Software Design Document

for

Climate Trace
(Central America)

Version 4.0

Prepared by Vitaliy Stepanov
Naoki Lucas
Andre Weertman
Max Ayala

Global Energy Transparency

June 15, 2021

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Max Ayala	12/1/2020	MongoDB turned out to be more cost effective	1.1
Max Ayala	12/1/2020	The overall architecture was too confusing	1.2
Max Ayala	1/26/2021	Complete rehaul of section 4.2 because a new database organization was finalized.	2.1
Max Ayala	2/9/2021	Small changes to domain layer to reflect new forecasting tools	2.2
Vitaliy Stepanov	2/23/2021	User Interface changed from Unity to React. D3 libraries used for data visualization	2.3
Naoki Lucas	2/23/2021	Updated ‘Raspberry Pi’ to ‘Server’ as hardware limitations/requirements come to realization	2.3.1
Vitaliy Stepanov	3/16/2021	Added MERN stack Architecture	3.0
Naoki Lucas	3/16/2021	Updated ‘server’ to ‘manager’; Added flow chart figure 3.2	3.0
Andre Weertman	3/16/2021	Updated data flow diagram (3.3) and description	3.0
Max Ayala	3/16/2021	Updated system overview for new GUI design and change in manager structure	3.0
Andre Weertman	5/27/2021	Product Scope, Application, Persistent layer, and Cloud Heroku paragraphs updated	4.0
Andre Weertman	5/27/2021	Updated Data Flow Diagram, GUI interface design, Use case diagram, Manager Diagram, and software interface diagram.	4.1
Andre Weertman	6/12/2021	Updated Application interface, GUI images and the paragraph to describe them	5.0
Andre Weertman	6/12/2021	Perfected Title, title section layout, and page notation aligned with the table of contents above	5.1
Andre Weertman	6/15/2021	RTM matrix completed	5.2

1. Introduction

1.1 Purpose

The purpose of the software design document (SDD) is to describe the architecture and system design of the website scrapers, forecasting model, and GUI for visualizing that data. It shows diagrams to engineers, designers, and product managers on how all the components should be created and interact with each other.

1.2 Product Scope

This document describes the implementation of this project and its three main components, and a database they all share: First, the web scrapers, their adapters, and the manager system that keeps our database up to date by continuously running them on a timed interval. Second, the forecasting model forecasts our data using FB Profit, which the manager system also controls in timed intervals to maintain currency. Third, our user application for visualizing our data. Which is split into two major components, the backend server which is our secure link between the database and front end client server where the data is visualized and presented for public use, both hosted on Heroku.

1.3 Definitions, Acronyms and Abbreviations

- MongoDM - free non-relational (NoSQL) database
- DB - database
- Document DB - a document based non-relational database (NoSQL)
- GUI - Graphical User Interface
- Use Case Diagram - Shows potential use cases on an actor by actor basis
- Time-Seris Algorithm

1.4 References

"Adapter Pattern - GeeksforGeeks." GeeksforGeeks, 3 May 2016, www.geeksforgeeks.org/adapter-pattern/. Accessed 12 Nov. 2020.

Layered Architecture. "Software Architecture Patterns." O'Reilly Online Learning, 2020, www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html. Accessed 12 Nov. 2020.

2. System Overview

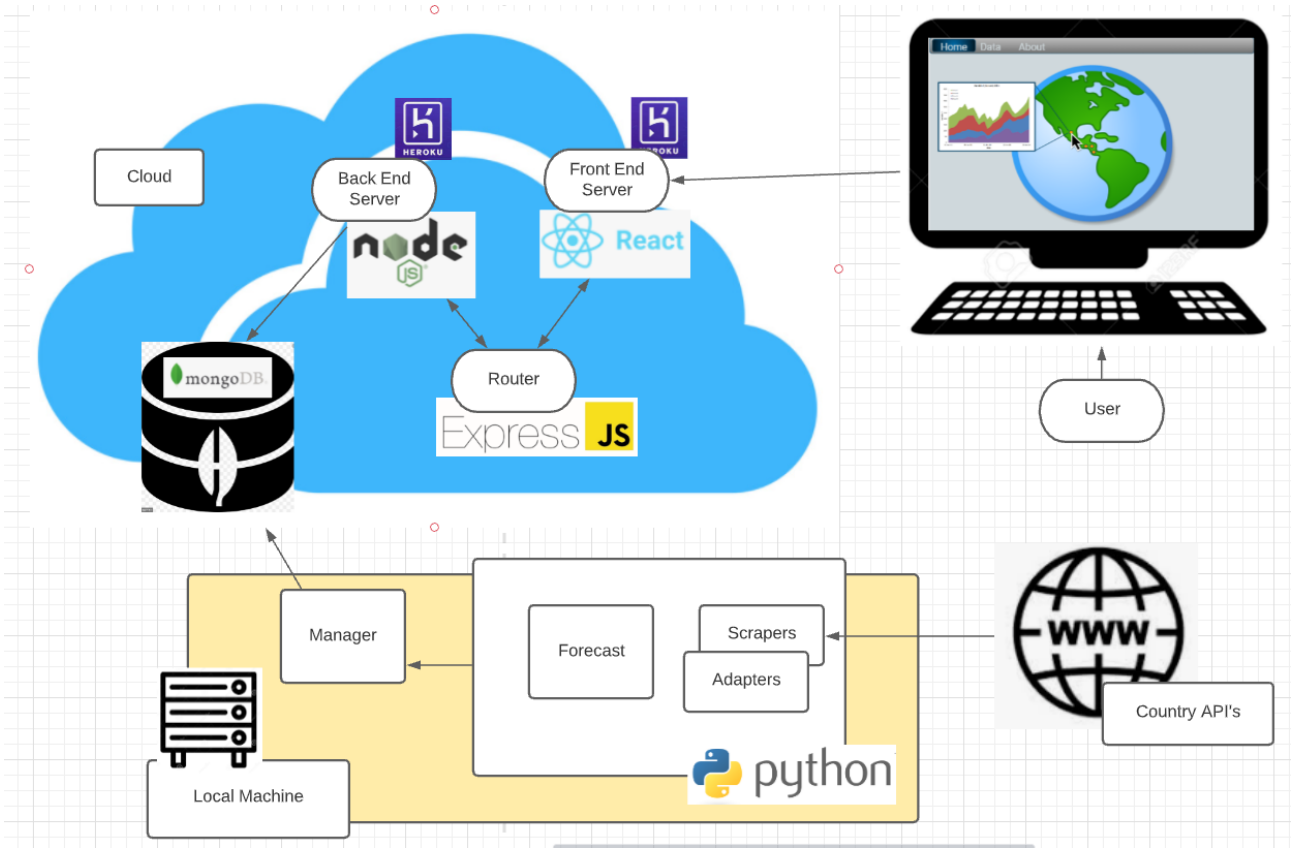


FIGURE 2.1

2.1 The Cloud

External Web Pages:

- Each country present in our model has its own unique webpage from which it reports it's electrical generation data.
- These webpages are all different, they report data in different formats and with varying degrees of availability and accuracy.

MongoDB:

- The data that is scraped from external web pages will eventually return to the cloud as it will be stored in a database using MongoDB
- The data in the database will follow a certain format and will therefore be easy to access and manipulate for our visualization tool.

Heroku:

- Hosts our back end server takes queries from our front end user client and accesses our database with a secure authentication layer no client user is privy to.
- Separately hosts our client front end server, where public users can scroll our globe to the countries we've scraped, and select the time frame of data they want presented for them.

- Routes queries and data back and forth from front to back end server with express

2.2 The Manager

A manager (Windows or Linux machine) will be set up to automate scraping and forecasting of the data. It will be running one main program functioning for both scrapers and forecasters in timed intervals. A basic flowchart can be seen in figure 2.2.

Scrapers:

- The manager will facilitate scraping as well as reformatting and then store the data in the database
- It will implement the scrapers using an adapter pattern as described in section 3.2.2

Forecast:

- This process will use an appropriate forecasting model for each energy type, including additive regression model (fbprophet) and random regression trees.
- It will then stored the predicted data back into the database

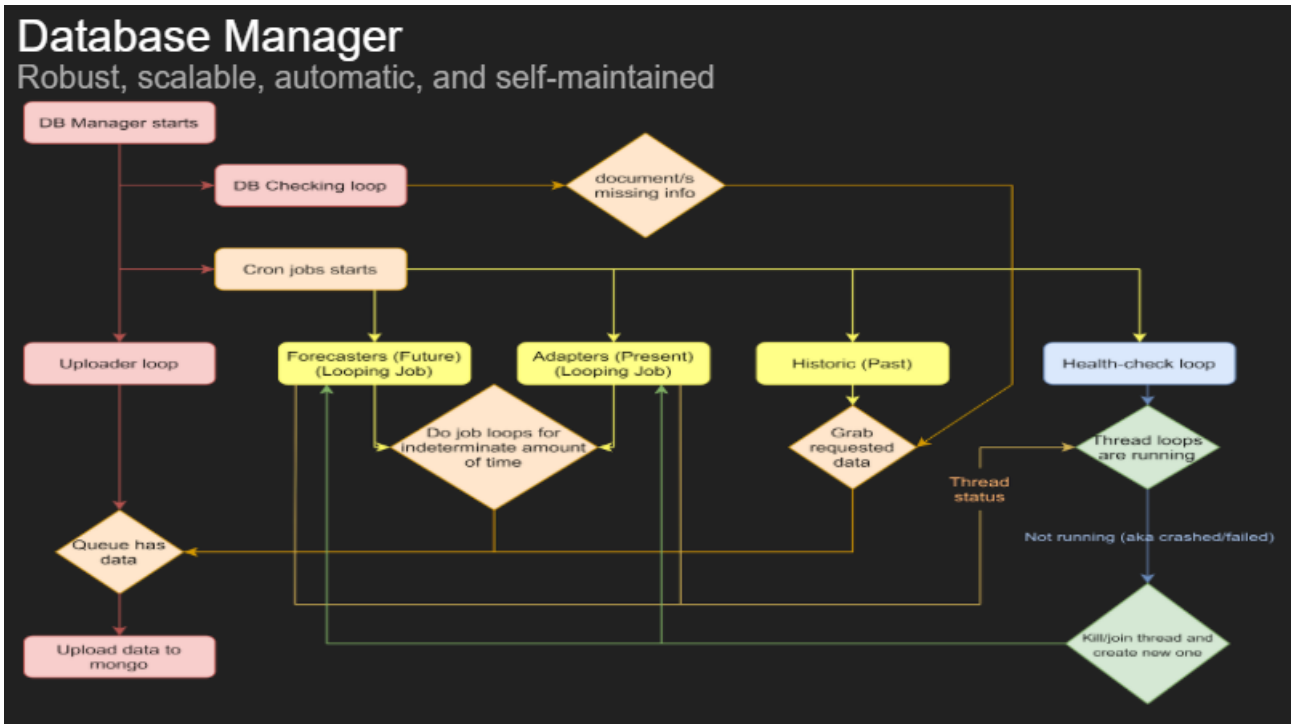
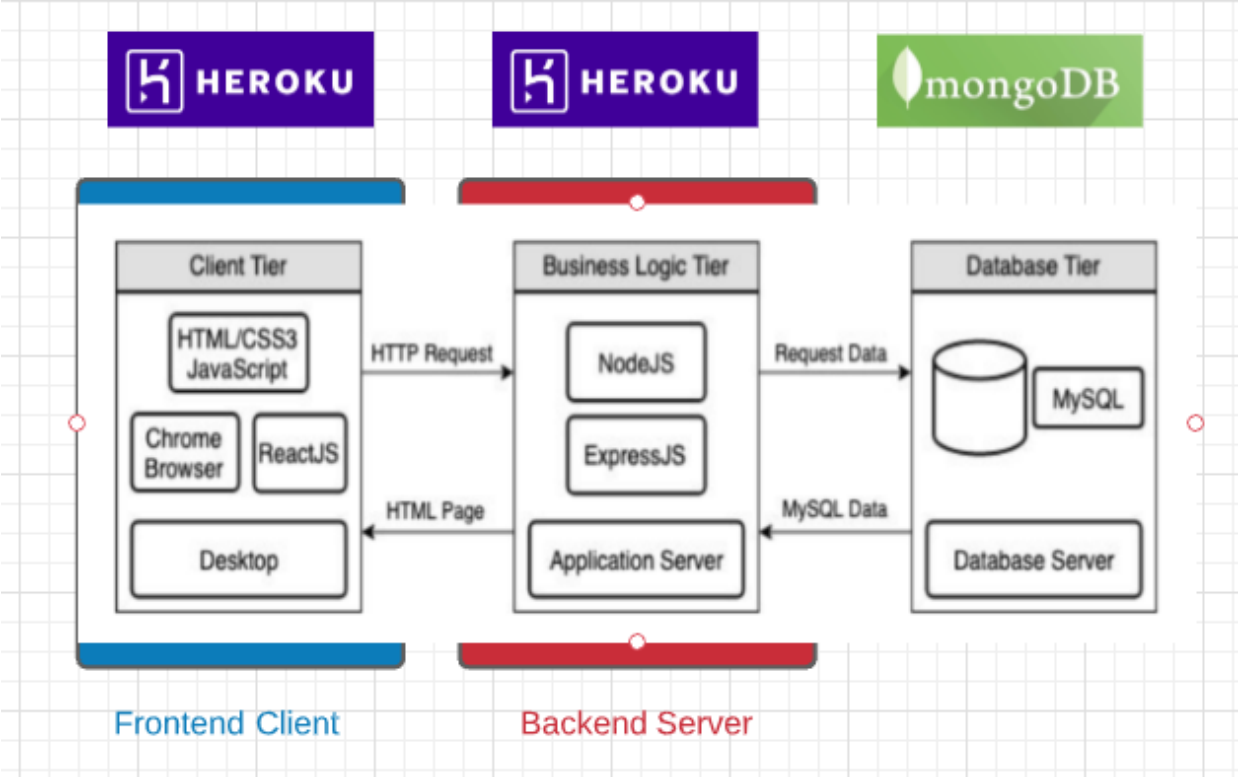


FIGURE 2.2

Basic flowchart for the manager used to help explain the layout and incorporation.

2.3 Application



The GUI application for our users will incorporate a MERN stack architecture pattern, using MongoDB, Express, React, and Node.js. This architecture splits the server into a secure backend server, and a frontend client server. This will allow for users to navigate our globe and query specific time ranges of countries they want visualized graphically, which is routed to our backend through express. Backend handles the data extraction from our real time manager maintained Mongo database, and again using express routes it to the front end. Finally using client-side rendering, our frontend processes the data and graphs plotly for our users to visualize. Both front and backend servers are independent applications from each other therefore require separate hosting, both of which are run as individual applications on Heroku.

3. System Components

3.1 Decomposition and Dependency Description

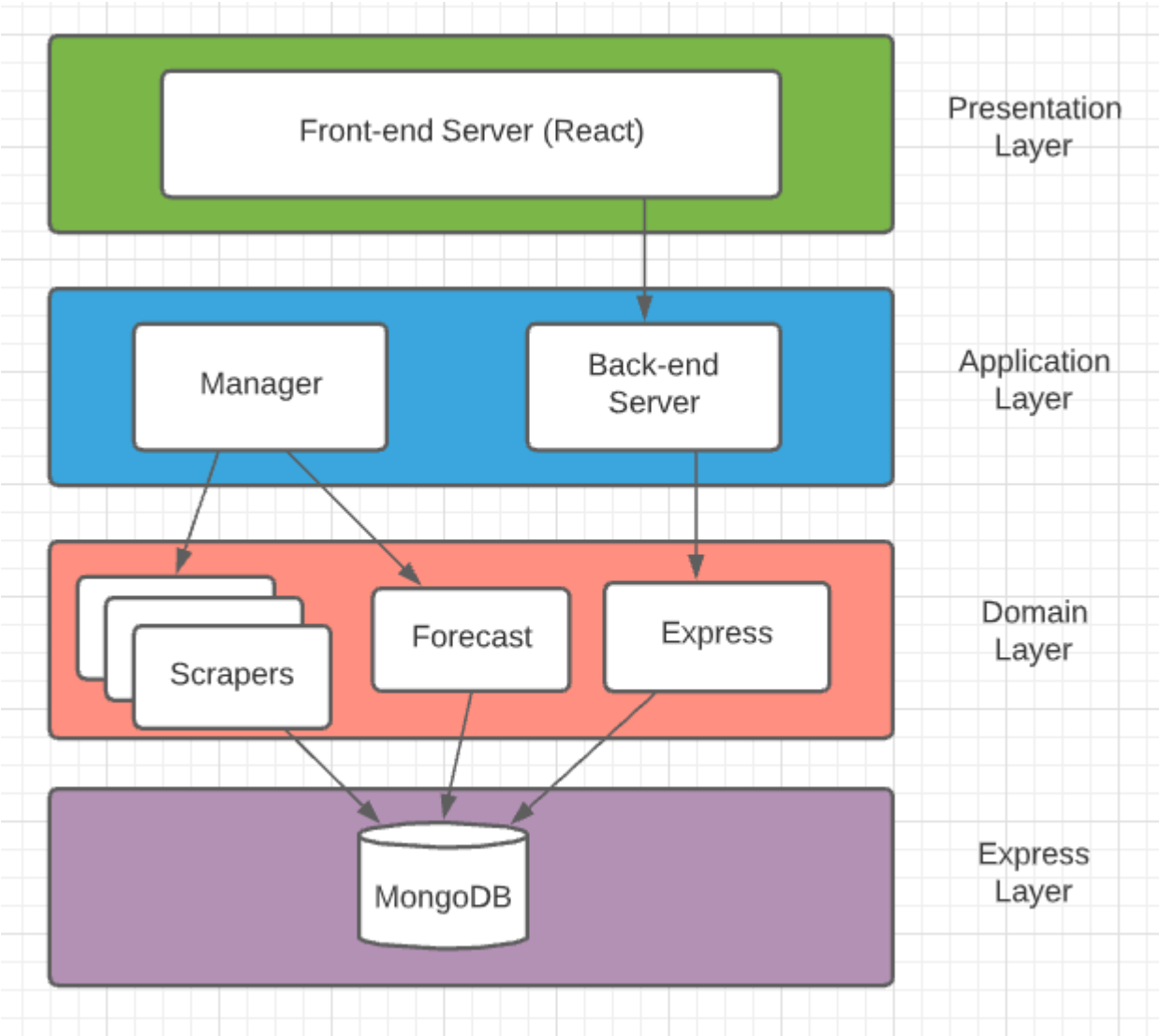


FIGURE 3.1

Figure 3.1 describes the overall architecture for this project which is based on a layered architecture, with all the dependencies flowing downward. The domain layer is open, meaning that some processes in the application layer will bypass it and communicate directly with our database. Overall each layer is independent with lower layers unaware of higher ones and should be unaffected by any changes higher up. The main benefit to this approach is that each of these layers classes can be isolated for better testing.

3.1.1 Presentation Layer

The front end server of our react based webpage will form the presentation Layer, the public user interactive layer. It will present a globe for our users to navigate, and upon country selection, will present data within a range that our user can change to the range of their choice. For dynamic data presentation, JavaScript components are rendered on the client side, after queries are pushed to our backend client and data is returned with express.

3.1.1.1 Visualization tools

React will be used for front end
Node.js and express are used for our back end.
D3 data visualization libraries.
Plotly will be used for interactive graphics.

The globe will be our main country navigation tool, and a calendar will provide the user input a time frame of data to present. Depending on whether the time frame selected includes both historical and/or forecasted data, we will present one or the other individually, or both side by side. Alongside an accuracy measurement of our forecast and an overall green/renewable energy pie chart. The main data will be presented as plotly area graphs, that have zoom in and out capabilities for the user.

3.1.2 Application Layer

3.1.2.1 Manager

This is the program responsible for automated scraping and forecasting. It will be dependent on the scrapers themselves. It will use the various scrapers for each country through an adapter. This enables a lot of flexibility in creating the scrapers, and the ability to potentially use scrapers that we did not write without directly modifying them. It will also be responsible for running the forecast class and running periodic testing on the accuracy of our forecasts.

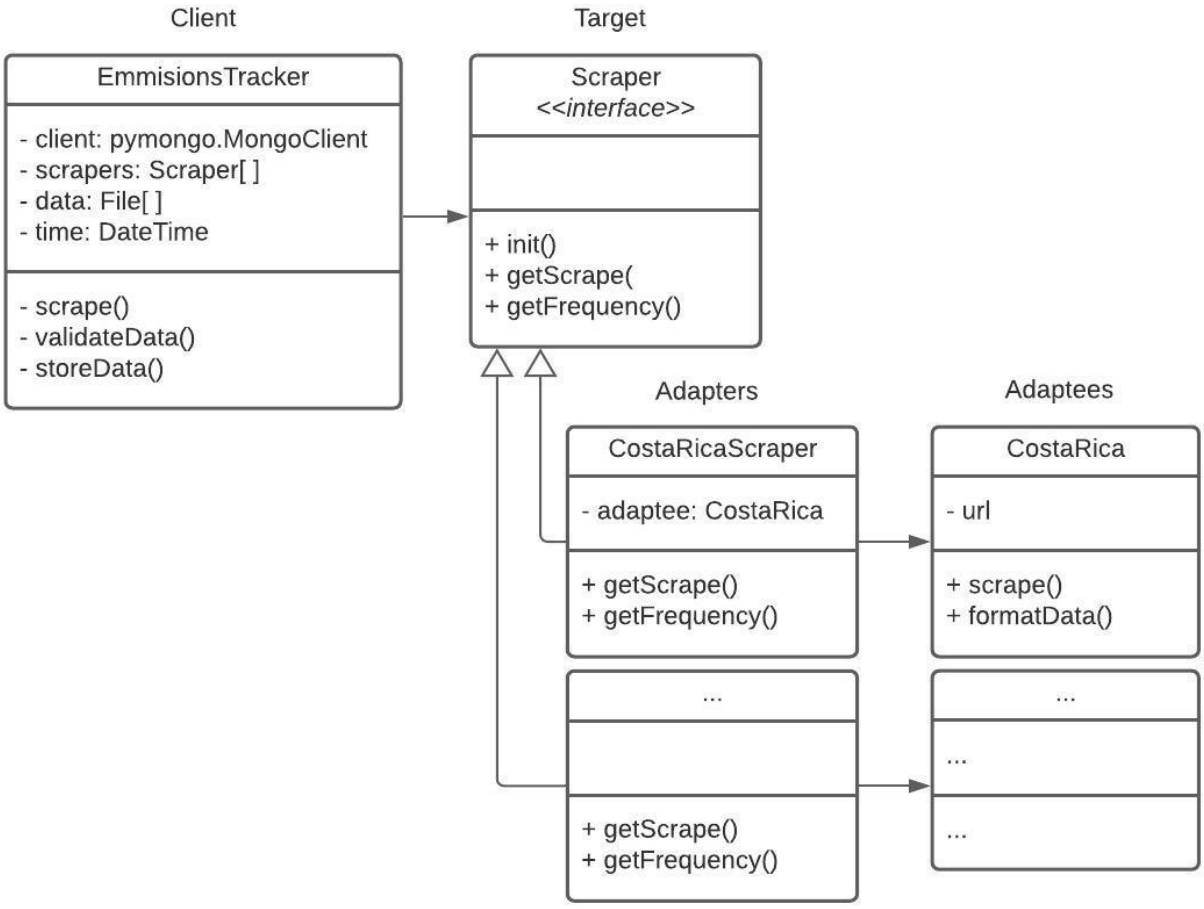


FIGURE 3.2

Figure 3.2 illustrates the design of the scrapers adapters

3.1.2.2 Back end Server

Serves as the authentication security between our front end server and our database. It runs as a separate application from the front end client server, and handles the queries the front end makes to our database, by being the direct connection to the database and retrieving the specific data requested.

3.1.3 Domain Layer

The domain layer consists of the business logic processes, which in our project correspond to the physical scrapers and the forecast class, and routing data.

Each scraper has a dependency on its corresponding website. These sites are prone to changes and as such the scrapers will likely require updates. Furthermore the scrapers themselves depend on a variety of outside libraries:

- BeautifulSoup
- Selenium
- Requests
- Pandas

The forecast tool will grab data from the Historical data stored in our MongoDB database and use it to create predictive time-series forecasting models. It will then be capable of cross-validating the data and producing statistical metrics of accuracy. Finally it will publish the results back to the database.It too will be dependent on various outside libraries.

- Pandas
- Pystan
- Fbprophet
- Scikit

Routing data and queries between front and back end servers are managed through express and API routes. Pushing using the router allows for two directional API's so queries and data can run down the same routes, easily connecting our front and back end server.

- Router
- Push
- Express

3.1.4 Persistence Layer

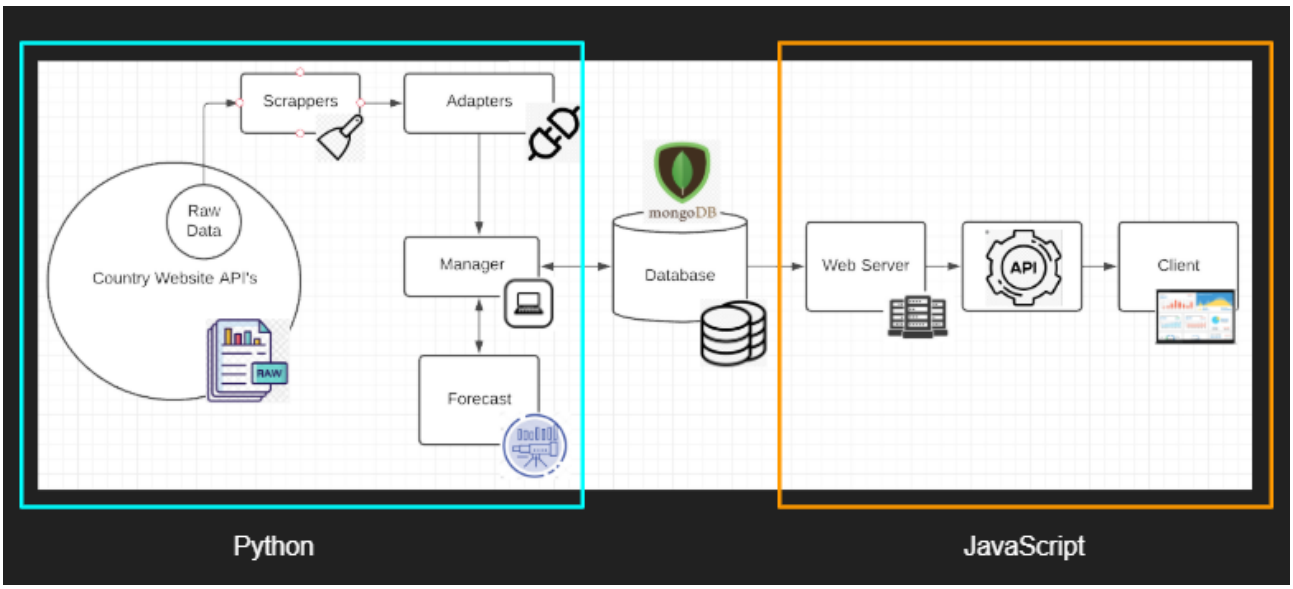


FIGURE 3.3

Figure 3.3 represents our data flow diagram. The persistence layer consists of the Database, in our case MongoDB, which is the center piece to the whole project. Data is gathered by scrapers filtered by adapters and organized into our database by the manager. Data is organized by country, each with two categories, historic and forecast. Forecasted data is based on the historic data run through FBprofit with our manager. The data is then pulled from mongoDB by our backed end server to be client side rendered after being routed through express API's.

3.2 Interface Description

3.2.1 Module Interfaces

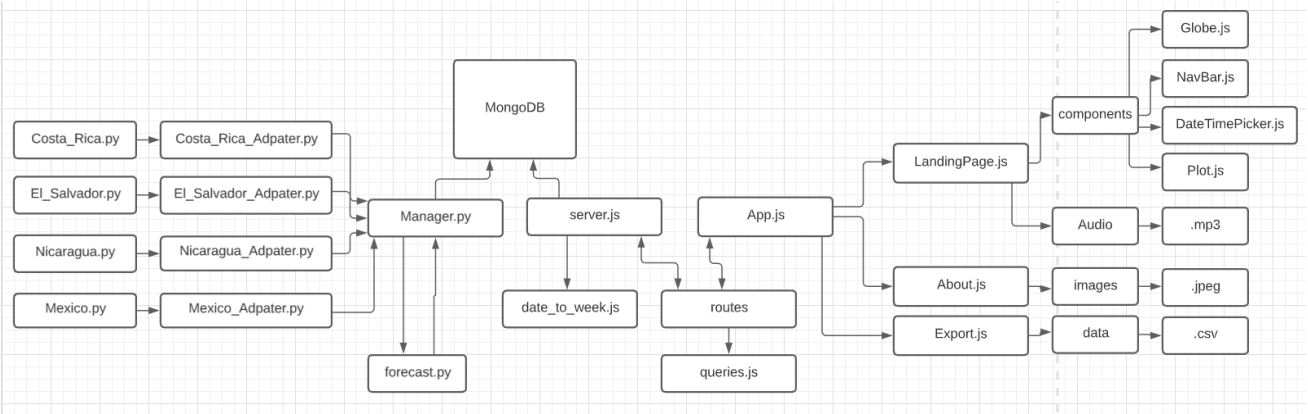


FIGURE 3.4

Figure 3.4 is the software interface that illustrates the relationship between our software components. From isolated scrapers built for the specification of WattTime, the adapters are built fitted to the scraper and the manager to run them alongside the forecast while concurrently uploading to the database. Then through our backend server upon query request from the front end routed through express which is finally rendered inside the components. There is a chain of dependence, illustrated by this figure.

3.3.2 User Interface

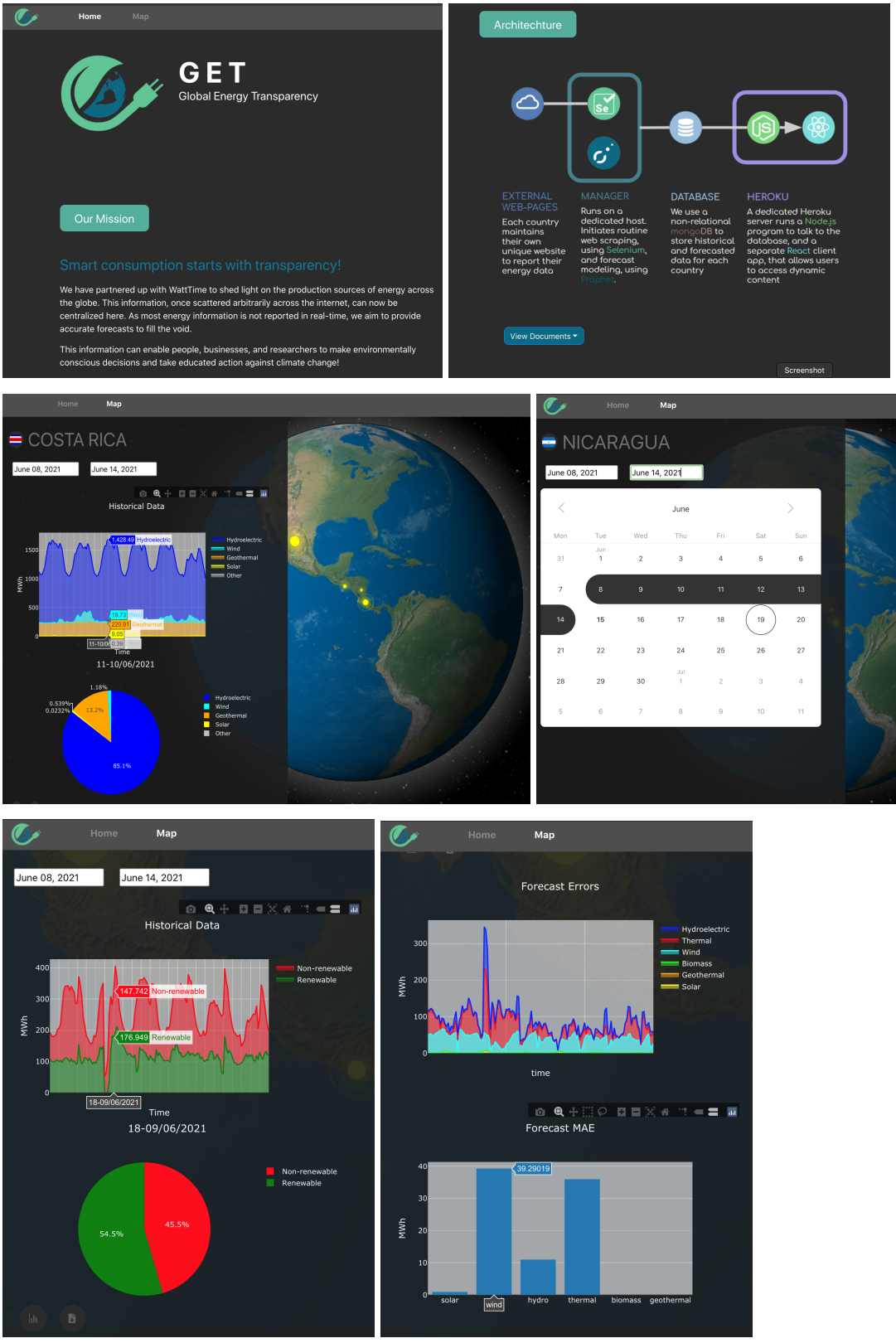

Home
Map
GET
Global Energy Transparency
Our Mission
Smart consumption starts with transparency!
We have partnered up with WattTime to shed light on the production sources of energy across the globe. This information, once scattered arbitrarily across the internet, can now be centralized here. As most energy information is not reported in real-time, we aim to provide accurate forecasts to fill the void.
This information can enable people, businesses, and researchers to make environmentally conscious decisions and take educated action against climate change!
Architechture
EXTERNAL WEB-PAGES
Each country maintains their own unique website to report their energy data
MANAGER
Runs on a dedicated host. Initiates routine web scraping, using Selenium, and forecast modeling, using Prophet.
DATABASE
We use a non-relational mongoDB to store historical and forecasted data for each country
HEROKU
A dedicated Heroku server runs a Node.js program to talk to the database, and a separate React client app, that allows users to access dynamic content
View Documents
Screenshot
COSTA RICA
June 08, 2021
June 14, 2021
Historical Data
Hydroelectric
Wind
Geothermal
Solar
Other
MWh
Time
11-10/06/2021
NICARAGUA
June
Mon
Tue
Wed
Thu
Fri
Sat
Sun
Non-renewable
Renewable
147.742
176.949
18-09/06/2021
54.5%
45.5%
Forecast Errors
Thermal
Biomass
time
Forecast MAE
39.29019
solar
wind
hydro
thermal
biomass
geothermal

FIGURE 3.5

Figure 3.5 is the GUI representation of our website API. There are two main tabs, home and map. The home or landing page, represented by the two main figures at top, is the about page explaining our website and our mission, and it's where the user starts. The Map page is the functional component of our webpage, what the user interacts with, and is the globe represented in the second row of images. A sidebar on the left side of the screen visualizes historical and forecasted data. The user can change the date range in the calendar. A pie chart is attached to the hover data on plotly as the user hovers the mouse over the plotted historical or forecasted data. There is a button under the graph that changes the data into a 2 part divide, renewable and non-renewable, instead of the individual production types (bottom left image). Lastly we have forecasting accuracy represented in the bottom right image, using an MAE model.

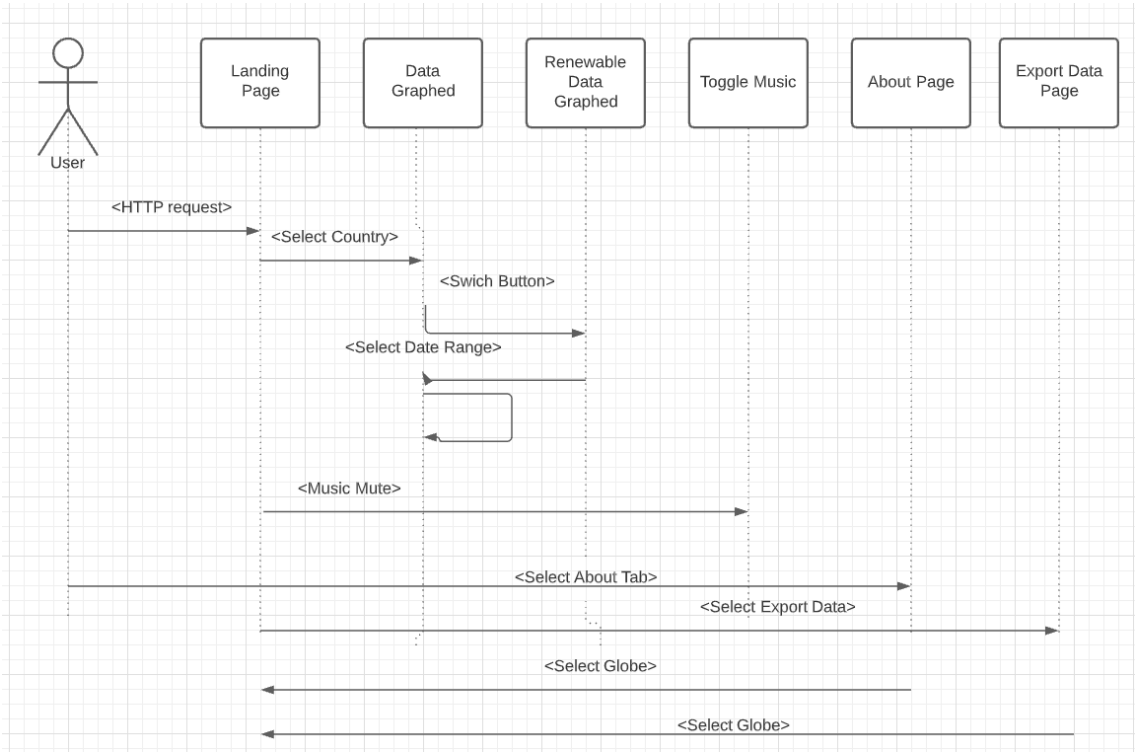


FIGURE 3.6

Figure 3.6 represents our use case diagram. The user first enters the landing page which shows the globe and listed countries. Upon selection of any country, data is represented for the user. The user may select a different data time frame. The user can also access our about page, and export data page, navigate back and forth between them and the landing page, as well as toggle the background music.

4. Detailed Design

4.1 Data Detailed Design

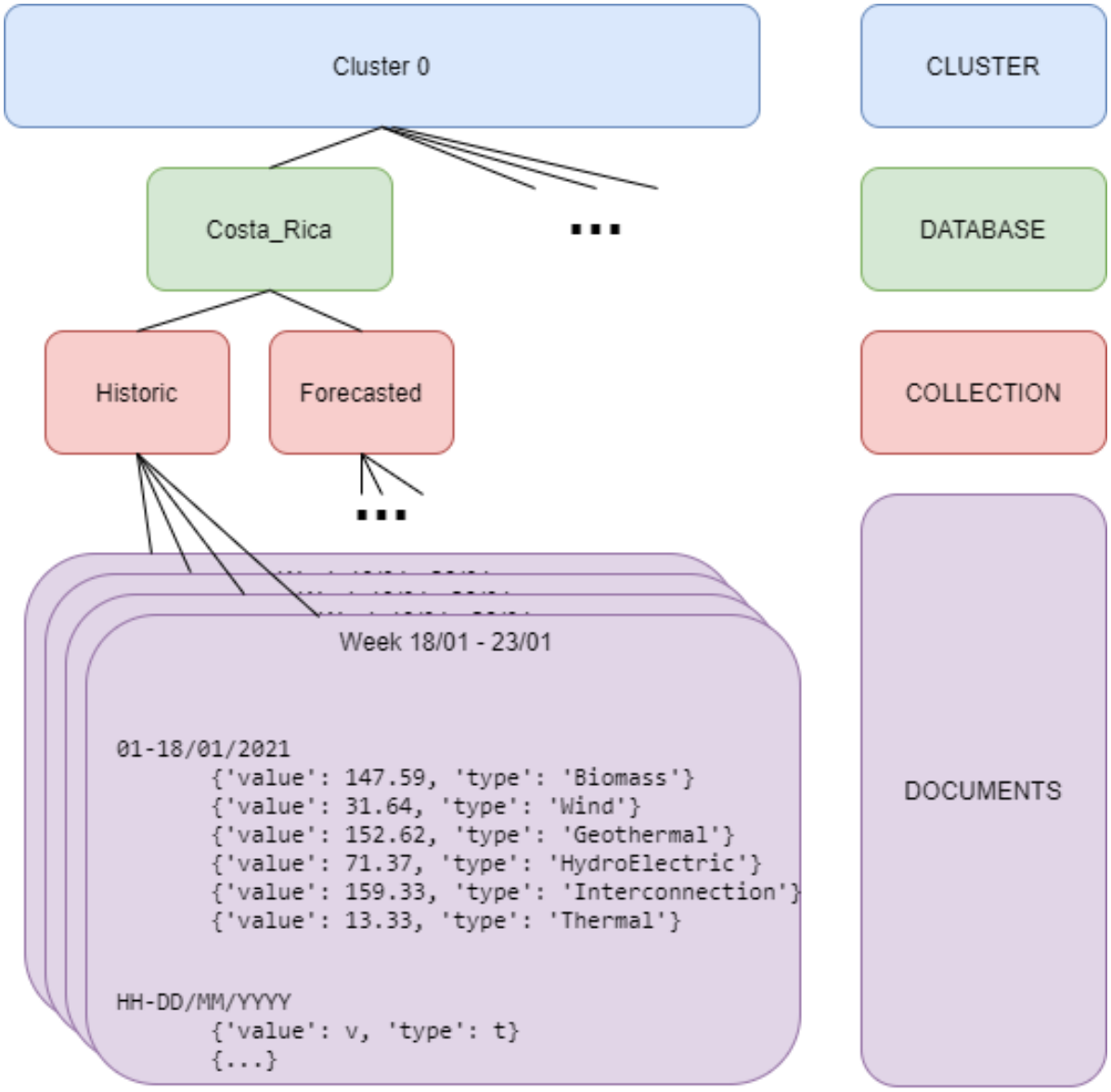


FIGURE 4.1

Figure 4.1 illustrates the design of our database. We will have a single cluster with a separate database for each country (currently 4). Each country will have two collections, one for historic data, and another for forecasted data. Finally within these collections will be json documents with datetime entries for each hour of each day in the scope of a week. The json documents will follow the format in the example document in the figure. The key will be the date time entry, which will return a small list of dictionaries containing the aggregate energy data values.

4.2 RTM (Requirements Traceability Matrix) (after system is defined)

Req. #	Requirement	Design Specification	Program Module	Test Specification	Test Case(s) Numbers	Successful Test Verification	Modification of Requirement	Remarks
1	Retrieve data in dictionary format.	3.2.1 Data Scrapers	3.2.1.1	Py test	60	60	None	Solid
2	Forecast Data	3.2.2 Forecasting	3.2.1.1	Py test	1	1	None	Solid
3	Select country	3.2.2 Visualization Tool	3.2.2.2	React	1	1	None	Solid
4	View past and future data	3.2.2 Visualization Tool	3.2.2.3	React	1	1	None	Solid
5	View clean vs dirty	3.2.2 Visualization Tool	3.2.2.4	React	1	1	None	Solid
6	Filter data by pollution type	3.2.2 Visualization Tool	3.2.2.5	React	1	1	None	Solid
7	About Page	3.2.2 Visualization Tool	3.2.2.6	React	1	1	None	Solid