Projeto final

2021/2022

Extreme Weather Conditions Dashboard

Sérgio Oliveira - 2049719

ÍNDICE

Introdução

O aquecimento global é a maior ameaça à saúde que a humanidade enfrenta nos dias de hoje, e milhões profissionais de saúde espalhados por todo o mundo respondem diariamente aos danos à saúde causados por esta crise em desenvolvimento.

O *Intergovernmental Panel on Climate Change* (IPCC) concluiu que, para evitar impactos catastróficos na saúde e evitar milhões de mortes relacionadas às mudanças climáticas, o mundo deve limitar o aumento da temperatura a 1,5°C. As emissões passadas já tornaram inevitáveis um nível de aumento da temperatura global e outras mudanças no clima. No entanto, cada décimo de grau adicional de aquecimento afetará seriamente a vida e a saúde de toda a humanidade.

Embora ninguém esteja imune a estes riscos, as pessoas cuja saúde está sendo prejudicada primeiro e pior pela crise climática são as pessoas que menos contribuem para as suas causas, e que menos são capazes de se proteger a si mesmas, e às suas famílias contra estas alterações climatéricas. Estas são pessoas em países e comunidades desfavorecidas.

Esta crise climatérica ameaça desfazer os últimos cinquenta anos de progresso no desenvolvimento, saúde global e redução da pobreza, e ampliar ainda mais as desigualdades de saúde existentes entre e dentro das populações. Compromete gravemente a realização da cobertura universal de saúde (CUS) de várias maneiras – inclusive agravando a quantidade de doenças existente, e exacerbando as barreiras existentes ao acesso aos serviços de saúde. Mais de 930 milhões de pessoas - cerca de 12% da população mundial - gastam pelo menos 10% de seu orçamento familiar para pagar os cuidados de saúde. Com as pessoas mais pobres em grande parte sem seguro, os choques de saúde levam anualmente cerca de 100 milhões de pessoas à pobreza, com os impactos das mudanças climáticas piorando essa tendência.

Eventos climáticos de risco em Portugal

Ao considerar o caso específico de Portugal, observamos uma alta correlação entre a taxa de mortalidade e as condições meteorológicas extremas (ou seja, ondas de frio e calor). Vários fatores podem explicar esta correlação. Entre eles, é de mencionar que a maioria dos edifícios portugueses são antigos e não cumprem os requisitos mínimos relativos às condições de conforto térmico.

Outro aspecto que merece destaque é a renda média da população. Em Portugal, o salário médio é mais baixo do que grande parte dos outros países europeus, e apesar do custo de vida ser relativamente baixo, o preço da energia é mesmo assim consideravelmente alto. Por esta razão, as pessoas muitas vezes evitam o uso de dispositivos elétricos de aquecimento ou refrigeração, expondo-se assim a condições desfavoráveis.

Muitos acabam por preferir optar por soluções mais baratas - como o uso de lareiras para aquecimento, pois a madeira é mais barata do que a eletricidade. No entanto, as lareiras libertam CO2 dentro de casa, expondo os ocupantes ao risco de morte durante a noite devido à inalação de CO2, que é um gás especialmente perigoso por não possuir cheiro e ser incolor.

Outro fator de risco é a taxa de envelhecimento em Portugal, que está a aumentar. Os mais idosos estão mais expostos a temperaturas extremas porque seu sistema imunológico é menos eficaz do que o dos mais jovens, e são frequentemente afetados por condições de

saúde adicionais associadas ao envelhecimento. Além disso, muitos idosos vivem sozinhos e portanto, acabam por ter dificuldades em receber ajuda em caso de necessidade.

Solução

O projeto de final de licenciatura que me foi proposto pelo professor Filipe Magno consistiu, em colaboração com a Universidade de Lisboa, na continuação da criação de um website que permite a visualização do clima.

Para tal, foi me disponibilizada a tese de mestrado realizada pelo colega Luís Carlos Gonçalves Freitas, ex-aluno de Engenharia Informática. Foi-me também disponibilizado todo o código desenvolvido pelo colega, e tive a oportunidade de agendar uma chamada com o mesmo, de forma a clarificar a implementação por ele desenhada.

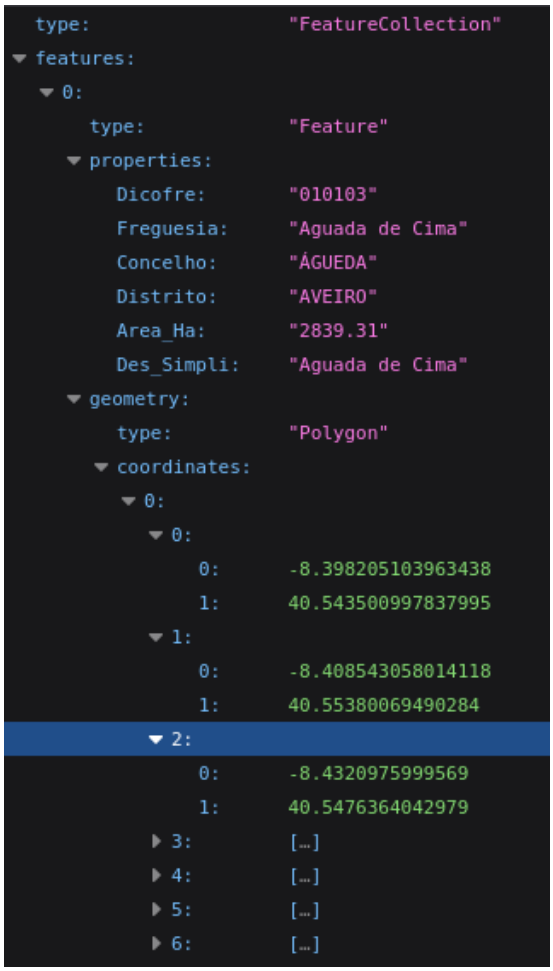
Este website teria como objetivo não só a visualização do clima, mas um *backend* suficientemente robusto para a elaboração de estatísticas relacionadas com a previsão de riscos naturais, como ondas de calor e frio.

Conceitos importantes

Antes de começar a explicar o projeto, é necessário explicar dois conceitos intrínsecos ao mesmo, que são os conceitos de GeoJSON, e de Coordinates Reference System (CRS).

GeoJSON

Um GeoJSON é um documento que representa várias regiões de um mapa. Este documento possui todos os pontos que delimitam uma região. Possui também várias propriedades de cada região. Estas propriedades variam conforme cada GeoJSON.



Um GeoJSON é sempre composto pela chave-valor **type:FeatureCollection**, e por um array de *features.* Estas *features* por sua vez têm sempre o conjunto chave-valor **type:Feature**.

Cada *feature* tem uma *field* designada por **properties,** e estas propriedades podem ter qualquer par chave-valor. Cada feature tem também uma *field* designada por **geometry**.

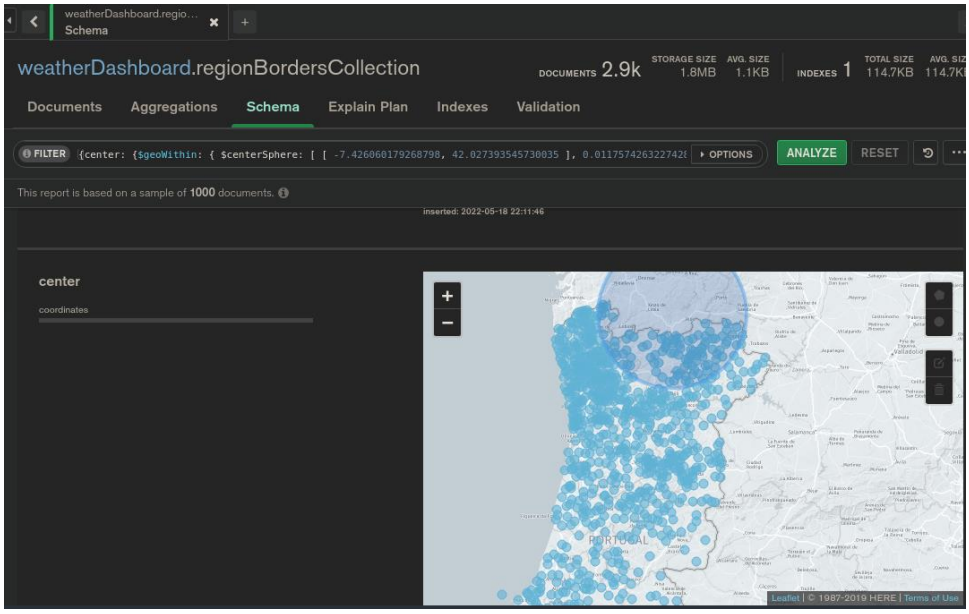
A geometria é composta por um tipo, que pode ser **Polygon** ou **MultiPolygon**. Uma região composta apenas por um último polígono é do tipo **Polygon**, contudo existem regiões **MultiPolygon** compostas por várias ilhas, ou outras regiões, como é o caso de França, que possui uma região na América do Sul chamada de Guiana Francesa.

Coordinates Reference System (CRS)

```
{
"type": "FeatureCollection",
"crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:OGC:1.3:CRS84" } },
"features": [
```

As coordenadas das regiões do geoJSON são dependentes do seu CRS (Coordinates Reference System). Por defeito, são coordenadas que seguem o standard **WGS84**, mas podem ser sistemas de coordenadas completamente diferentes.

O MongoDB consegue fazer queries com base em coordenadas, desde que estas respeitem o standard **WGS84**.



Projeto anterior

O projeto desenvolvido pelo Luís consistiu num *backend* programado em Javascript, utilizando Node.Js, onde a base de dados seria o MongoDB. O *frontend* foi desenvolvido em Vue.js, utilizando a biblioteca Leaflet.Js para a visualização do mapa com o clima. O Luís decidiu também separar a funcionamento da aplicação em 3 micro-serviços, cada um com um *container* de Docker separado, que seriam:

Dashboard – Container responsável por permitir ao utilizador visualizar o mapa com as informações climatéricas.

Data Sources Handler – Container responsável por permitir o upload de ficheiros com as informações das localizações.

Database - Container responsável pela base de dados.

Dificuldades encontradas

Ao configurar o projeto anterior deparei-me com um grande número de dificuldades, mesmo com a tese de mestrado disponível.

Em primeiro lugar, os ficheiros de configuração dos *containers* estavam errados, e necessitaram de algumas alterações para efetuar o seu *deploy*.

Em segundo lugar, acredito que a separação do projeto em três micro-serviços foi uma separação desnecessária, e que o projeto apenas necessita de seguir uma REST API, sendo dividido em dois componentes, um servidor para o *frontend*, e outro servidor que comunica com a base de dados e expõe numa API os dados necessários para o *frontend.* Esta nova abordagem reduz a complexidade do projeto, sem trazer consigo nenhuma desvantagem.

Em terceiro lugar, a maior dificuldade com o projeto anterior foi perceber a interação entre o container *Data Sources Handler* e o container da base de dados. Várias funcionalidades essenciais para o projeto teriam que ser feitas na forma de pedidos HTTP para este primeiro container, que por sua vez comunicaria com o container da base de dados. Um exemplo disto é adicionar os documentos com as coordenas das regiões, que teria de ser feito com uma ferramenta semelhante ao Postman:

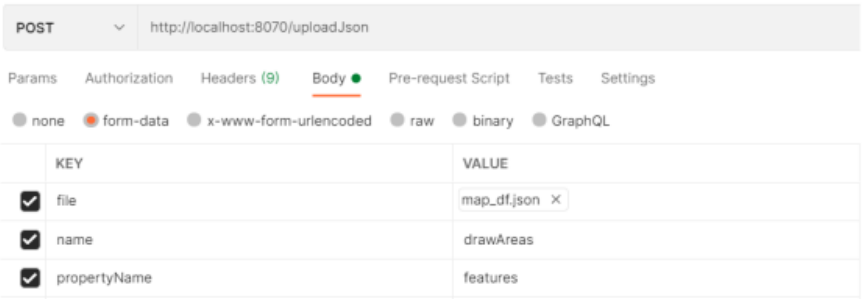


Fig. 33. Upload JSON File

Após adicionar as regiões, as suas respetivas coordenadas centrais individuais a cada região também tinham que ser calculadas pelo *backend,* na forma de outro pedido:

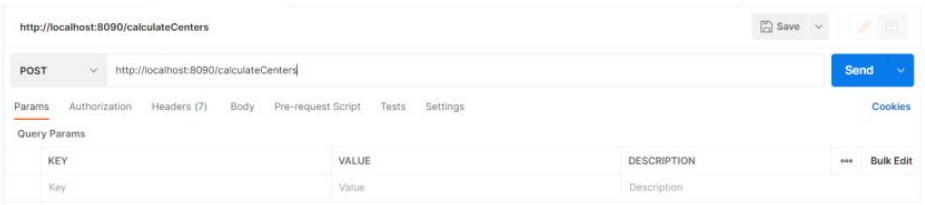


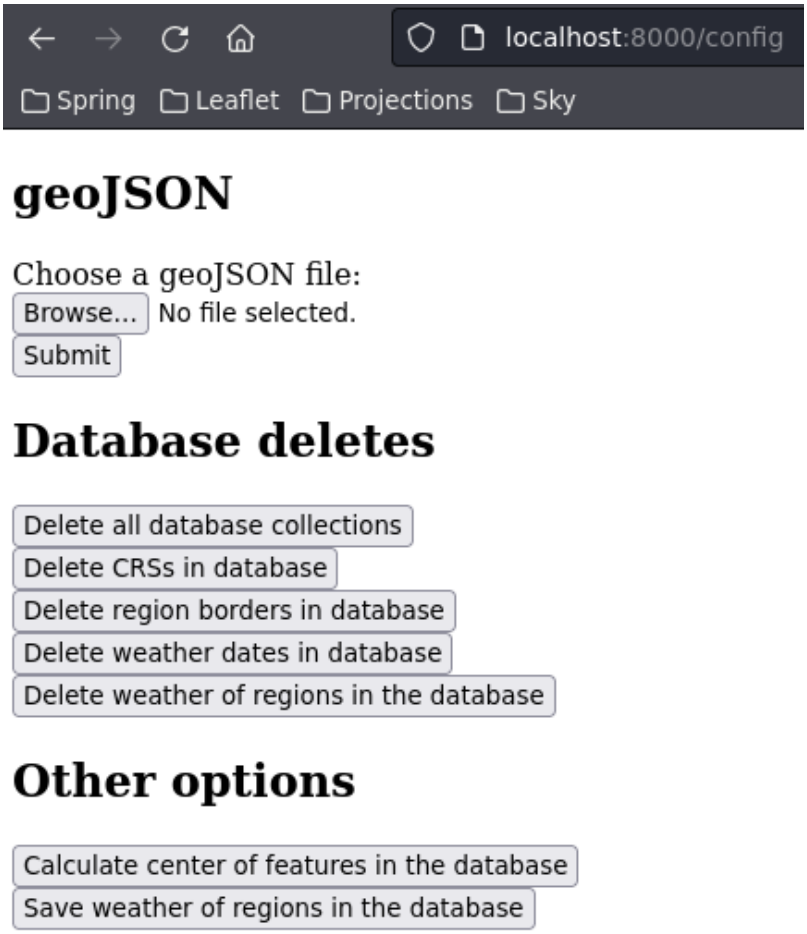
Fig. 58. Calculate Centroid Point

Resolução destas dificuldades

Tendo tudo isto em conta, o meu objetivo inicial foi efetuar um conjunto de alterações de forma a facilitar a configuração e o *deploy* do projeto.

Considerei também importante eliminar a dependência em ferramentas externas como o Postman para permitir a utilização do sistema por parte de utilizadores com poucos conhecimentos técnicos.

Criei então um painel de configurações que permite gerir todas as funcionalidades deste projeto.



Este painel de configurações permite a qualquer utilizador sem conhecimentos técnicos adicionar novas regiões, eliminar as que já se encontram na base de dados, calcular as coordenadas centrais de cada região e guardá-las na base de dados, e guardar o clima de cada região, bem como a data em que o clima foi gravado na base de dados. Este painel permite também mostrar algum erro que possa ocorrer no processo ao utilizador.

Desenvolvimento de um *backend* mais robusto

Ao analizar o *backend* anteriormente desenvolvido, foram encontradas e resolvidas várias limitações que não permitiriam escalar este projeto de forma adequada.

Suporte para novos Coordinate Reference Systems

O sistema original apenas suportava GeoJSONs cujas coordenadas se encontravam de acordo com o standard **WGS84**. Esta acabou sendo uma limitação muito grande para o funcionamento do projeto, pois não estão disponíveis na internet GeoJSONs com as regiões inteiras de Portugal. Seria também um obstáculo caso pretendêssemos aplicar o sistema a outra região. O código foi reescrito de forma a permitir a importação de um GeoJSON com qualquer **CRS**.

Suporte para vários Coordinate Reference Systems em simultâneo

Após reescrever o *backend* para suportar GeoJSONs com qualquer **CRS**, uma funcionalidade importante que foi implementada foi o suporte do sistema para importar vários GeoJSONs em simultâneo, cada um com um **CRS** diferente.

Isto permite ao utilizador visualizar no mesmo mapa, em simultâneo, duas regiões distintas importadas com diferentes sistemas de coordenadas.

Suporte para queries do MongoDB com base em coordenadas

Após programar a funcionalidade anterior, o sistema passou a suportar diferentes **CRSs** em simultâneo, no entando, é importante voltar a chamar a atenção que o MondoDB apenas suporta queries com base em coordenadas se estas coordenadas seguirem o standard **WGS84.**

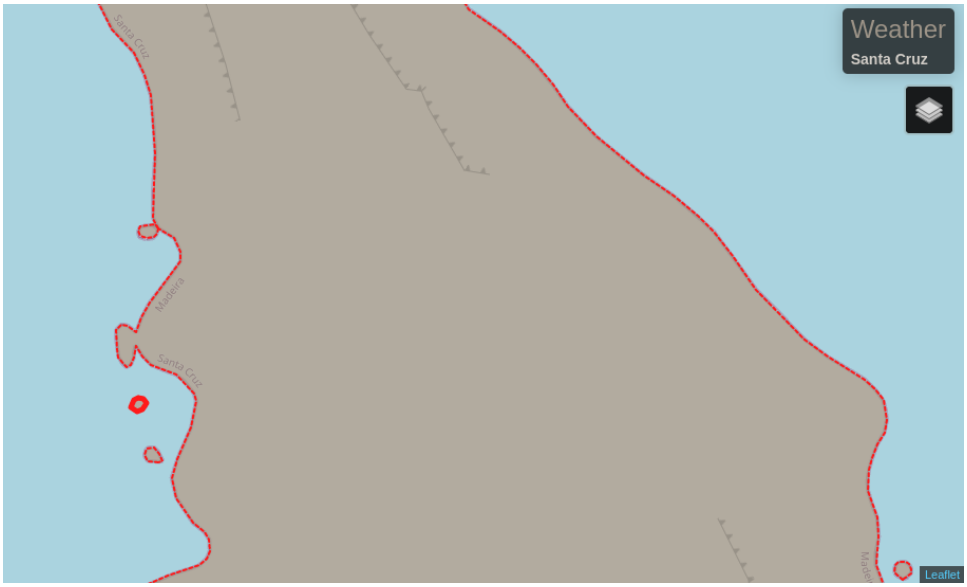
Após reescrever parte do funcionamento do programa, e alterar a arquitetura da base de dados, foi implementado um sistema de conversão de coordenadas, de forma a que qualquer GeoJSON gravado na base de dados possa ser alvo de queries com base nas coordenadas das suas regiões.

Suporte para regiões constituídas por vários polígonos

O sistema original tratava uma região constituída por vários polígonos como uma única região. Esta abordagem tem graves limitações, que serão clarificadas.

Como podemos verificar, existem casos onde a mesma região de um GeoJSON é constituída por polígonos geograficamente bastante separados, e com temperaturas bastante diferentes. O sistema foi programado de forma a ter este detalhe em conta, onde cada sub-região é tratada de forma individual.

Tendo em conta que o sistema encontra a temperatura de uma região com base no seu centro geográfico, o centro da região Francesa estaria situado entre o país e a Guiana Francesa, algo que não faria sentido.



Na prática, tratar todos os polígonos de uma região de forma individual acaba por não fazer sentido em todos os casos, como no exemplificado acima, mas é uma solução mais robusta do que a proposta inicialmente.

Implementação das novas funcionalidades

Estas novas funcionalidades exigiram uma reescrita completa do *backend*, por estarem diretamente ligadas à arquitetura da sua implementação.

O Vue.Js foi também uma escolha que limita futuramente o desenvolvimento do sistema, devido à fragmentação do ecossistema entre o Vue2 e o Vue3 no que toca a bibliotecas de mapeamento como o Leaflet.Js.

Tendo isto em conta, o *frontend* foi redesenhado em React, por ser de longe a framework de *web development* com mais adoção, e ao seu enorme ecossistema, quer seja a nível de documentação, facilidade de encontrar ajuda de outros utilizadores, ou às suas poderosas bibliotecas.

Tecnologias escolhidas

- O MongoDB foi mantido pela sua flexibilidade e escalabilidade, assim como pela sua funcionalidade de permitir queries com base em coordenadas.
- O Vue.Js foi substituído pelo React.Js, por ser uma *framework* com muita mais adesão, e por sua vez, mais funcionalidades. Acredito também que o React é um *framework* mais fácil de aprender, e que possui uma comunidade muito maior.
- Inicialmente, o código foi escrito em Javascript. Ao longo que o projeto foi desenvolvido, a *codebase* começou a se tornar mais difícil de manter. Reescrever o código em Typescript foi uma tarefa trabalhosa, mas sem dúvida a escolha certa a médio e longo prazo, e o código encontra-se agora muito mais fácil de manter.
- O Leaflet.Js é sem dúvida a biblioteca correta, pelo seu fácil uso, e suporte de GeoJSONs, que é essencial para projetos que pretendem mostrar estatísticas de certas localidades.

Abordagem utilizada no desenvolvimento das novas funcionalidades

O desenvolvimento das várias funcionalidades foi um processo iterativo, que implicou alterar várias vezes a arquitetura do sistema, e a estrutura da base de dados.

Coordinates Reference System

O Leaflet.Js apenas suporta GeoJSONs que aderem ao standard **WGS84**. Para permitir utilizar outros GeoJSONs, foi necessário implementar um sistema que converte qualquer tipo de coordenadas para coordenadas standardizadas.

Como referido, o sistema original apenas suportava GeoJSONs com coordenadas que seguem o **WGS84**, uma grande limitação.

Na minha primeira tentativa de resolver esta limitação, o sistema importava o GeoJSON, e gravava o seu sistema de coordenadas na sua tabela individual. O resto do conteúdo era gravado, sem modificações, noutra tabela. Nesta abordagem, os vários GeoJSONs eram enviados para o *frontend* com os seus respetivos **CRSs**, e o frontend era responsável por fazer a conversão.

Esta abordagem permitia ter vários **CRSs** no mapa em simultâneo, mas tinha uma limitação. Tendo em conta que as coordenadas eram gravadas na base de dados com **CRSs** que não o standard, o MongoDB não conseguia fazer queries com base nestas coordenadas.

Para tal, a responsabilidade da conversão dos **CRSs** foi movida do *frontend* para o *backend.* No momento em que um GeoJSON é importado, tem as suas coordenadas convertidas para o standard. Esta abordagem tem todas as vantagens da anterior, mas permitiu também reduzir o tamanho da base de dados, aumentar a velocidade do website, e permitir fazer queries com base nas coordenadas de qualquer GeoJSON da base de dados, independentemente do seu sistema de coordenadas original.

Separar as regiões constituídas por vários polígonos

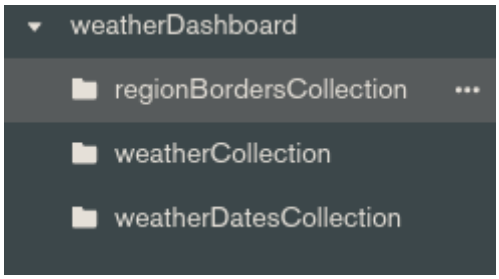
De forma a resolver o problema onde França teria a mesma temperatura de Guiana Francesa, que seria a temperatura de um ponto localizado algures no oceano Atlântico foi necessário a criação de um algoritmo.

Foi implementada uma função que identifica no GeoJSON as *features* que tem uma geometria do tipo *MultiPolygon*, e cria várias cópias dessa mesma feature, mas com os seus polígonos separados, e do tipo *Polygon.* Esta função é executada no momento em que um GeoJSON é importado para a base de dados.

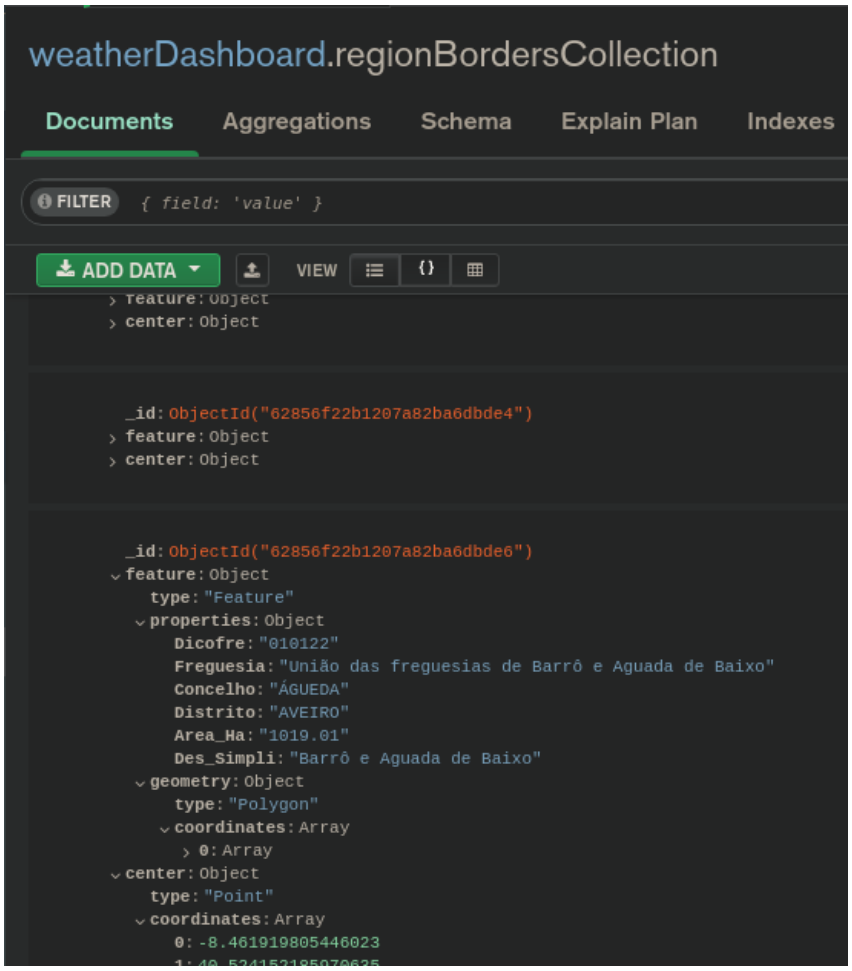
Arquitetura da base de dados

A arquitetura da base de dados é algo crucial para o funcionamento de qualquer projeto, e esta sofreu várias iterações, as quais foram documentadas anteriormente.

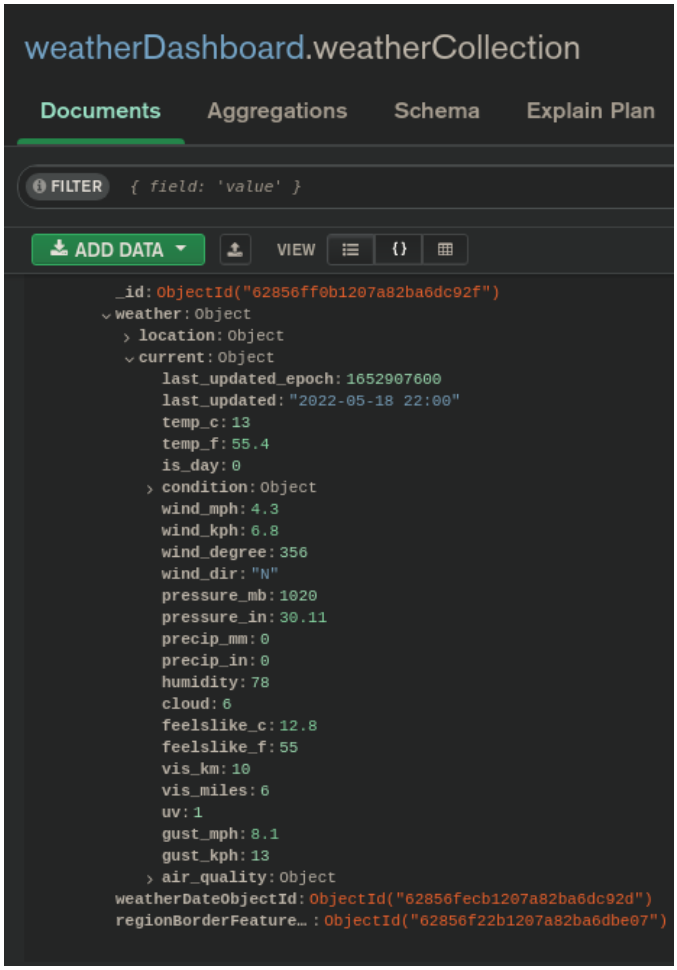
No fim, a base de dados ficou estabilizada, acabando assim na seguinte configuração.



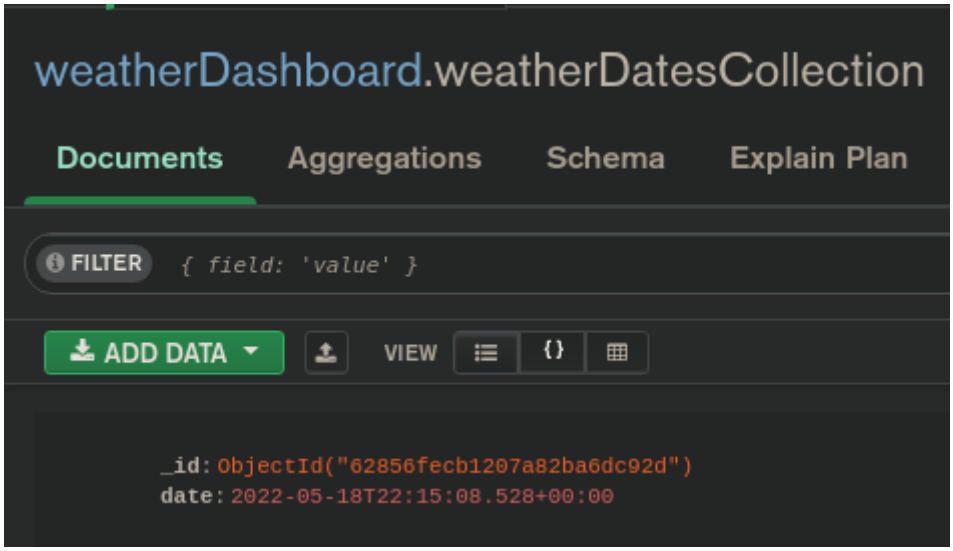
- **RegionBordersCollection –** Coleção com as várias regiões encontradas nos GeoJSONs importados.



- **WeatherDataCollection -** Coleção com as várias informações climatéricas recolhidas das regiões da tabela anterior.



- **WeatherDatesCollection -** Coleção com as datas em que as informações climáticas foram recolhidas.



Setup do projeto

A configuração do projeto implica criar um ficheiro **.env** no *backend* com os seguintes campos:

DB_USERNAME: Username do utilizador na base de dados

DB_PASSWORD: Password do utilizador na base de dados.

DB_URL: URL da base de dados.

DB_NAME: Nome da base de dados.

DB_WEATHER_DATES_COLLECTION_NAME: Nome da coleção onde as datas do clima serão gravadas.

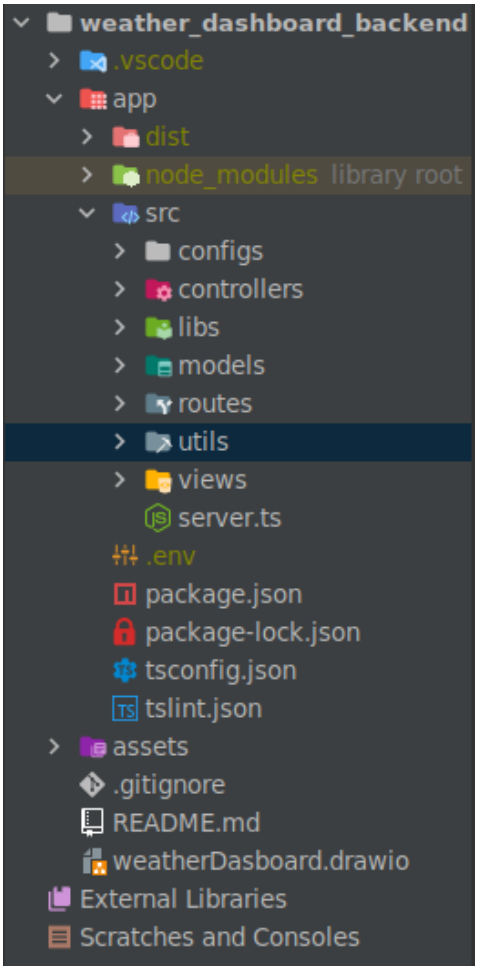
DB_WEATHER_COLLECTION_NAME: Nome da coleção onde o clima será gravado.

DB_REGION_BORDERS_COLLECTION_NAME: Nome da coleção onde as regiões importadas dos GeoJSONs serão gravadas.

WEATHER_DATA_PORT: Porte onde o backend será inicializado.

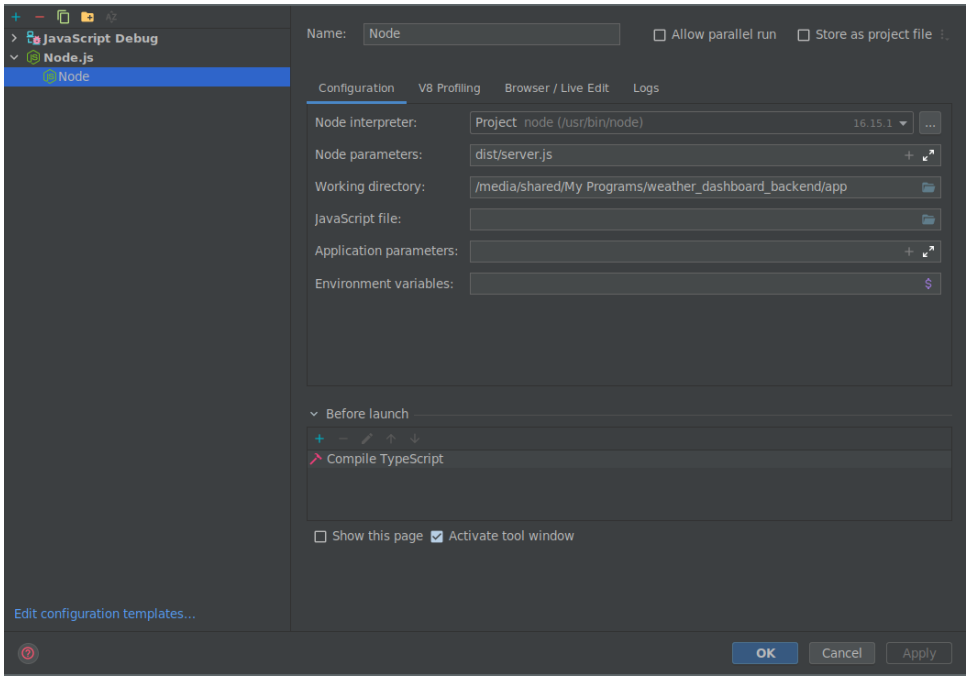
WEATHER_API_KEY: API Key do website api.weatherapi.com.

O ficheiro **.env** deve ser situado na seguinte localização:



Tendo em conta que o backend está escrito em Typescript, o mesmo deve ser *transpiled* para Javascript antes da sua execução. Este processo pode ser executado manualmente, seguindo as instruções encontradas neste link:
https://code.visualstudio.com/docs/typescript/typescript-compiling

Contudo, é vivamente recomendado o uso de um IDE como o WebStorm da JetBrains, onde poderá ser configurado este processo de forma a que seja executado automaticamente.


JavaScript Debug
Node.js
Node
Name: Node
Allow parallel run
Store as project file
Configuration
V8 Profiling
Browser / Live Edit
Logs
Node interpreter: Project node (/usr/bin/node) 16.15.1
Node parameters: dist/server.js
Working directory: /media/shared/My Programs/weather_dashboard_backend/app
JavaScript file:
Application parameters:
Environment variables:
Before launch
Compile TypeScript
Show this page
Activate tool window
Edit configuration templates...
OK
Cancel
Apply

Utilização do projeto

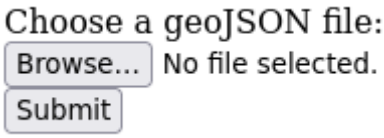
Backend

Após as configurações anteriores, o backend pode ser inicializado usando o botão de start do WebStorm 

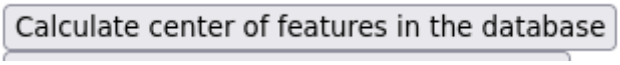
, ou o comando node <server.js> caso o código tenha sido anteriormente transpilado.

O utilizador pode então aceder ao painel de configurações do projeto, acedendo ao link http://localhost:8000/config.

Em seguida, o utilizador deve importar um GeoJSON à sua escolha utilizando o botão Browse, e fazendo submit após a sua escolha:



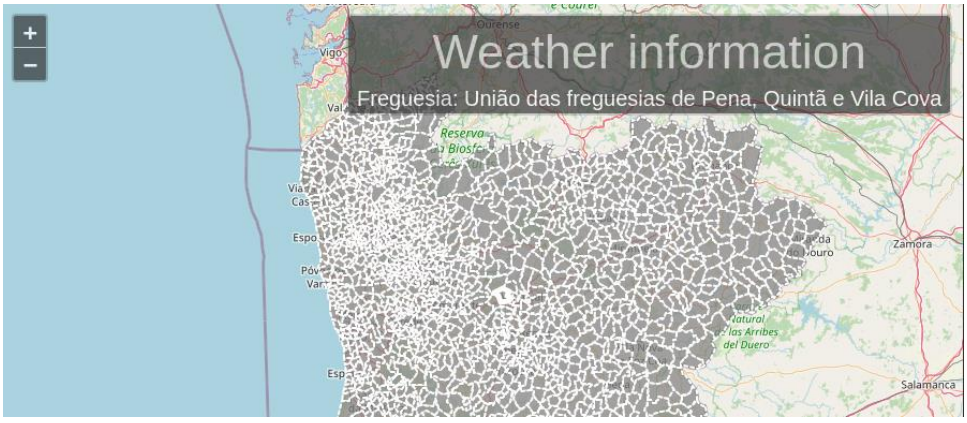
O utilizador deve então calcular os centros das regiões importadas utilizando o seguinte botão:



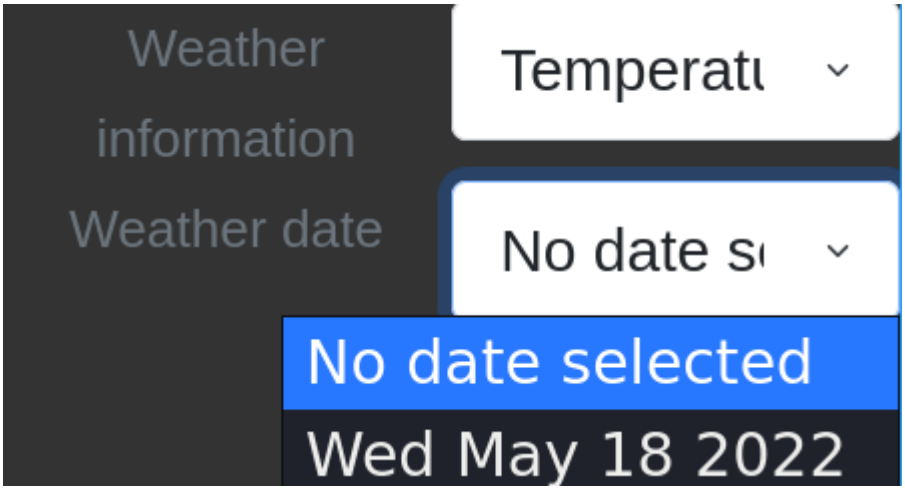
Após os centros estarem calculados, o utilizador pode gravar o clima atual das regões da base de dados usando o botão: 

Frontend

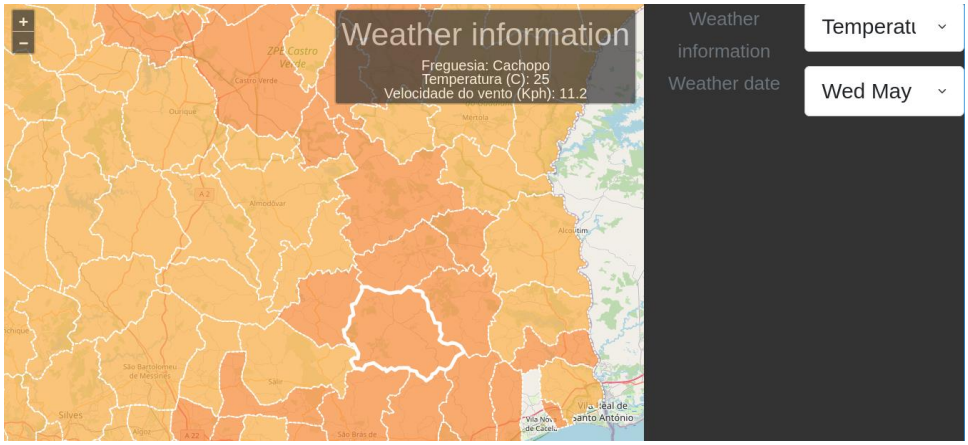
Após o utilizador ligar o *frontend* usando o comando **npm start**, caso já tenha importado algum GeoJson, as suas regiões aparecerão no mapa.



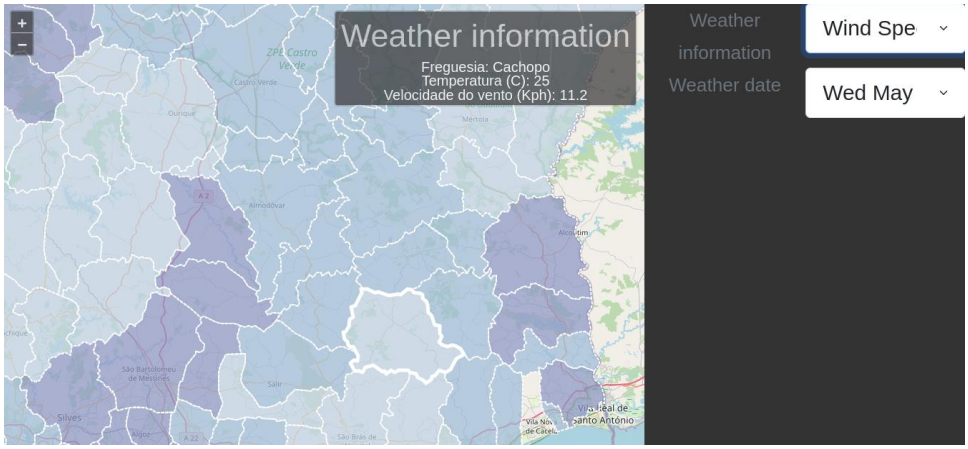
Caso o utilizador tenha decidido gravar o clima, as datas de quando o clima foi guardado aparecerão no menu lateral.



O utilizador pode então selecionar uma data, e a informação que quer mostrar no mapa, e a mesma será visualizada.



O utilizador pode selecionar diferentes tipos de dados, que serão mostrados com diferentes gradientes.



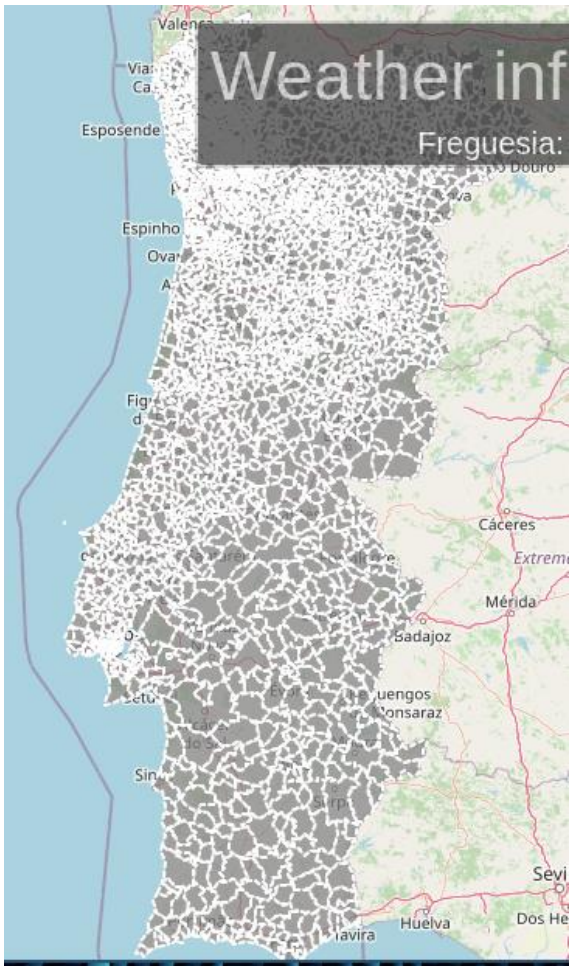
Futuras alterações ao sistema

Performance

O sistema encontra-se com bom nível de performance.

A nível do *backend*, o cálculo dos centros das regiões e os pedidos à API do clima, que são os maiores *bottlenecks* do sistema, utilizam threads de forma a maximizar a eficiência de ambos os processos.

O *frontend* não tem problema a aguentar com milhares de regiões em simultâneo, e consegue pedir as informações das regiões ao *backend* com um nível de performance aceitável.

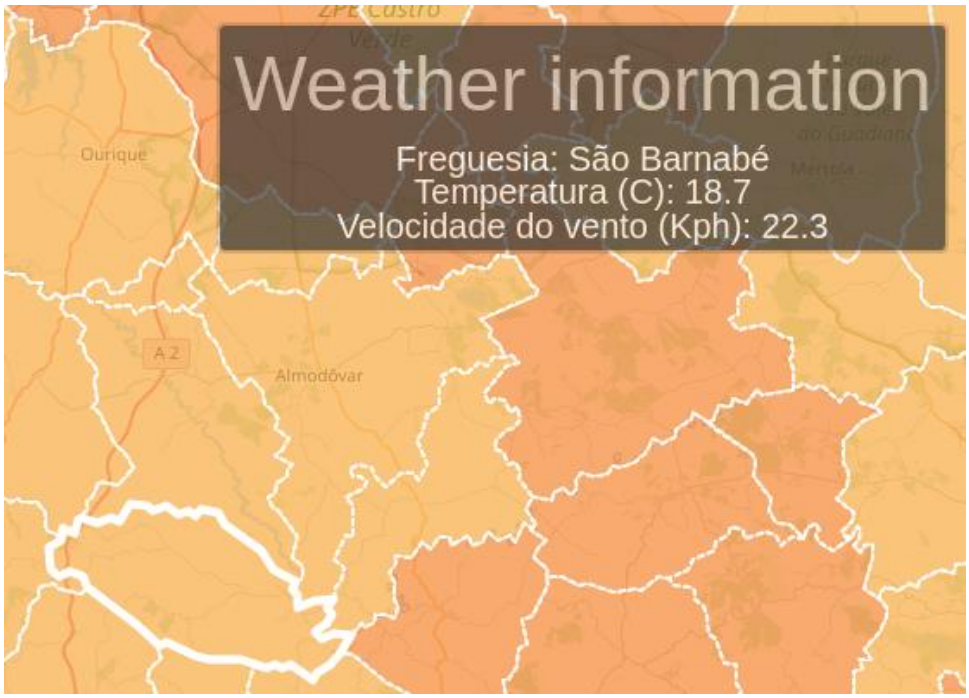


Caso fosse necessário renderizar milhões de regiões em simultâneo, a renderização do mapa teria que ser baseada em WebGL para manter um performance adequado.

Funcionalidades

O código do *backend* encontra-se robusto o suficiente para escalar facilmente o projeto sem alterações arquiteturais complexas. O projeto guarda as datas no momento da gravação, mas o código fácilmente poderia ser alterado para lidar com fusos horários separados, visto que a API assim os retorna.

A nível do *frontend*, a utilização de componentes de React permite a fácil extensão ou alteração do código existente.



Adicionar informação que o utilizador considere relevante ao painel de informação consiste em alterar o seguinte componente:

HoveredFeaturePanel.tsx

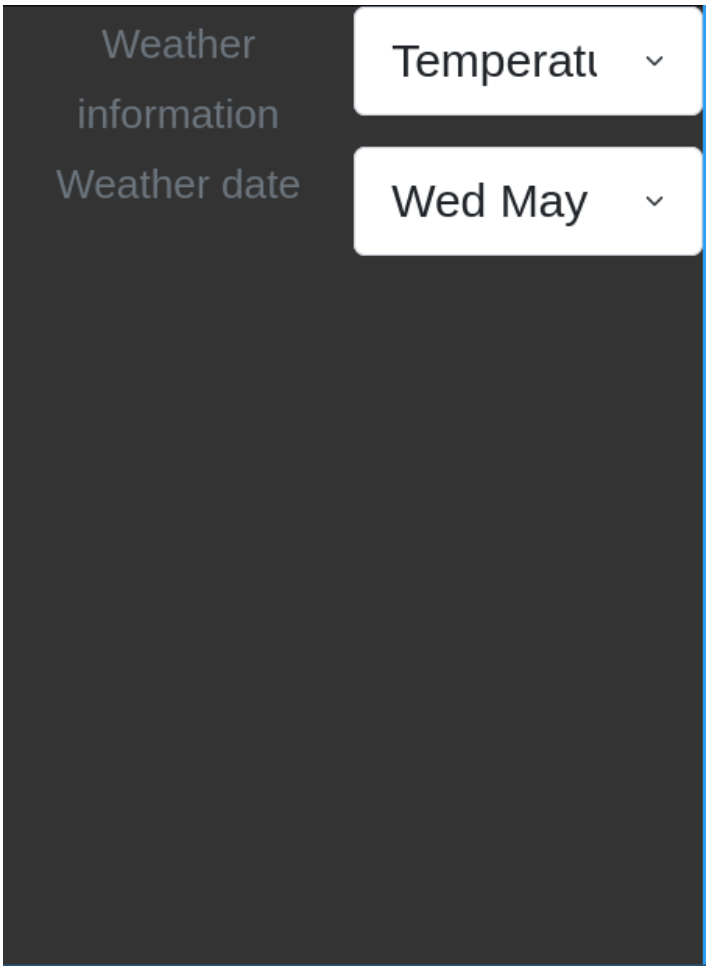
```
41      let windSpeed: JSX.Element = <div/>;
42      if (featureProperties?.weather) {
43          // @ts-ignore
44          windSpeed =
45              <div style={{fontSize: 20}}>
46                  {/*@ts-ignore*/}
47                  Velocidade do vento (Kph): {featureProperties?.weather.current.wind_kph} <br/>
48              </div>
49      }
50
51      return (
52          <div className={POSITION_CLASSES["topright"]}>
53              <div className="leaflet-control leaflet-bar MapFeatureInformation">
54                  <h4>Weather information</h4>
55                  {freguesia}
56                  {pais}
57                  {temperature}
58                  {windSpeed}
59              </div>
```

O código permite também adicionar de forma rápida novos gradientes ao mapa conforme o tipo de informação escolhida pelo utilizador:

GeoJsonLayer.tsx

```
201  function getTemperatureColor(temperature: number) {...}
222
223  /** Returns the wind speed of a given color. ...*/
227  function getWindSpeedColor(windSpeed: number) {
228
229      if (windSpeed > 40) return "#4d004b"
230      if (windSpeed > 35) return "#810f7c"
231      if (windSpeed > 30) return "#88419d"
232      if (windSpeed > 25) return "#8c6bb1"
233      if (windSpeed > 20) return "#8c96c6"
234      if (windSpeed > 15) return "#9ebcda"
235      if (windSpeed > 10) return "#bfd3e6"
236      if (windSpeed > 5) return "#e0ecf4"
237      if (windSpeed > 0) return "#f7fcfd"
238      return "#4d004b"
239  }
240
```

Foi deixado um espaço no painel do utilizador, com a ideia futura de adicionar gráficos com estatísticas das áreas selecionadas pelo utilizador:



Conclusão

Acredito assim que foi criado com sucesso um sistema robusto o suficiente para a visualização de qualquer conjunto de dados geográficos, quer climáticos ou de outra natureza.

O sistema foi desenhado com o objetivo de ser o mais flexível possível, de forma a permitir quaisquer alterações futuras. Este sistema pode, por exemplo, ser alterado facilmente de maneira a interagir com outro API, como um que retorna os casos de covid de uma determinada região.

A elaboração de estatísticas pode também ser futuramente desenvolvida, na forma de um serviço no *backend* que corre periodicamente, e compara certos dados em diferentes datas da mesma região.

Referências

https://www.who.int/news-room/fact-sheets/detail/climate-change-and-health

https://www.w3.org/2015/spatial/wiki/Coordinate_Reference_Systems#WGS84

https://vue2-leaflet.netlify.app/

https://blog.mastermaps.com/2013/06/converting-shapefiles-to-topojson.html

https://gis.stackexchange.com/questions/81508/leaflet-geojson-webpage-performance-out-of-hand-how-to-convert-to-png

https://leafletjs.com/index.html

https://mapshaper.org/

https://gis.stackexchange.com/questions/189126/leaflet-geojson-with-projected-coordinates-3857

https://colorbrewer2.org/#type=sequential&scheme=BuGn&n=3

https://github.com/kartena/Proj4Leaflet

http://jsfiddle.net/nathansnider/ru7oyquq/

http://kartena.github.io/Proj4Leaflet/

https://www.section.io/engineering-education/building-simple-maps-using-leaflet-js/

https://blog.mastermaps.com/2012/10/mapping-regions-of-new-zealand-with.html

Extreme Weather Conditions Dashboard MASTER DISSERTATION - Luís Carlos Gonçalves Freitas