing our PostgreSQL Database
- PostgreSQL - Database management system used to store our tables

- pgAdmin - Administration and development platform for PostgreSQL

Hosting

We used NameCheap to get our domain name and added CNAME records from a certificate using AWS Certificate Manager for culturedfoodies.me and www.culturedfoodies.me. We used an S3 bucket from AWS to host our frontend React project and created a CloudFront distribution connected to our S3 bucket. Using our issued certificate, we were able to link our CloudFront distribution to our NameCheap domain and add a CNAME record that allows our domain name to point to our CloudFront distribution. We also added a URL redirect record so that our bare domain name would redirect to HTTP.

GitLab

Our GitLab repo can be found here.