Energenius: Technical Report

Table of Contents

Hosting

Energenius.me

The domain name “energenius.me” was registered on February 26th, 2019 through namecheap.com. The website is using the Google Cloud Platform service. More specifically, the PaaS, app-engine is our main service on GCP. After we created the project, we customized the domain name to the “energenius.me” by adding the record file in the DNS setting. The SSL certificate is managed by Google when we finish customizing the domain name. Lastly, we install the Google Cloud SDK to handle our project locally and deployment. After we initialize the configuration of the project, the only thing is to develop the front-end web page and deploy.

Motivation

The energy decision for each resident is critical to the environment, economics and even security for the whole society. Nowadays with relatively abundant energy resource, the public rarely realizes the impact of excessive use of specific energy might have on the future.

Our website aims to provide clean data to help each person to make a better decision on their energy consumptions. We will quantify the environmental impact of most new and traditional energies, and advise the public on using and saving energy for different purposes. With the educational mission, the website will provide insights into the development of future energy to inspire more civic engagement and achieve energy reform.

The future quality of life is dependent on today's energy use. Improving public awareness of energy reform is essential for building a better future. We hope to increase people's concepts of power by developing this web application to impact our world in a good way.

Restful API Description

Here is the link to our Postman Website:
https://documenter.getpostman.com/view/6823871/S11NMwp4

Energy Category

The Energy Category model introduces the different types of energy that are heavily utilized or under development during human history, for example, electricity, nuclear fusion, etc. The energy forms are divided into two major categories: renewable or nonrenewable. For each type of energy, the analysis was performed on current major utilization mean and the consumption ranking compared to other energy types. Map or picture was used to display the geo-distribution of the energy within the US. Each energy is related to the country that has the largest production and the major usage of that type of energy.
The attributes of each energy are:

- Energy name
- Type (Physical/ Renewable/ Non-renewable)
- Major Production/ Usage Methods
- Energy Consumption Ranking in the US
- Electricity Generation Ranking
- Top Energy Generation Country

- Description of Energy Utilization History
- Images of Energy
- Educational Video scraped from the Youtube

API Options:
GET All Energy Info:
curl --location --request GET "https://www.energenius.me/api/energy?name=all" \
This call accepts "all" and returns all the Energy information in the database.
GET List Energies:
curl --location --request GET "https://www.energenius.me/api/energy?name=list" \
This call accepts the "list" as a parameter and returns the list of all energies' names from our database.
GET Get Energy's Info by Name:
curl --location --request GET
"https://www.energenius.me/api/energy?name=Solar%2520Energy" \
This call accepts the energy's name and returns the energy information we collected in the database.
GET Filter Energy:
curl --location --request GET
"https://www.energenius.me/api/filter/energy?Type=Physical%20Energy&Major_Use=Industrial&Top_Producing_Country=China"
This API call filters the database using three attributes: type, major usage, and the corresponding top producing country.

Production and Usage

The Production and Usage model introduces the general way of industrial energy production and social usage of different energy. Each method of production or usage is related to a type of energy and has a type attribute of either Production or Usage. Each activity has a certain value of carbon dioxide emission and a year of the invention.
The attributes of each production and usage are:

- Production/ Usage Name
- Type (Production/ Usage)
- Related Energy
- CO2 Emission
- Year of Invention
- Usage Field Involved
- Description of the Energy Activity
- Images of Production
- Educational Video scraped from the Youtube

API Options:
GET All Usage and Production Info:

curl --location --request GET "https://www.energenius.me/api/production?name=all" \
Retrieve information regarding the area of application of certain energy.

GET List Usage & Production:

curl --location --request GET "https://www.energenius.me/api/production?name=list" \
This call accepts the "list" as a parameter and the list of all productions' names from our database.

GET Get Production & Usage Info by Name:

curl --location --request GET "https://www.energenius.me/api/production?name=Medical" \
This call accepts an activity's name and returns the activity information we collected in the database.

GET Filter Energy:

curl --location --request GET "https://www.energenius.me/api/filter/energy?Type=Physical%20Energy&Major_Use=Industrial&Top_Producing_Country=China"
This call filters the database by three attributes: type, major use, and the top producing country.

Country

The Country model introduces the essential information about the energy of countries in the world. Towards each country, the consumption and production of electricity were analyzed. The total electricity production and consumption for each country are listed. The issue of energy shortage of some countries is shown by the number of electricity outage days. The country is related to an individual energy type by its top producing energy type.
The attributes of each country are:

- Country Name
- Total Electricity Production Amount
- Total Electricity Consumption Amount
- Electricity Outage days per year
- Region
- Population
- Description of Country's Energy Situation
- Images of Country's Flag
- Educational Video scraped from the Youtube

API Options:

Get All Countries' Info:

curl --location --request GET "https://www.energenius.me/api/country?name=all" \
This call accepts the name of a country and returns the information summary of energy for that country.

List Countries:

curl --location --request GET "https://www.energenius.me/api/country?name=list" \
This call accepts the "list" as a parameter and returns the list of all countries' names from our database.

Get Country's Info By Name:
curl --location --request GET https://www.energenius.me/api/country?name=United%20States" \
This call accepts the country's name and returns the country information we collected in the database.
Get Filter Country:
curl --location --request GET
"https://www.energenius.me/api/filter/country?Region=all&Population=all&Total_Production=%3E2000"
This call filters the database by three attributes of the country instances: region, population, and the total electricity production of that country.

Data Source

At the current stage, we mostly used the Wikipedia and google image API to get our source data. However, we project to implement more diverse data extraction in the future. Through our understanding of the energy models and research on available data sources, we identified the following API that we can harvest on:

- For energy consumption data, utilizing the data from the U.S. Energy Information Administration, The World Bank website.
- From the National Renewable Energy Lab(NREL), we can obtain greenhouse gas emission data for different cities and countries.
- From OpenEI, we could harvest the general information and introduction regarding each energy type.
- From Youtube, we scrape the educational videos regarding each energy and industrial utilization.

Here is the link of the example of the API and sample data we have used for the current website.
https://docs.google.com/spreadsheets/d/1gwxoz9N8zbH_LdgsMPt7g1Dpue2qR1MXgvCGW26ccb4/edit?usp=sharing

Testing

We added the unit testing backend within the backend folder. It is testing the input page portal valid and API invalid page not found.
We added Postman testing through Postman and then used Newman to generate the JSON file to automate the testing process. It is in our root directory.
Frontend testing includes testing components of every page. This ensures individual components appears correctly. Additional tests about instance pages are also added. They check whether data related to a certain instance is retrieved correctly. It is the App.test.js file located in the client folder.

Tools

ReactJS

ReactJS is the frontend framework we utilized to build up the dynamic website. By using ReactJS, we were able to construct the website to increase the efficiency of building new pages and change the existing frames. We also utilized the ReactJS to extract the data from restful API directly on the frontend.

Material-UI

The Material User interface is an open source react library that contains useful components for our website. Combined with React-Bootstrap, we were able to stylize our website, add pagination, grid, and card features.

Google Platform App Engine

Google App Engine is the main service we used for the website's backend environment. The PaaS and app-engine provide the standard OS environment including many libraries. As long as we declared the library name, app.yaml, and package.json, it will fetch those libraries and packages for our front-end and back-end buildup.

Postman

Postman was used to displaying our API function and testing the API calls. We added the description regarding each API call through Postman.

Newman

Newman is the software to automate the unit testing process in Postman. Newman will either accept a URL as Postman's link or the local JSON file for testing. It also provides the testing result statistics in the end.

React-Bootstrap

We use React-Bootstrap as the CSS framework for our React application. It's crucial for a responsive and mobile-first appearance while keeping the website light-weight.

Selenium

The main tool for GUI test, provide acceptance test on website features, use browser chrome driver.

Mocha

Provide unit test, help to test Javascript function in details.

Gitlab CI

Gitlab CI was deployed for autonomous testing for each Gitlab commit.

D3

D3 is a visualization tool we used in the React environment to visualize our data to the users.

Database Description

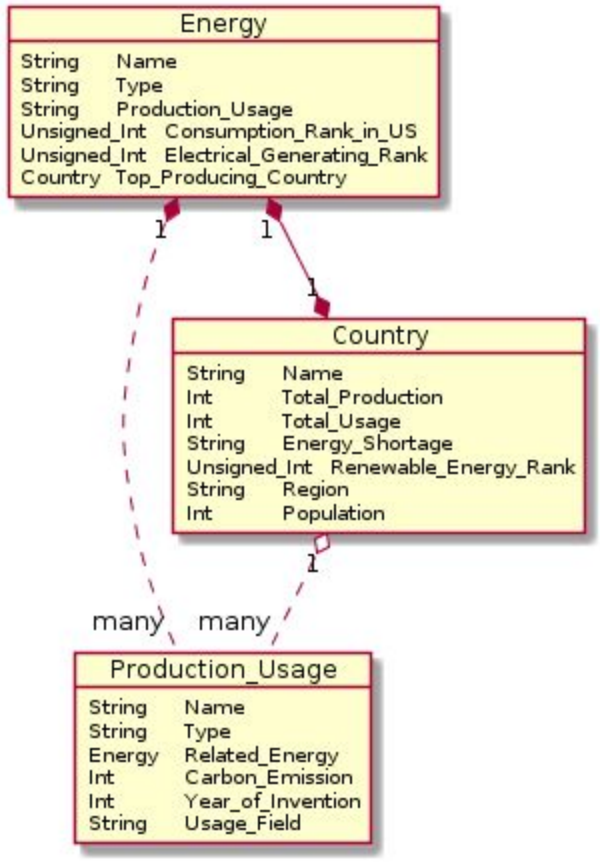

Figure1. The UML Diagram

We use the NDB (NoSQL database) stored in the Google Cloud Platform data store. To design the database structure, we first identified each attribute for three major models. Then we spent a lot of time to search for the target data and then finalized the attributes for each model. We also included the attribute to buildup the data relation between each model. For example, for each

energy, we added the top producing country. Then we built up and tested the database framework using python and python unit test.

Pagination

We implemented pagination with the ReactJs component react-js-pagination. When the model page is loaded, it dynamically requests instance data from our database through our API. This array is stored as a variable in the model class. Using the active page number, the starting index of the first instance on this page is calculated. The page is then rendered indexing into the array stored. When a page button is clicked, handlePageChange function updates the starting index accordingly and renders the page again.

Sort

The sort is implemented using the sort function in JavaScript. I explicitly declared a compare function GetSortOrder(prop) where the Prop is the property to compare the two elements with. When the user selects a property to sort, handleSort function sort the instances array with the property selected using GetSortOrder to compare every pair of instances. The page is then reloaded with sorted instances shown in order.

Filter

The filter is implemented by calling our API. When the user clicks the “Apply Filter” button, the “handle filter” function is triggered. In the function, types the user has selected to filter by is retrieved and parsed. New instances are fetched from our API with these types specified. The page is then reloaded with new instances.

Search

For global search and local search are both handled in the front end. For global search, we added the search bar in the navigation so it appears on every page. After the user gives an input, we send this input as an ID to another page called search. Then we scrap all the data of all instances of all models from our database. And search for occurrences of this search id. Then we displayed the results with highlighted search key in Google search fashion.
For model search, we implemented the same way. Except now we don’t scrap all the data because that will be a global search.

Issues from the Previous Phase

One of the biggest issues we had from phase 2 was API request calls. For the front end, we directly fetched our media data from outside sources. In this phase, we migrated all the data into our own databases. In addition, we added related model instances in each instance page so all instances across different models are linked.
The UML Diagram was updated to contain the attribute's type.

User Stories

To our Developer

Our group is the customer of the website: perfectfitfor.me. After careful reviewing of their purpose and social engagement factors, we created sixteen user stories as follows:

1. The Job Model Description
In each job page, there is only a few information such as location, update time and job type. As a customer, I hope to see some more information to describe this job. Also, if you could make one more instance, the company will make more sense. So how many companies in this city. The transportation between companies or something like that.

2. Mobile Version has not been compatible yet
Images are presented in a strange size. The menu and dropdown list on the mobile failed to load in the correct format. As a mobile user, I would like to have the mobile version of the website working so that I can access the data more flexible.

3. Ununified Block Sizes in About Page
The block such as "about our site," "about our data," and the toolbox block are in different sizes. The uniform block size on the same page will be visually pleasing to the users. As a user, I would like to see a simple and straightforward website so that I can obtain as much information from the page.

4. The transportation page formatting
The transportation page should have a better image format. As a customer, I want to make it in excellent order and organized so that I can proudly recommend this website to other groups.

5. Dropdown button leads to error
As a user, I would like to see more options on your home page. Maybe implement this dropdown feature to each of the three model buttons. And the dropdown could display some attributes?

6. Potential Multimedia Presentation
As a user, I would like more multimedia presentations on your website, so that I am more likely to be attracted by the city. For example, a short intro video about the city and famous tourists sites' photos will help a lot.

7. The description of City's Stats
As a user, I would like to know the description of how you determine the rate of City Stats so that I can have more insights about how to consider these statistics on my own. For example, explain what does 10% of Education and 82% of Housing mean for San Jose.

8. More Connectivity in A Page
As a user, I would like more connectivity within a webpage, so that I can browse the website more flexible. For example, in each city's page, you may add links to its transportation and job review.

9. A Description on Transportation Ratings
As a user, I would like to know how you rated the transportation quality between different cities so that I could feel more confident in relying on these statistics. A simple formula or reference may help users to understand.

10. More Content in the Job Model
As a job finder, I would like to see the business development trend in a city, so that I can determine if the city suits my career ambition in the long run. The opening jobs in a town may not help me to achieve that. A business development analysis, for example, the GDP percentage of tech companies in the city, will give readers more insight.

11. Adding Filtering Function
As a user, I would like to perform filtering function on Jobs, Cities, and Events models, so that I can view interesting data from each page. For example, as a job seeker, I want to filter the jobs that only meet a certain level of Potential Income, so that I can design my career path based on the result.

12. Adding Searching Function
As a user, I would like to perform the searching function on Jobs, Cities, and Events models, so that I can obtain the specific data from the database faster. For example, I want to search for the music festival from the Events model without going through every page and find by myself.

13. Adding Sorting Function
As a user, I would like to perform sorting function on Jos, Cities, and Events models, so that I can get the intuition of the order of the target data. For example, I want to sort all the cities based on the rating so that I can compare different top rated cities.

14. Clean Event Model's Data to Eliminate Repetition
As an event finder, I would like to get different activities available without redundancy so that I don't need to view the same event over and over again. For example, the "Life Coaching" activity appears 15 times from page 19 to page 21. Also, each event grid contains an empty circle, which I assume is used for the rating. If not, please eliminate that circle from each grid.

15. Improve Pagination by Adding Previous Page Numbers
As a user, I would like to view the previous page numbers as well as the following page numbers so that I can more efficiently navigate the pages. Right now, the pagination bar on the bottom doesn't show the previous page numbers. For example, if I would like to jump from page 4 to page 2 directly, I need to click twice on the backward arrow. It would be nice if I can click page 2 directly.

16. Adding at least Five Attributes to Job Model
As a job seeker, I would like to view at least five attributes related to a job in the Job model pages so that I can get more information more quickly. Right now, you have the job title, potential income, and the location as three attributes. More attributes such as the direct link to apply and related pictures could be good attributes that you can add in each job's grid.

17. Differentiate Two Search Bars
As a user, I would like to know the difference between the two search functions on each page so that I can search for information within my target range. The suggestion is to visually differentiate between the two searches or combine two searches into one in case of any confusion to users.

18. Arrange Sorting Filtering on each Page
As a user, I would like to see the sorting, filtering and searching areas to be aligned and allocated on the side of a page. The suggestion is based on the visualization. For example, the Amazon shopping page.

19. Multimedia on Job Page
As a job finder, I would like to see more multi-media elements in the job's page so that it's visually appealing. For example, pictures, video, or embedded google map.

20. Dynamic Links on Home Page
As a user, I would like to navigate myself through models not only through the navigation bar but also from your home page. The suggestion is to also add dynamic links on your home page's descriptions or pictures of different models.

21. Create a Sliding Window on Home Page
As a user, I would like to see the sliding window of images on your home page so that the site is more visually appealing. The suggestion is to add photos of different cities and create a sliding window on the home page.

From our Customer

Our website received several suggestions from group 15. We have solved all the issues at this time. The suggested user stories and our comments are as follows:

1. Search button doesn't work yet

Developer Comment: We will implement the search function as soon as we are transparent with our data managing. We estimate the search will take about two days to be implemented.

Update:

Now we have migrated our website to ReactJs, the actual implementation will require additional work, and we will get it implemented in the next phase.

2. Some pages on mobile require side scrolling to view

For example, the residential page has a pie chart that can be more easily viewed if compressed to proper screen size.

Developer Comment: We will probably replace the static pie chart with a dynamic pie chart that will update according to the fetched data. We also haven't adjusted our website for the mobile browser; as our development progress, the website will be more refined and compatible.

Update:

We use to React and Bootstrap to accomplish the Mobile Version adaptation.

3. Images in the Auto-scrolling needs improvement

Since the images in the home page are only for aesthetics, there is no particular need for a user to click through. Auto-scrolling the images would look nice. Also, the front page is too monotonous to attract visitors.

Developer Comment: I believe our website is using auto-scrolling. The issue is the auto-scrolling process is taking too long. We will decrement the time to auto-scroll the images in our front-end. We might add additional features on the auto-scrolling slides.

Update:

Once our backend database is up, we will import a summary of each category on the front page. Also, the auto-scrolling feature might change totally, we will keep this issue in mind.

4. Tab name "Production and Usage" is switched in sublinks: "Usage and Production"

For example, https://www.energenius.me/country/china.html

The top bar says "Production and Usage", but the sidebar says "Usage and Production," though they are meant (I assume) to refer to the same category.

Developer Comment: The inconsistency in our category bar is annoying. We have already fixed the difference along with some typos and grammar issues. Thank you for the advice.

Update:

We have fixed this issue in our ReactJs framework.

5. Home button leads to a 404
Developer Comment: We have noticed this issue and have already fixed this bug from our backend side. Thank you for letting us know.
Update:
It now leads to the home page correctly.

6. Getting a 404 Not Found error when clicking on Energy Category
Developer Comment: We are fixing the backend database as well as the frontend javascript, it should be fixed by now.

7. Buttons on Navbar not working
Developer Comment: We were adapting to the ReactJs earlier. Because the routing to subpages under each model requires the support of Javascript, we had to temporarily disable it. It works now and we look for more update on the Navigation Bar in the future.

8. Little information about site provided on the Home page
Developer Comment: This issue will be our priority on the next phase, the frontend members are cooperating with the backend team for displaying instances. We will add additional features like interactive element to present an interesting summary of the website.

9. Learn more button results in 404 error
Developer Comment: We have fixed this issue. The API of Gitlab only return back the commit by page each time. We are still trying to fix this bug. Our commit and unit tests number is slightly off

10. Parts of the Instance Pages aren't implemented
Developer Comment: We are trying to use several API from different websites at the same time. Some instances pages implementation are still in progress. We also realize that both appearance and content are poorly designed and bored. These issues will be our main focus on the next phase.

11. Search option is not working properly
Developer Comment: Estimated time of completion was 4 hours. The actual time of completion was 10 hours. The search function was mainly implemented in the front-end. The searching range includes every attribute of each instance. The searching target string will be highlighted in the result.

12. Filter Energy is not working
Developer Comment: Estimated time of completion was 2 hours. The actual time of completion was 4 hours. Filter request is constructed in the front-end and sent back to the backend. The backend will filter data from the database and send to the front end to display. The major difficulties were request construction, request parsing, database filtering, and dynamic data display.

13. Navigation bar not showing properly on mobile
Developer Comment: Estimated time of completion was 1 hour. The actual time of completion was 30 minutes. Added additional tags for each label in the Navigation bar and it can now be auto-scaled to fit different devices.

14. Countries Make the image be the country's flag for all of them - currently, most are the map, some are the flag
Developer Comment: The inconsistency was fixed. Right now all of the country's grid will display that country's flag.

15. Search is case sensitive
Developer Comment: Thanks for letting us know, we have fixed this issue. Now input can be in either uppercase or lowercase, and the highlighted content remains consistent to the original content. We have also noticed that our search bar can only search words but not sentence, we will fix it in the next stage.

16. Reverse sorting
Developer Comment: Estimated time of completion was not enough to finish it. We will keep updating until the presentation.

17. Can't re-use search bar after searching once; have to go back to the previous page to search again
Developer Comment: Front end people are struggling for the D3 visualization in this phase of the project. They took two hours to try to fix this problem but not work. They will keep tracking on this issue until the problem.

Self Critique

What did we do well?

IDB4

For phase IV, we added pretty visualization, refactor the program, and fixed warnings or errors from the previous phases. We found that the most challenging part is to create dynamic visualization utilizing the data we have stored in our database. Since we have taken the data management course, we have a basic idea and experience of presenting data using tools like Tableau and R. However, it was the first time for us to use the D3 library within the react environment. We did well on the object-oriented programming and data fetching for the visualization. The experience of using D3 is similar to the previous phases' page

implementation. We all have gained valuable knowledge in applying the D3 library to React and improved on the perception of data presentation. The refactor part not only made our code straightforward and easier to comprehend, but it also strengthened our understanding of Javascript's object-oriented programming.

IDB3

For phase III, we achieved the filtering, searching, sorting function in the React environment and fixed the API issues we ignored in phase II. The searching function was implemented mostly in the front-end. We successfully configured the "global" and "local" search in which it will target the different range of data. Filtering and sorting were achieved by the backend before sending any data to the frontend. We implemented in this way mostly because of the data is close to the database and the Python library offers efficient tools to sort and filter data. During phase II, we implemented the frontend to fetch the API from the outside websites. However, we should store the data in our backend so that the front-end could fetch data from the backend directly and more efficiently. We fixed this issue by modifying our database and front-end fetching functions.

IDB2

Phase II was the heavy lift part of the whole website development. We successfully transformed our static HTML website to the dynamic React based website. The transformation was extremely hard at the beginning since we all did not have any experience in website development using Javascript. We set up the backend server to provide the API service to the front-end and other developers. In our case, we used the Google App Engine instead of the Google Cloud Platform. The Google App Engine and PaaS provide the standard OS environment including many libraries. We successfully deployed the React based website in our Google App Engine server, which is not commonly seen for the recent years' website development. We refined the API using Postman and wrote unit tests from different platforms.

IDB1

For phase I, we came out with two website ideas, constructed a static website in HTML and built up our GitLab repo for future development. We set up the Postman to document our expectation of the API. The technical report was finished using Google Doc. Our group decided to use the Google App Engine for our backend server. We have a lot of discussions over the website proposal and backend server selection. We explored the restful APIs from different sources and settled down several useful ones. Part of the About page could fetch data from GitLab dynamically.

What did we learn?

IDB4

For phase IV, we learned the D3 as a new tool for data presentation, refactoring as an important component for future software development, public speaking skills for each one of us, and documentation techniques. Through implementing data visualization using the D3 library, we learned a lot about constructing new models based on the existing software program. D3 implementation was similar to the object-oriented programming style from the previous website construction in javascript. Through recent lectures, we gained knowledge about the importance and technical details of refactoring code. We experienced different refactoring strategies such as singleton to clean up the code and improve time and space efficiency. For the upcoming presentation, we will use PPT to visualize the website's components and practice speeching to give out a professional presentation.
Another big lesson we learned is to always pull before doing any file operation in case of conflicts. And do not delete the whole directory and be confident that you have pushed. The truth is that you may have not pushed :) Ideally, keep multiple copies by data redundancy is good protection.

IDB3

Our group learned to implement searching, sorting, and filtering and we gained a better understanding of the front and back cooperation. We improved our software quality assurance skills by creating unit-tests in Mocha, Postman, Python unit tests and Selenium. Along with the Postman's unit-testing, we learned the new automation tool called Newman. The website used to have only 9 instances for each page. At this stage, we explored more raw data from the outside API and stored them in our backend database. After phase III, we managed to have more than 81 instances for the website. We strengthened our understanding of object-oriented programming by implementing sorting and filtering from our backend.

IDB2

From phase II, we learned a lot about using React, backend design, deploying website service on Google App Engine, unit-testing, group communication, and software developing documentation. Since we all didn't have experience in Javascript, we learned most of the React through online tutorials. The online examples were helpful but different from our expectation. We started the whole project by understanding the data flow between the front and backend and designing database structures. Then we searched for restful API from many sources. Experimenting the restful APIs using Postman and deciding which API is suitable was a great learning experience. After that, we started to build up the front-end with the Object-Oriented

Programming mindset. For the backend, it was easier to buildup the whole structure using Python. However, we came across serious issues when we were trying to deploy our website to the GAE server. There were many helpful documents to guide us through the React website deployment on GAE. The deployment experience taught us about how to monitor and deploy React service on the GAE server. Postman was a great tool for us to learn. At this stage, we need to present and test the real API functions using Postman. We learned to use Mocha, Selenium and Python unit test library to test various parts. Version control was emphasized throughout the development. GitLab CI was used to achieve the autonomous unit testing for each push.

IDB1

As a group, we learned about efficient group communication, idea proposal, static website development, and using Postman to document API. Most of the members of the group haven't worked with each other before. It is important for us to know each other and learn how to work within the group. Each one of us has different specialties in a different area. To assign each developer working on their skilled area is essential for group efficiency. The group was having challenges deciding the proposal and the backend server to use. During the process, we learned to efficiently discuss our ideas and achieve an agreement after each meeting. We learned to use Postman to present our API and write HTML to prototype our website ideas.

What can we do better?

IDB4

One thing that we could improve on is to generate more complex graphs for our data visualization. For right now, our data visualization impression is limited mostly because of our unfamiliarity with D3 in React. After this experience, we felt more confident to construct more detailed graphs using D3. We also could explore to build up the graph using Tableau and embed it on the website.
Another thing we should improve on is to design the backend data more cleverly beforehand. Our team came across this issue over and over again because of the frontend changed its need for data over time. This issue cost us a lot of time since we need to rebuild the backend data structure. A better approach is to discuss the requirement for the frontend at the beginning, then include a wide range of data as we proceed.

IDB3

A major issue we could fix in the future is the Searching function. Currently, the front-end page consists of two search bars, in which one is for the global search and another for a local page

search. We would want to combine these two search bar and differentiate it more dynamically in our code. Two search bars will be somewhat confusing to the user. Our searching will only query the text content without concerning the attribute headers. The improvement is to has a wider range of searching query. The UML diagram is lacking cardinalities and we need to improve the diagram to display the database's structure more straightforward.

IDB2

There were a lot of things that we need to improve on after Phase II. Especially, we didn't allocate enough time on Phase II during the Spring break. The whole development phase was finished in a rush manner. There were warnings we didn't fix and many front-end structures that we need to refine. The front-end is currently fetching data directly from outside websites. It is not using any service from the backend database. This was a major issue we need to improve on. The Postman API was lacking flexibility because we didn't fully consider the use cases of our API services. We should also improve our response towards customer issues to give an estimated time of completion and negotiate with our customers on some technical challenges.

IDB1

For phase I, we proposed our idea very late. We have a few ideas that were taken by other groups. Hence, we need to work on improving group efficiency and manage time well. We searched for many Restful APIs but haven't tested most of them. The only site we tested was the Wikipedia API. In the future phases, we should specify the tests before any further implementation to ensure the expected performance.

What puzzles us?

IDB4

The thing that puzzles us the most in phase IV was which D3 library to use in the React. Clearly, there are a lot of D3 libraries available but we do not have a clue or suggestion of any one of them. Eventually, we chose just the D3 library to use. The visualization part is hard to start and we watched a lot of tutorials. Another challenge is to refactor the code after constructing most part of it. It will be easier if we have the mindset of object-oriented programming from the beginning. However, it was a great learning experience for us to get in touch with the D3 and refactoring.

IDB3

At first, we were unsure about where should we implement the searching function. The backend could perform the search but it will be hard for highlighting the target text. The front-end

Javascript's search function was unfamiliar to us. However, in order to categorize the searching result based on models and highlight the searching text, we allocated the searching implementation in the front-end. We were also having trouble to decide which part of data we should search on. In the beginning, we want to only query the attribute data which doesn't include the introduction of each attribute. However, the Google search will certainly include the full content of each page. Therefore, we need to combine most of the data together and then perform the query.

IDB2

We were unclear about the structure design and data flow of our website. The group spent two days discussing the database design and API sourcing. After we settled down the APIs and split the work to front-end and back-end developers, the previous design became unsuitable pretty quickly. Then we decided to learn from the other examples. The Postman unit testing was easy to write. However, the searching for the correct tool to use for automated testing took a bit of time. Fetching data in the front-end was hard to achieve. In the beginning, we have the cross-origin restriction issue. We were unsure about where and how should we import the data from. Eventually, the front-end was able to consume the outside API by itself. The ideal model is for the front-end to consume data fetched from the back-end. We will need to implement that for the following phase.

IDB1

The major concern we have was to migrate our code from HTML to React in the later phase. We searched for some suggestions about how to construct our HTML code to eliminate difficulty for the future. However, we couldn't find ideal advise. We also didn't know much about how to design the Postman API at the beginning. After going through several examples, we have some ideas about how to design the API. In the end, we did not have experience in writing the technical report for the website. However, the course page gives more instruction on the essential parts and the overall structure, which was helpful for our documentation.

Visualization

Carbon Emission of Energy Production and Usage

This image shows different ways energy is produced and used. The size of the circle represents the amount of carbon (in Million Metric Tons) being emitted from each process. We can see that offshore drilling emits the most amount of CO2 followed by land rig and vehicles. We would like our users to learn about how different processes are affecting the environment. The graph can be found the Production Visualization under the navigation bar.

Energy Production and Usage of Countries in the World

The image shows the production and usage of each country by rank. The production is represented by a red bar and the usage is represented by a blue bar. The production and usage are generally equal for each country. We chose this visualization because we want to find a relation between production and usage for each country. The viewers could also get an idea about the exports and imports based on this graph.

US Total Energy Consumption Map

The map utilized the API data from the U.S. Energy Information Administration website. The map shows the amount of total energy consumption(in Million Btu) for each state by different coloring, in which the darker color represents more energy consumption. We consider this graph is straightforward in its way of expressing the data and utilizes the multimedia. The graph could be found under our website's navigation bar: State Map.

Group Member

Yige Wang, Pengdi Xia, Shijing Zhong, Wenyuan Wu, Yaoyang Liu
(Contents above has been checked by Grammarly)