Bib X Maitre-Restaurateur Project

Rémi PERRIER

Tables of Contents

Overview

All my source code can be found in the directory *source* and on my GitHub. To run the website by yourself use *npm run start* inside *my-app*. You don't need to install anything all dependencies are already here.

Server-side with Node.js

Scrapper for Michelin

Objectif

Get data on all French located restaurants with Bib Gourmand distinction.

Strategy

In order to scrap data on each restaurant pages we need these pages URLs. However, on guide.michelin.com the restaurant page URL are based on their name so there is no easy way to get them all.

To counter these problems, we can scrape the URLs too! Indeed, this website allows us to search for restaurants, so it's possible to get the wanted URLs by scrapping them on the different search result pages. What's more, by precising we want only restaurant with Bib Gourmand distinction in the search we only get the URLs of restaurant we want. It's a great gain of time.

The process has two steps. First scrape the search results pages to get the restaurant page URLs. Then, scrape the restaurant pages to get their data.

Implementation

All guide.michelin.com scrapping is done in server/*michelin.js.* The scrapping is done with cheerio and axios. The code is straightforward. You can find the result in *my-app/src/data/Michelin.json*.

Scrapper for Maître-Restaurateur

Objectif

Get data on Maître-Restaurateur restaurants.

Strategy

This time we can directly access to restaurant pages since there URL are indexed. However, there are some problems.

First, some index leads to nothing, it's probably restaurant that deleted their account. We need to take precaution with these cases. We will use these empty profiles to detect the end of our scrapping: when we encounter numerous empty pages in a row, we will stop.

Second, it's seems that some restaurant has multiples pages that are duplicates or near. We will clean our data once the scrapping is over.

This time the process is easier but takes a lot more time since we are scrapping a lot more of pages.

Implementation

All maître-restaurateur.com scrapping is done in *server/maitre.js*. The scrapping is done with cheerio and axios too. Again, the code is straightforward, and you can find the result in *my-app/src/data/maître.json*.

Intersection

Objectif

Compare the result of both scrapping and make their intersection.

Strategy

To get all Maître-Restaurateur with Bib gourmand distinction is a comparison between two list of items. We will consider two restaurants to be the same if there are similar names, to cover some various notations like the presence or not of apostrophe, and if there are in the same department.

Once a restaurant is identified in both lists, a new object is created with the more significant data resulting from both scrapping. Now a restaurant is a name, an address (street, city, department), a location (longitude, latitude), a telephone and a website URL.

Implementation

All code for this intersection is present in *server/bib.js* and will write its results in *my-app/src/data/maitrebib.json*.

Client-side with React

Restaurant card with Bootstrap

Objectif

Display restaurant information on a webpage with a bit of design.

Strategy

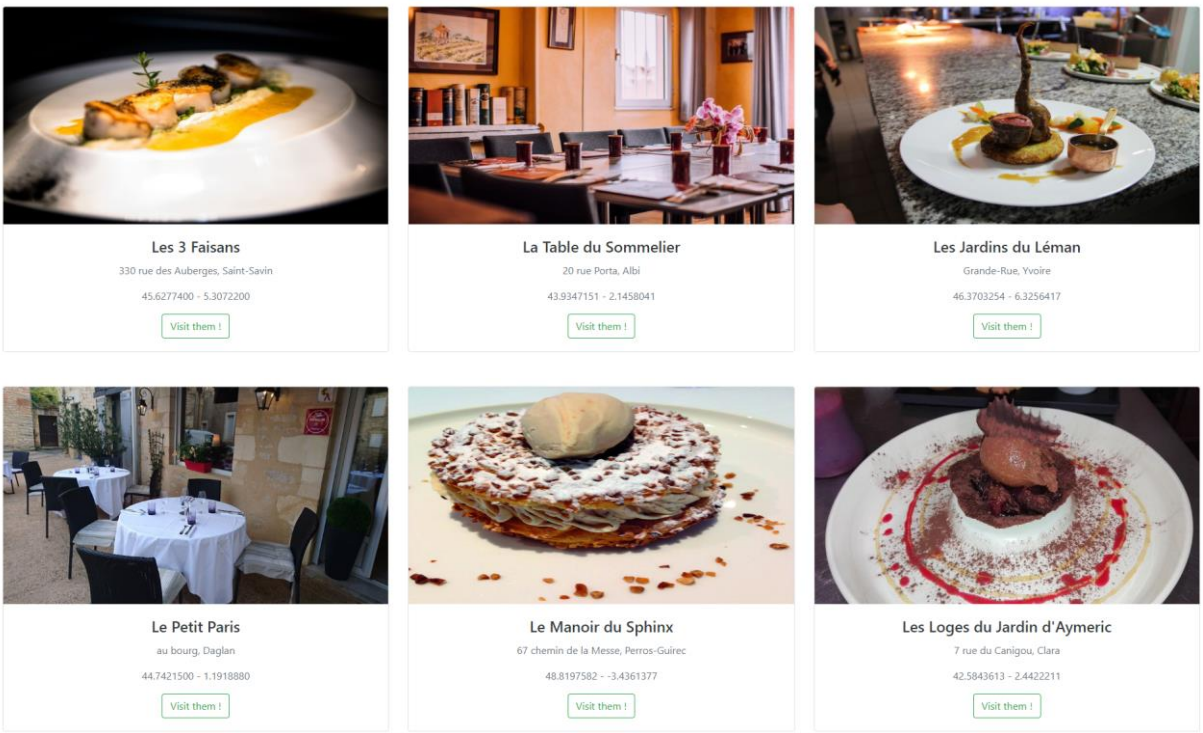
In order to create a responsive website, we will use React.js and Bootstrap to make simple card. To create a grid of card each corresponding to a restaurant we will use two classes: RestaurantCard and Cards.

RestaurantCard is used to transform a restaurant JSON object to a Card object. By using different Bootstrap options we can obtain a esthetic card. Cards class is used to create and manage all the different cards and order them in a grid. Then we simply render a Cards instance within App.js.

Implementation

RestaurantCard.jsx and *Cards.jsx* can be found in *my-app/src/cards*. RestaurantCard implement a single function that returns the HTML for a restaurant card fill with necessary information. Cards is more complex because there are additional functions for searching. However, only the render function is needed to display the cards.

Here is a preview of what you can see on the website at this point.



Each card contains a photo of the restaurant, its address, its longitude and latitude (we will later try to display the distance!) and a link to the restaurant webpage. Most links work great, but a few raise a security error, there must be a problem with the page they try to send the user.

Search function

Objectif

Display a search bar and allows user to search restaurants by name.

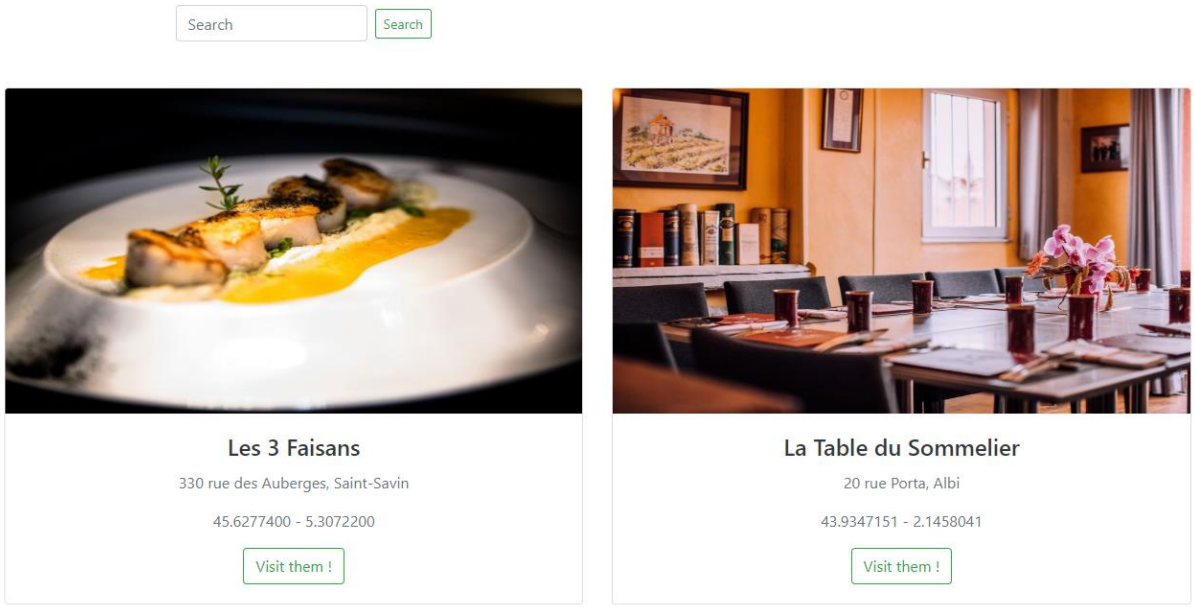
Strategy

To display and manage the search bar we will create a new class SearchBar. This class will define the design of the research bar and communicate to Cards on the user input. Then Cards will update its list of displayed restaurants to correspond to the user need.

Implementation

While the strategy looks simply, I failed to implement it. It seems there are some problems of communication between my two classes. What's more, in order to use setState I need to call WillMount but even after calling it its display an error message saying that I haven't. I am missing something here in order to make it work.

I left the code I add even if it's not useful and I rechange the displayed list of restaurants to the initial list. Here is a look of the search bar.



Distance sort

Objectif

Get user location and sort displayed restaurant by distance.

Strategy

The user location is obtainable thanks to the **react-geolocated** library. The distance can be computed with the **geolib** library. To sort displayed restaurants we need to sort the restaurant list used by the Cards instance.

Implementation

Sadly, I haven't be able to implement this feature due to short deadlines.