ISITGETTINGHOTORISITJUST.ME

Anthony Moeller, Celik Savk, David A. Neveling, Ramon Fabrega, Supawit Chockchowwat, Teo Jun Hui

Introduction

The world today is facing a slew of environmental challenges, such as global warming, overpopulation and resource depletion, which taken together have significantly compromised human welfare. A variety of organizations and charities have sprouted up to tackle some of these challenges, but many find it difficult to gain traction among the general population.

Motivation

In spite of a wealth of information available on our environmental problems, and a multitude of work of charities and environmental organizations to address these challenges, this information is often disparate. Thus, knowledge of environmental issues alone, may not necessarily translate into meaningful action, especially if people do not know how to start.

This web application seeks to consolidate the various publicly available information on climate change to demonstrate the breadth and severity of environmental challenges confronting each country, in a bid to galvanize further grassroots action on environmental protection.

It is intended to educate the general population about the reality of our environmental woes while providing a platform for them to channel their concern into meaningful tangible actions.

By presenting people with a list of charities that have a presence either in their country of interest or in a field users can most identify with, it is hoped that this will encourage more people to rally around the cause of environmental conservation.

User Stories

1. **Embedded donation to charities:** Embed a donation box for the charities you list so visitors to your site can directly donate to them, if possible.
    a. Added a donation link to the charity JSON object. However, the donation link still needs to be scraped automatically. Time took: 10 minutes.
2. **Rank Countries by EPI:** Rank the countries by EPI as described in #20 (closed) so users can see which countries are best addressing their environmental issues and which ones need the most help (maybe these need more donations to charities).

    a. The API supports requesting country by its order. So the only thing to do is store countries in the database in the EPI order. Estimated time: 20 minutes.
3. **Rank environmental issues:** Environmental issues can be ranked according to how severely they impact the public. For example: air quality is a more pressing issue than biodiversity.
    a. The severity of environmental issues are subjective, however, if the customer has a statistic to determine it like the number of countries affected, then as noted in (2) current API supports retrieving by order. Time took: 5 minutes.
4. **Filter charities by country:** Allow users to search for pertinent charities by afflicted country.
    a. It can be done by doing a GET request on the endpoint /country with the country's name. The data it returns contains the list of charities of said country. Time took: 0 minutes as was already implemented.
5. **Add EPI for each country:** For each country you list, report its Environmental Performance Index which outlines how well that country's policy addresses its environmental problem.
    a. Added EPI field to each country on the API specification. Time took: 10 minutes.
6. **Making the Countries Instance Pages Cleaner:** This is similar to the sorting issue. So far, each Country's instance page just has a list of statistics that go way off screen. It would be nice if we do not have to scroll down so far just to see current Emission data. It would look nicer maybe if the tables were horizontal or something. Also, what is the unit for the Emission data? Please specify it like the Temperature.
    a. Page recreated to display issues as buttons in card and display temperature and rainfall data in graphs.
7. **Multimedia on Country and Issues Pages:** While these instance pages have a lot of data, it seems to be lacking any media to keep the users engaged. It would be nice to have some pictures or something on these instances.
    a. Graphs of temperature and rainfall data are now implemented, later I'll add the emission data back as a graph but its format was a bit different from the other two. This should help give the users a much easier time reading and understanding the data, later on if there's time I may add a basic regression to observe trends in data. Once the API is implemented I'll also incorporate a basic country profile with the flag above the issue card.
8. **Filter Info By Year:** So far the data itself seems good. It would be nice to be able to view the data for a specific year instead of having to scroll down to it on each country's instance page.
    a. Because the data is now displayed on a graph rather than a poorly formatted table, this shouldn't be necessary, all the data is displayed in an easy to observe format.

9. **API Call for Getting Countries by Name:** Maybe there can be an API call to get Country information by country name. It would be convenient if you did not know the ISO code.
    a. Modified the current API specification to allow requests by country names. However, it is not yet implemented in the Backend.
10. **API Functions/Parameters to Sort Data:** It would be nice if you can choose what order to sort the data by in the API, such as Z to A on country names or lowest to highest on EPA.
    a. Modified the API specification to support these. However, not yet implemented it on the backend.
11. **Filter by type of Environmental Issues**: The type of issues presented is very broad. It'll be good to categorize them, e.g. land issues, sea issues, etc.

a. Since there are no data sources to group those issues into types, this is not doable.

12. **Countries sort by alphabetical order:** This would let users find countries more easily.

a. Added to the API specification and implemented.

13. **Countries filter by continent:** Namely Africa, Antarctica, Asia, Australia, Europe, North America, South America.

a. API allows for filtering based on the latitude and longitude which approximates continent information.

14. **Charities filter by country of establishment:** e.g. US, UK, etc.

a. Current data source - charity Navigator - only contains charities that are in the US.

15. **Search bar for charity modal page:** Please add a search bar for users to search for a specific charity. It should be able to match the searched strings with one or several charity names.

a. Implemented.

RESTful API

The API is documented here using Postman. The API is composed of GET requests and the arguments are passed as GET parameters and the API returns a JSON object as described in the documentation.
When designing the API our main consideration was how the front end was going to use it. There are two scenarios when the frontend needs data from backend:

1. In order to render the tables for individual models there are two requests done for it:
    a. Frontend first requests metadata for the models using and uses the returned count of instances to calculate the number of pages.
    b. Uses the current page number with the number of instances returned at (a) to calculate the order of instances it is going to display on the current page.

i.e. If current page number is 9 and the number of instances are frontend will request instances [10,11,12,13,14,15].

c. Requests data about the instances using the API.
i.e For the example above frontend does the following GET request: api.isitgettinghotorisitjust.me/<model name>?query=[10,11,12,13,14,15].

2. In order to render information about a single instance:
   a. Frontend requests data for the instance using the instance's name. API returns data about the instance that contains statistics, links, text, and references to related instances of other models.
   b. The data contains references to related instances of other models. Frontend queries the API to get data about those instances.
   i.e. If we are trying to render the page about the USA then it will contain cross-references to issues and charities. We do the following call to get data about issues:
   api.isitgettinghotorisitjust.me/issue?query=["Desertification", "Air pollution"]

We wanted the API to be easily usable and robust so instead of splitting query by order and query by string we made them a single query. You are even allowed to interleave orders and names in the same query. i.e. have query=["USA", 4]. Because of the use case in (#1) we chose to return a list instead of a dictionary because the output needs to be ordered. Also, to make the API more robust we chose to not return an error code in case of a faulty entry but put the error message to the corresponding list entry in the output.

Get Data About Countries

JSON Object for each country looks like this:


```
{
   "country_code": "USA",
   "country_name": "United States of America",
   "issues": ["Desertification","Water pollution" ],
   "charities": ["SoundWaters","Island Press" ],
   "temperature": [
     [1920, 1939, 4.05357345619076],...
   ],
   "rainfall": [
     [1920, 1939, 790.6361028238144],...
   ],
   "emission": [
     [1960, 15.9997791565885],...
   ],
   "EPI": 71.19,
   "longitude": "-97",
   "latitude": "38",
   "Flag_link":
```

"https://api.backendless.com/2F26DFBF-433C-51CC-FF56-830CEA93BF00/473FB5A9-D20E-8D3E-FF01-E93D9D780A00/files/CountryFlagsPng/usa.png"
}
Here is what each field corresponds to:

- **country_code:** ISO3166 alpha 3 country code of the country. Type: string
- **country_name:** English name of the country. Type: string
- **issues:** A list of environmental issue names that occur in that country. This data is gathered from https://www.cia.gov/library/publications/the-world-factbook/fields/2032.html. Type: list of strings
- **charities:** The list of charities that operate in this country. Type: list of strings
- **temperature:** A list of average temperatures of 20 year periods of the country. The unit of temperature is in Celsius. The data is gathered from https://datahelpdesk.worldbank.org/knowledgebase/articles/902061-climate-data-api. Type: list of lists of format [start year:int, end year:int, temperature:float]
- **rainfall:** A list of the average rainfall of 20 year periods of the country. The unit of rainfall is in millimeters. The data is gathered from https://datahelpdesk.worldbank.org/knowledgebase/articles/902061-climate-data-api. Type: list of lists of format [start year:int, end year:int, rainfall:float]
- **emission:** A list of average annual CO2 emission amounts of the country. The unit of CO2 emission is metric ton per capita. The data is gathered from https://data.worldbank.org/indicator/EN.ATM.CO2E.PC . Type: list of lists of format [year:int, emission:float]
- **EPI:** Environmental Protection Index of the country. Not yet implemented. Type: float
- **longitude:** Longitude of the country. Type: float
- **latitude:** Latitude of the country. Type: float
- **Flag_link:** Link to the country's flag. This data is gathered from https://fabian7593.github.io/CountryAPI/. Type: string

Get Data About Issues

JSON Object for each issue looks like this:

{
"issue_name": "Desertification",
"url": "https://en.wikipedia.org/wiki/Desertification",
"summary": "Desertification is a type of land degradation in which a relatively dry area of land becomes increasingly arid, typically losing its bodies of water as well as vegetation and wildlife...",
"images": "https://upload.wikimedia.org/wikipedia/commons/e/ec/Cabrasnortechico.JPG",
"countries": [

"SAU",...
],
"charities": ["Huron River Watershed Council"]

}

Here is what each field corresponds to:

- **issue_name:** Name of the environmental issue as the title of the page in Wikipedia. Type: string
- **url:** Link to the environmental issue's Wikipedia page. Type: string
- **summary:** Summary of the environmental issue as taken from Wikipedia. Type: string
- **images:** Link to an image of the environmental issue as taken from Wikipedia. Type: string
- **countries:** List of ISO3166 alpha 3 country codes of the countries that have this issue. This data is gathered from https://www.cia.gov/library/publications/the-world-factbook/fields/2032.html. Type: list of strings
- **countries:** List of charities that work on that issue. Type: list of strings

Get Data About Charities

JSON object for each charity looks like this:

{
"charity_name": "World Forestry Center"
"tagLine": "Promoting sustainable forests",
"website": "http://www.worldforestry.org",
"mission": "Founded in 1964 in Portland, Oregon, the World Forestry Center works to educate and inform people about the world's forests and trees, and their importance to all life, in order to promote a balanced and sustainable future. We do this with three programs: Discovery Museum, tree farms, and the World Forest Institute. Our 20,000 square foot Discovery Museum is located in Portland's beautiful Washington Park. All new hands-on, interactive exhibits are family friendly and designed to engage visitors to learn about the sustainability of forests and trees of the Pacific Northwest and around the world. The World Forest Institute was established in 1989 as the information services division of the World Forestry Center.",

"address": {
"country": US,
"stateOrProvince": "OR",
"city": "Portland",
"postalCode": "97221",
"streetAddress1": "4033 Southwest Canyon Road",
"streetAddress2": null
},

```
  "countries": [
    "MLI",
    "NCL",
    "NZL"
    "PNG",
  ],
  "issues": [
    "Pollution"
  ],
  "image": "https://logo.clearbit.com/http://www.worldforestry.org",
  "donation_link":
"https://www.charitynavigator.org/index.cfm?bay=my.donations.makedonation&ein=936034757
&"
 }
```


Here is what each field corresponds to:

- **charity_name:** Name of the charity. Type: string
- **tagLine:** A statement that concisely describes the mission of the charity involved. Type: string
- **website:** Link to the website of the charity. Type: string
- **mission:** Description of the main activities of the charity. Type: string
- **address**: Provides the location of the charity in the US. Type: dictionary
- **countries:** List of ISO3166 alpha 3 country codes of the countries that the charity has operations in. Type: list of strings
- **Issues:** List of issues the charity works on. Type: list of strings
- **image:** Link to an image of the charity if one exists. Type: string
- **donation_link:** Link that points to the charities donation address. Type: string

Models

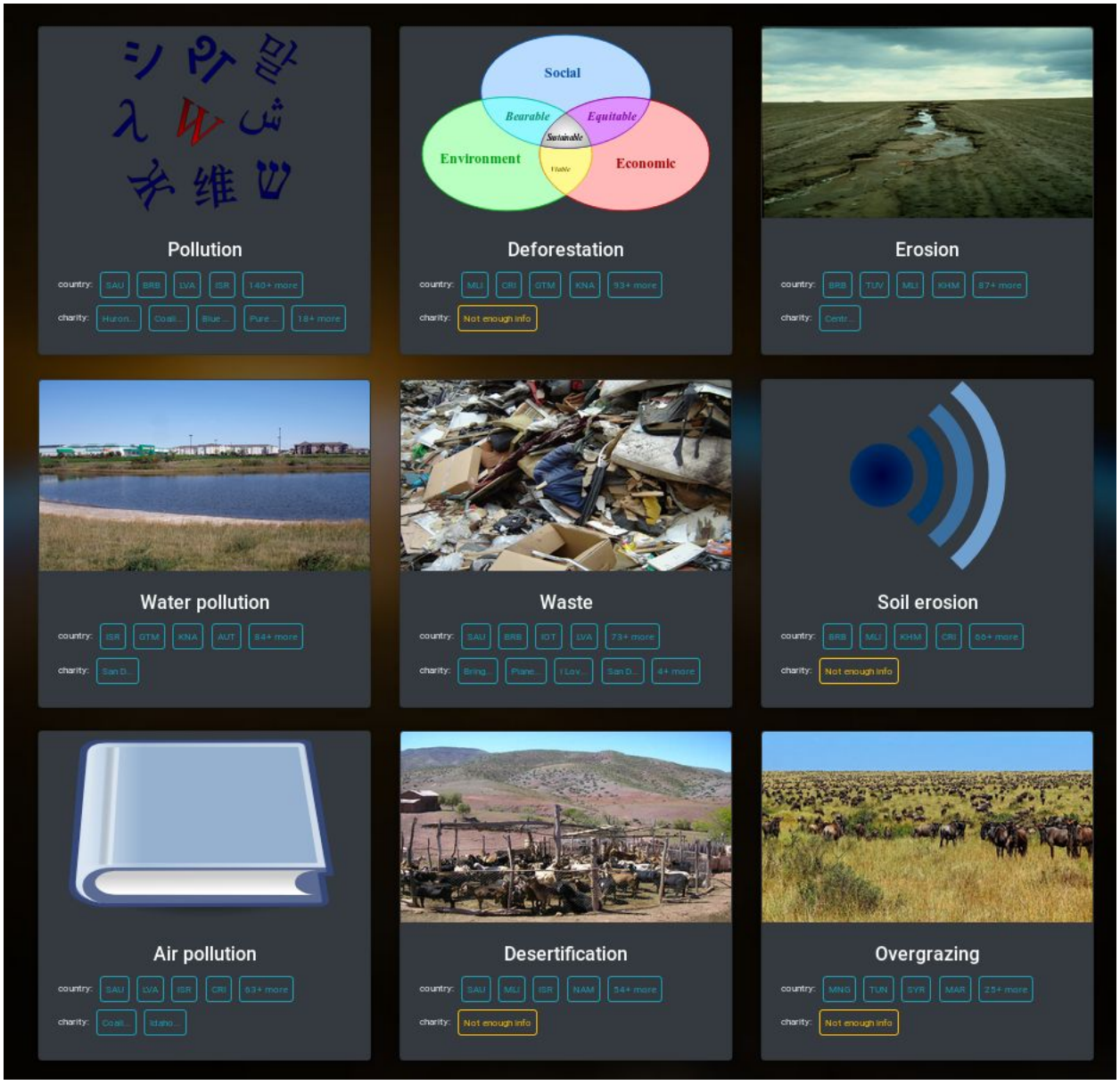
Countries

The countries model is comprised of each country, listing its name, and giving an overview of the environmental issues plaguing it, alongside both temperature and carbon emissions data over the past several decades. The countries model is related to the issues model, by the issues field, which lists the environmental issues that the country currently faces. The countries model is further linked to the charities model in that for each country, we can obtain the list of charities that operate in it. The country model is similar to the issue model listed below in the way it links to both charities, issues, and the table of countries that you can select.

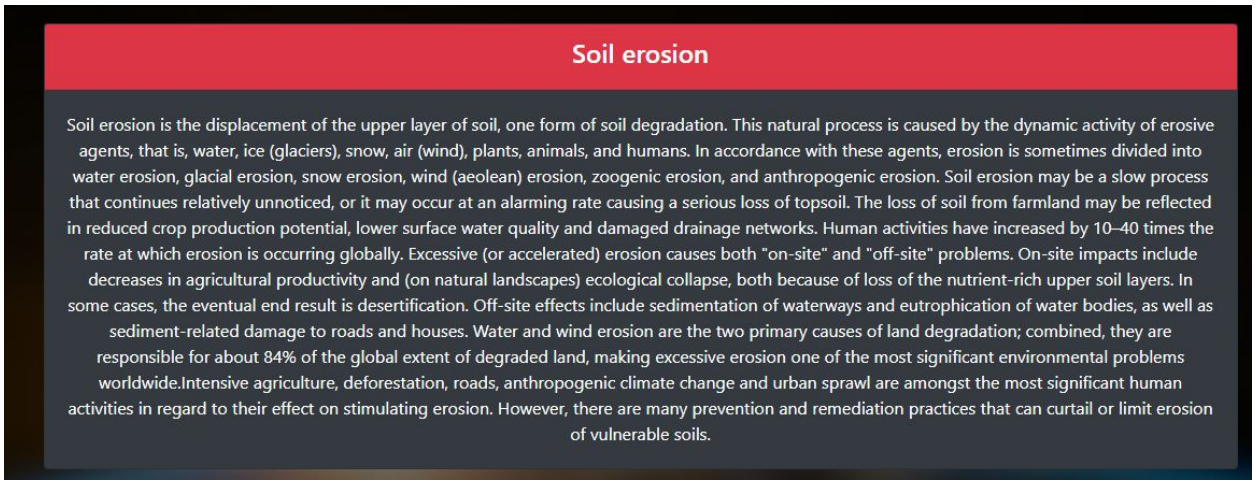
Issues

The issues model comprises of a description of the environmental issues as its key, and a list of countries facing the issue, as a link to the countries model.

The user begins with a screen showing them a collection of environmental issues.



They can then choose a card sending them to the information page of their issue of choice. Once on their selected page, a summary of their selected issue is presented.



Immediately underneath the summary, are a list of countries affected by the selected issue and charities that help offset the issue.

Each country and charity is shown as a card in a collection.

Each of these cards links to the corresponding page with that country's information.

Charities

The charities model has the name of the charity as its key, and essentially contains details of the mission of the charity, and further links to its image and website, allowing for a link to the

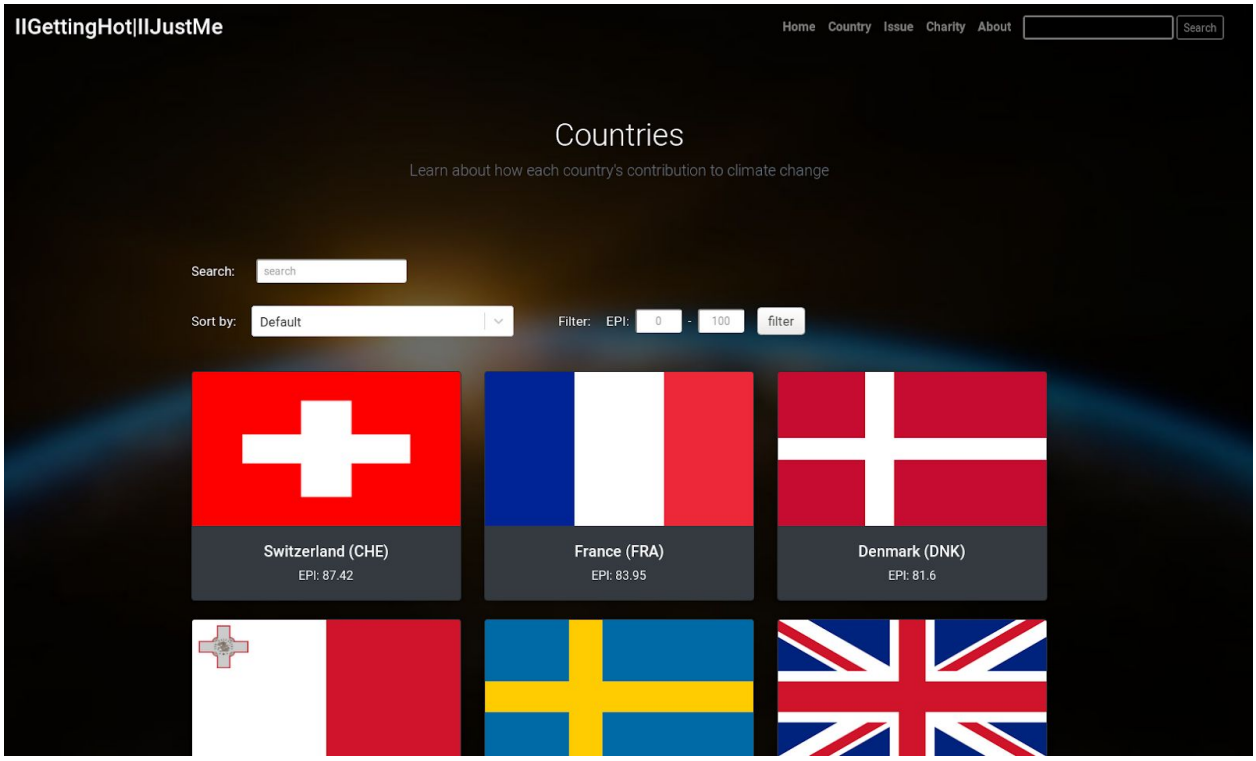
issues model. As mentioned above, there exists a field containing a list of countries in which the country operates in, as a link to the countries model.

About Page

The About Page shows an overview of the project, as well as the members of the team and some metrics of their involvement. Every member has a role, as well as metrics for commits, issues resolved, and tests created. There is also a tracker on repository-level metrics, as well as links and explanations to how the data recollection worked. The tools used for the project and the links to external project assets such as the API and repository are also located here.

Tables

The table pages are constructed using the same abstracted class: Table.js. It handles common functionality of the table pages, including API interface, appearance, and navigation. The derived table classes, each for countries, issues, and charities, only need to specify the corresponding URL for their instances and implement data parsing. The pagination buttons are handled by “react-js-pagination” which in turn triggers the table class to reflect the active page index if a user clicks a pagination link. Table pages also handle sorting, filtering, and searching. Fundamentally, they parse user’s input and query backend server in order to avoid computation on user side.



DB

A Postgresql database is used on the back-end. It comprises of 3 tables corresponding to the 3 models, namely Countries, Issues and Charities. The tables have as their primary key of type VARCHAR, the country_code, the issue_name and the charity_name respectively. Furthermore, each table has a further details attribute of JSONB type that in turn has as its fields, the respective attributes of each model as listed above.

The relationship between the models are enforced as follows:
Both the Issues table and the Charities table has as one of its fields within Details, an attribute of countries, that is comprised of a list of country codes, as foreign keys that reference the country table.

The country table in turn contains a list of issues and charity_names that act as foreign keys that references the Issues (issue_name) and Charities(charity_name) table respectively

Finally, the Issues table contains a list as an attribute of details - charity_names - that references the Charity's table primary key. The Charity's table similarly contains a List attribute - issues - that references the issue_name in the Issue table.

The SQL calls we have:

1. select * from country order by cast(country.details ->> 'EPI' as float) desc, country.country_code asc limit 1 offset <N>;
   Get the country with the Nth highest EPI.
2. select * from country where country.country_code = <country code>;
   Get the country with the country code.
3. select * from issues order by JSONB_ARRAY_LENGTH(issues.details -> 'countries') desc, issues.issue_name asc limit 1 offset <N>;
   Get the Nth issue ordered by the number of countries the issue is seen at.
4. select * from issues where issues.issue_name = <issue name>;
   Get the issue with issue name.
5. select * from charities order by JSONB_ARRAY_LENGTH(charities.details -> 'country') desc, issues.charity_name asc limit 1 offset <N>;
   Get the Nth charity ordered by the number of countries the charity works on.
6. select * from charities where charities.charity_name = <charity name>;
   Get the charity with charity name.
7. select count(*) from <country, issues, charities>;
   Get the number of instances we have for the corresponding model.

Testing

Unittest

Backend is tested using the Python unittest library. The unittests are done automatically using GitLab CI pipeline on each push to Gitlab. To run the tests cd into iighoiijm-be folder and call python testing/test.backend.py. Since our backend code is small (<150 lines without the database class) there are not many tests needed.

Postman

Postman is used to test the API. We currently have 33 test cases checking each endpoints' status code, return type, format, whether it is a valid JSON, return count and contents of each JSON object. It does these tests both by querying by order and also querying by instance names.

Selenium

We apply Selenium written in Python to test GUI interaction from end-to-end. We picked Firefox binary as the primary web driver. "iighoiijm-fe/test/acceptance/guitests.py" contains 10 tests which tests link navigations in overall.

Mocha

We've set up Mocha unit testing in the frontend repository. We managed to get the tests up and running, however we did not figure out how to run custom functions relying on calls to the API. We have a few test cases set up, but plan to make our list of test cases more robust in the future.

Filter

Frontend:

Filtering is handled on table pages simply through React's Form and Input. The filter values are stored in corresponding state variable, which is used to compose API query when the form is submitted or the table page instance list is resetted. There are two types of filtering: between and contain filters. We apply the former one to give an access to query EPI in a number range. This allows users to inspect their EPI of interest easily. The latter serves a simpler purpose by allowing filtering in the country column of both issue and charity table pages.

Backend:

A filter request is passed to the backend in the following format: <attribute>=<value> where attribute is the name of the attribute we want to filter on(i.e. EPI, longitude, model name) and value is usually a list. For most of the attributes we use the value as an interval and return all instances which have those attributes in between. But for some i.e(country for charity and issue) we just match all attributes containing those values.
Filtering is done on the backend side and the logic is implemented in sql_builder.py. It is the first subquery we run since filtering is cheaper than both sorting and searching.
Multiple filters act as AND so only the instances that satisfy all queries are returned.
List of filters for each model:

1. Country: country_name, EPI, longitude, latitude all get a list of two elements and returns the instances in the interval defined by those elements.
2. Issue: issue_name, country. issue_name gets a list of two strings and returns all issues with names between those two strings. country gets a list of country codes and returns all issues that are seen in any of these countries.
3. Charity: charity_name, country. charity_name gets a list of two strings and returns all charities with names between those two strings. country gets a list of country codes and returns all charities that operate in any of these countries.

Search

Frontend:

A search entry is passed from the navbar component to the search component as a prop. The search entry is then parsed for keywords and requested to the backend. The initial series of requests return the metadata necessary to then request the data of the instances that matched the search entry. The instances are then rendered to display.

Backend:

A search request is passed to the backend in the following format: keywords=[<keyword 1>, <keyword 2>,...] where each keyword is a string.
Search is done on the backend side and the logic is implemented in sql_builder.py. It is the second subquery we run since searching is cheaper than sorting.
The search is done as follows:

1. Convert each instance to a JSON string.
2. Use ILIKE %<keyword>%(*<keyword>* regex in python) to find all matches of keyword in that JSON string.
3. Similarly run the same regex in the primary key.
4. Take the OR of these to operations.

5. Take the AND of all results for all keywords.

Sort

Frontend:

Given that we restrict sorting to only one attribute at a time, the layout design decision follows the dropdown menu. We implement a dropdown menu and handling once and let the children components (country, issue, charity tables) decide their own options. Moreover, we also support sorting in ascending and descending order through selecting the same sorting value the second time. The default ordering is determined manually (e.g. ascending by default for names, and descending by default for EPI and number of associated instances). Since the endpoint provided by backend team solidifies the interface among three different model, it is convenient to reuse the same mechanism to sort on different table pages.

Backend:

A sort request is passed to the backend in the following format: sort=<attribute> and order=<asc|desc> where attribute is the attribute we want to filter on and order determines whether the results will be ascending or descending. Both parameters are optional and you can specify only a single one of them if needed.
Sorting is done last as its the most complex task. It is implemented in sql_builder.py and there are always two sort parameters, the second one being the primary key, in order to break ties and eliminate duplicate results caused by ORDER BY and LIMIT interaction.
List of sortable attributes for each model:

1. Country: country_code, country_name, EPI
2. Issue: issue_name, country(sorts by country count), charity(sorts by charity count)
3. Charity: charity_name, country(sorts by country count), issue(sorts by issue count)

Tools

Python wikipedia module

Used for scraping the data from Wikipedia.

Hdx-python-country

Used to do fuzzy name matching of country names.

Python ISO3166 module

Used to get the list of all country names in the world and getting ISO3166 codes for the countries when scraping data about countries.

Postman

Used to design the API and publish it here. It allows adding example GET requests and adding testing the API.

React JS

Used to implement the frontend of the entire website.

Create React App

Allows for the creation of React apps with no build configuration and simple bootstrapping of a project.

Bootstrap

Design website layout from HTML/CSS-based.

NameCheap

Used to register our site's DNS.

Flask

Used for hosting the Backend.

Docker

Used for containerizing and deploying the Backend. Also used to have a testing environment on GitlabCI.

PostgreSQL

Used as the database of the Backend. Hosted on AWS.

Psycogp2

Used in order to access the Postgre database instance from Python.

Nginx

Used as a reverse proxy to route calls coming to the frontend and the backend.

Hosting

Currently, our website is hosted on an AWS EC2 instance. We use AWS Route53 to route our Namecheap domain to this instance and NGINX to do the internal routing from port 80 to frontend and backend local hosting port. Our RDS database is hosted on this instance too. We use docker in order to deploy the backend container and npm's serve to serve the frontend application.