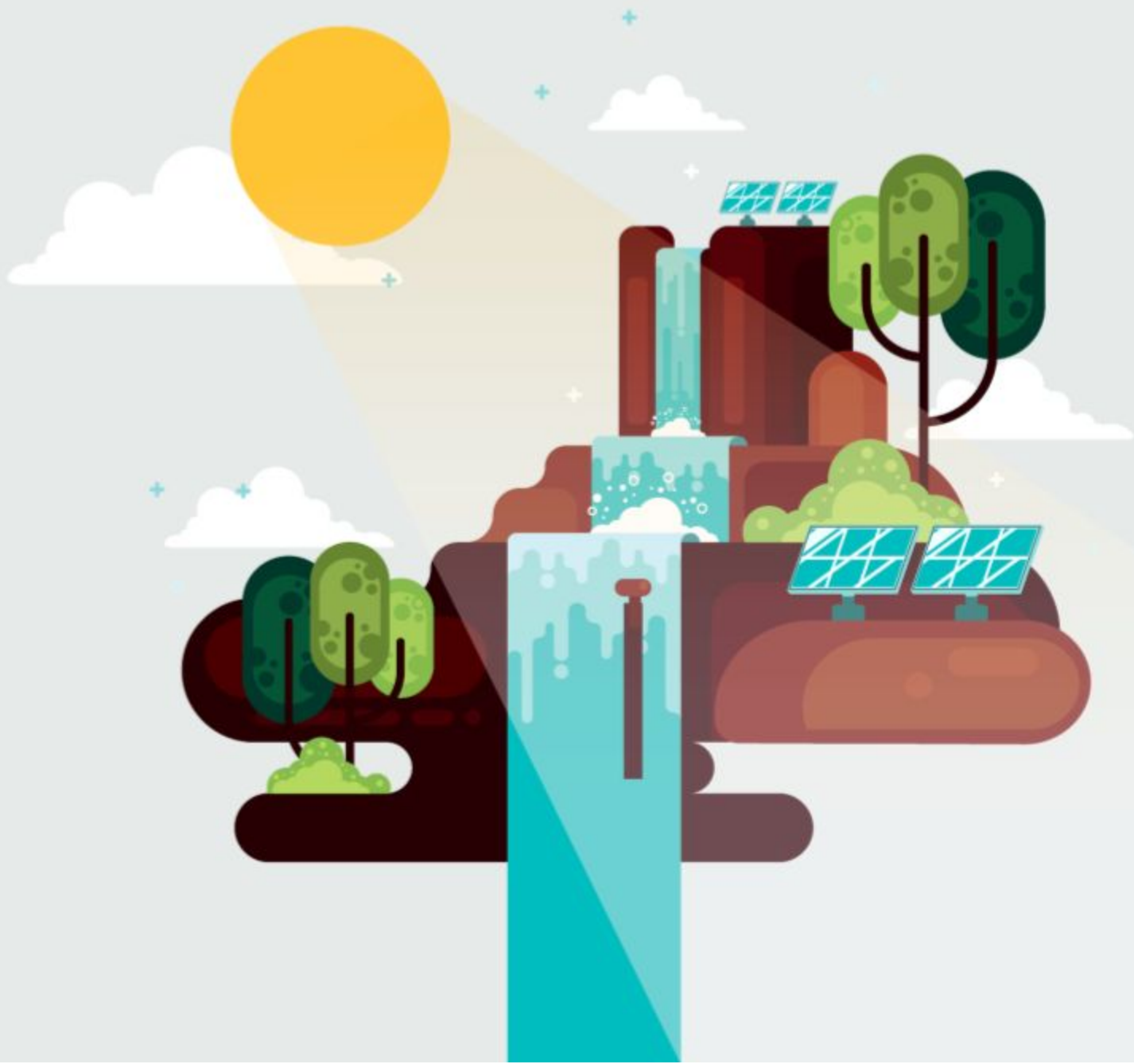


Team Members:

Alan Martinson, Carly Gloge, Hemant Bajpai
Mark McDonald, Pritam Dey, Taylor Meyer



Introduction and Background	4
Journal Selection	18
Changes from original requirements	20
Application Design	22
Developers Manual	25
Testing	25
Documentation	26
Project Software Tools	27
Estimates and Planning	29
Team Approach	30
Development Process and Lessons Learned Reflection	30
Risks	32
Appendix A - Requirements	34
Appendix B - UI Wireframes	58
Appendix C - ERM	62
Appendix D - Sequence Diagram	63
Appendix E - Sprint Estimations	64
Appendix F - API Documentation	67
Appendix G - Tests and Code Coverage	91
Appendix H - Application Documentation	94
Appendix I - Developers Manual	96

Introduction and Background

Abstract

The European Union (EU) aims to become the first climate-neutral continent by 2050 as outlined in their “Green Deal” initiative. As part of that effort, the EU has a goal of making climate-change data accessible for its constituents. While the EU has successfully aggregated large amounts of data through Eurostat, the European Environment Agency, Copernicus, and Europa.eu, the data is structured in different schemas and formats. Prior to our solution described in this paper, no platform existed which harmonized all of these disparate data sources into a holistic view that would allow a user to quickly understand how the EU is tracking towards their Green Deal goals. The “Green Deal Dashboard” addresses this need by aggregating seven unique data types into one application that reveals quick insights on progress towards the EU’s air quality goals. The application has been designed to be extensible to other data sources in the future to serve other Green Deal initiatives such as water quality and improvements in food production.

Keywords: EU Green Deal, Climate data, Visualizations, Air quality

Introduction

The European Union (EU) aims to become the first climate-neutral continent by 2050. As part of this initiative, the European Commission stated that “it will be important to ensure that across the EU, investors, insurers, businesses, cities, and citizens can access data and to develop instruments to integrate climate change into their risk management practices”¹.

Unfortunately, the EU is not yet meeting its goal of data accessibility for its constituents. 54% of EU citizens do not feel well-informed about EU air quality problems in their country (Directorate-General for Environment, 2019). Many business leaders have expressed similar

¹ European Commission, “European Green Deal Communication”, 2019, p.5 , https://ec.europa.eu/info/sites/info/files/european-green-deal-communication_en.pdf

sentiments expressing a lack of climate data understanding. “As long as we have no measurement data to measure emissions rates, we can’t do anything,” stated Heinrich Bovesmann, Managing Director at the Institute of Environmental Physics in Bremen in regards to access to air pollution data (Kean, 2019).

The statements above appear to allude to a lack of data, but the root issue seems to be a lack of accessible data. While there is significant climate data publicly available from Eurostat, the European Environment Agency, Copernicus, and Europa.eu, it is not available in a user-friendly format. Data are composed of formats that cater to technical users. For example, accessing air quality data may require writing SPARQL queries through the EU Open Data Portal, processing NetCDF data from satellite data, or reading in SHP files for regional boundaries based on the Nomenclature of Territorial Units for Statistics system which was developed by the European Union to group regional areas based on similar population sizes.

We created a platform that simplifies the complexity of understanding and acting upon scientific climate data. The application accumulates diverse types of climate data from a wide range of sources, and funnels the critical insights from the data into a streamlined, easily understood, actionable, and customizable front-end interface. Although the source data are aimed at scientific users, the platform caters to a non-technical audience and, more specifically, is directed at commissioners and leadership of the European Green Deal. The front-end organizes data into a series of dashboards, which include visualizations pre-filtered with data based on a user’s profile and then optionally refined by the user. Visualizations include interactive maps, charts, and tables. In addition to current and historical data, the platform presents comparisons to Green Deal target values for data points that have published attainment requirements.

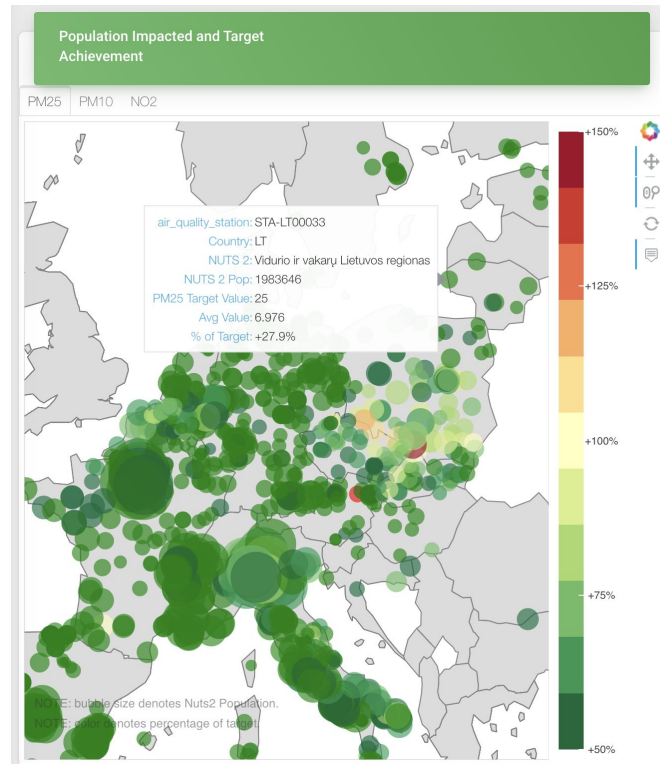


Figure 1. Application Screenshot: Bubble sizes represent population size while bubble colors denote target achievement for all EU measurement stations. Hoverable pop ups provide details for underlying bubbles.

Additionally, all data supported by the platform is available on a country and regional level as defined by the EU NUTS classification (Nomenclature of Territorial Units for Statistics). The application's novelty is in its ability to consolidate a flexible number of external data sources and enable users to navigate through a uniquely tailored series of interactive data visualizations. The "Green Deal Report" application described in this paper focuses on air pollution, but the application was built to accommodate additional types of climate-related data.

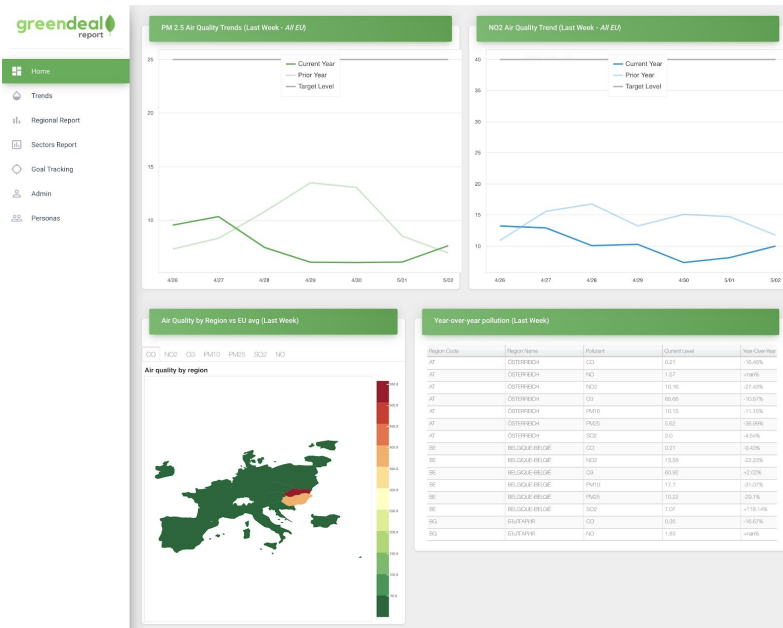


Figure 2. Application Screenshot: Home page provides a quick snapshot of key pollutants compared to targets for the past week.

Background

Many software developers and climate experts have created solutions that bring data together from various sources and integrate them in a user-friendly manner. Adler and Hostetler (2015) created the USGS National Climate Change Viewer which can generate time series plots and maps of temperature data (Figure 3). While their visualizations are user-friendly, the views lack granularity. For example, users can only view summarized data over 20 year periods. In addition, it only plots temperature data and does not provide data on underlying pollutants.

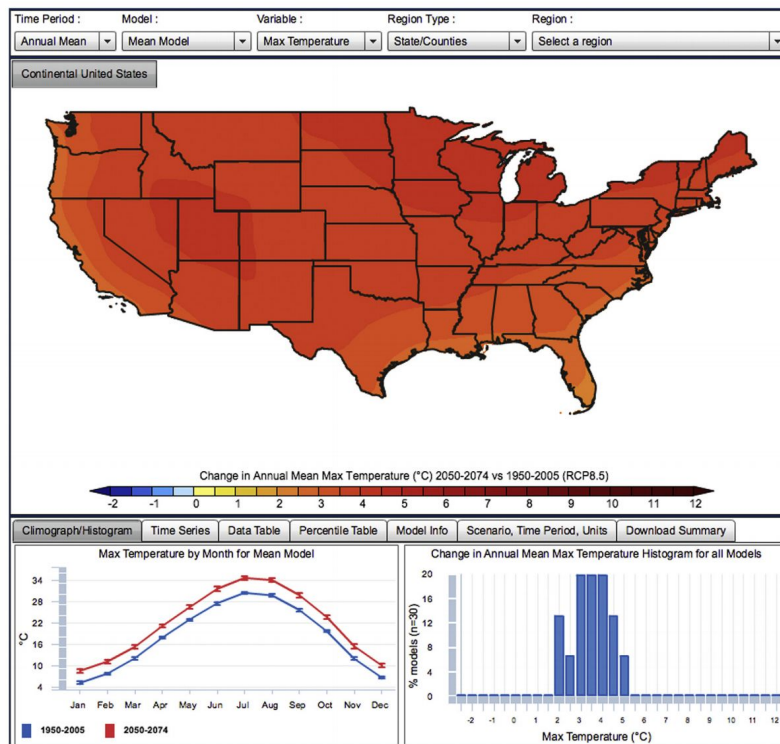


Figure 3. USGS National Climate Change Viewer displaying mean temperature data

Zeng, Chang, and Fang (2019) published a methodology that aligned with many of our philosophies to provide an accessible platform for users to view air quality data. The solution presented an ETL framework that extracts particulate matter data from the Taiwanese Environmental Administration into an intuitive map and tabular data. While this approach provides a useful baseline, it does not provide the end-user context on how those air pollution levels correlate to climate change goals. Another key differentiator for our solution is its ability to summarize pollution data by regional boundaries. We believed this was critical for EU commissioners to be able to make actionable decisions around adjustments that align with their long-term climate change goals.

An approach by Tominski, Donges, and Nocke (2011) outlined a visualization for climate change data. While their proposed solution leverages satellite data, their solution, like the USGS National Climate Change Viewer, does not provide information on the underlying pollutants that correlate with climate change. Although the visualizations (Figure 4) in this application serve as

an inspiration for our effort, the level of data presented is more technical than is intended for our non-scientific target audience.

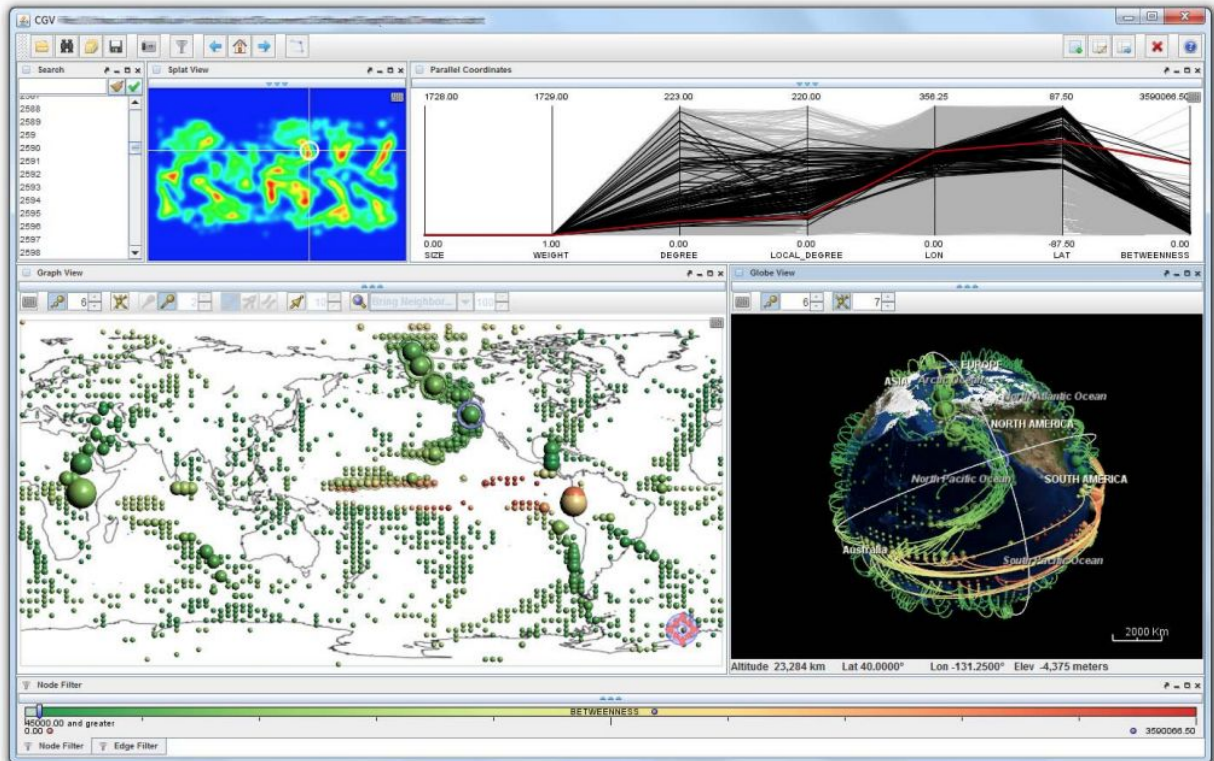


Figure 4. Coordinated views as provided by the system outlined in “Information Visualization in Climate Research”.

In addition to the academic journal papers cited above, several current and publicly available software tools provide visualizations of global atmospheric composition data, including the Copernicus website, the World's Air Pollution: Real-time Air Quality Index, the Panoply Viewer by NASA, and the Quantum Geographic Information System (QGIS). While all of these tools have user-friendly interfaces that allow non-technical users to view air quality data, they do not meet the core need of quickly viewing summarized air quality levels at a regional level and comparing them to targets.

European Commissioners need the ability to view data on the following pollutants: ozone, nitrous dioxide, sulfur dioxide, CO, PM10 aerosol, PM2.5 aerosol, wildfire Particulate

Matter (PM) and dust. This information plays a critical role in allowing commissioners to find the root cause of air quality changes. The Copernicus website has data for individual pollutants, but it doesn't provide a user-friendly air quality index. The World's Air Pollution: Real-time Air Quality Index website (<https://waqi.info>), on the other hand, provides a user-friendly air quality index of various points on a world map, but it doesn't break down the individual pollutants.

The Copernicus website and Panoply Viewer can display all necessary pollutant data (Figure 5). However, both tools only render country or continent boundaries and cannot provide summary statistics based on regional boundaries.

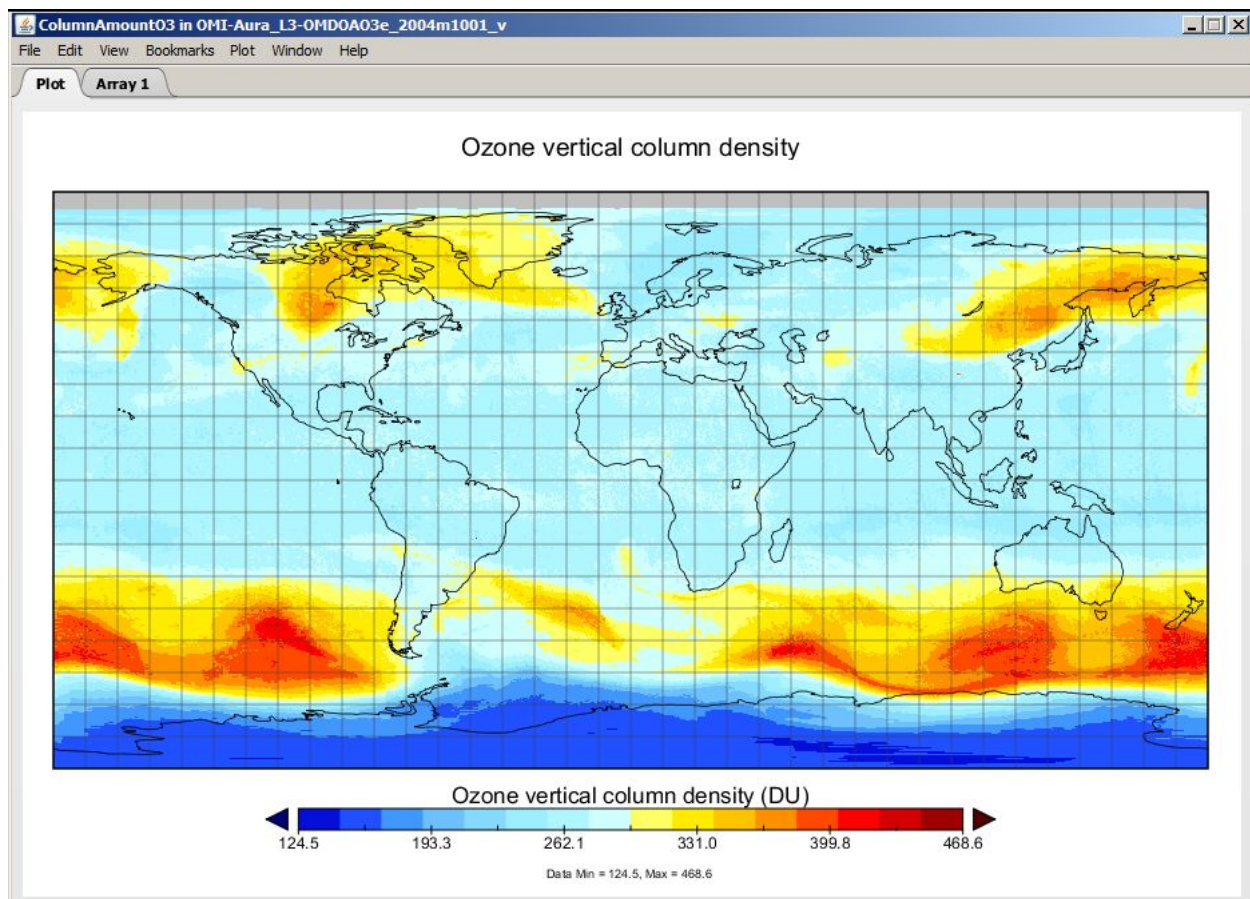


Figure 5. The Panoply viewer rendering global ozone levels.

QGIS can provide summary statistics as defined by a regional boundary but requires the end-user to perform spatial joins which would require installation of specialized software like QGIS or ArcGIS, importing raw SHP files and pollution data, defining the vector data to be

summarized, defining the layers to be joined, and finally grouping the data (QGIS Tutorials, 2020). A spatial join is a geographical information system operation that joins attributes from one feature to another based on the spatial relationship between them—although reasonable for a scientist, expecting the target audience of our platform to perform these actions is unrealistic. Additionally, the task of accumulating this data limits any user, expert or not, from performing any quick 'on-the-fly' analysis. In our testing, it took several minutes to download the original raw data before any consolidation of data sources. Although the process may yield a high value for scientific analysis, our objective is to provide a higher-level of summarized results using interactive visualizations.

In a study of best dashboard design practices, researchers found that dashboard users showed "no interest in, for example, most metadata items, schema names or transformation constraints, which are found between the data source layer and business layer" (Presthus, Canales, 2015). The authors discovered that users valued having the ability to drill down within reports, but found that "it is not necessary to go all the way down to the data lineage/data source layer. It is sufficient to go down to the business layer". We believed that the EU commissioners would have a similar preference since there's a wide range of scope and granularity under each commissioner's purview. They also need to be able gain insights quickly, so exposing any underlying data schema would likely lead to a less than ideal user experience. With that in mind, we determined that standard business intelligence tools on the market would not be sufficient for providing different levels of drill-down capabilities for the various commissioner personas. While Tableau, Power BI, Qlik, Looker, and Data Studio all can support drill-downs, they do not support configuration of drill-down depth based on a predefined user preference.

Some critical detractors for dashboards included manual data entry, the inability to drill up or down, interfaces that were difficult to use, and slow speeds (Bugwandeem, Ungerer, 2019). Our solution addresses each of these issues by automating the collection of data from each original data source, providing drill-down capabilities, designing the interface with the user in mind, and storing the data in a format optimized for fast access.

After reviewing solutions currently available in the market as well as those outlined in academic journals, there is no user-friendly interactive climate data portal, which enables

comparison to targets, let alone one that supports the EU Green Deal. We hope that this dashboard serves as a functional tool that can be extended to meet future needs for a set of climate change data.

Our solution provides simple visualisations immediately after the user logs in. For example, on the dashboard homepage, the user can view air quality data over the past week, a regional heatmap by pollution channel of EU countries and the year-over-year change by pollutant and country (Figure 6).

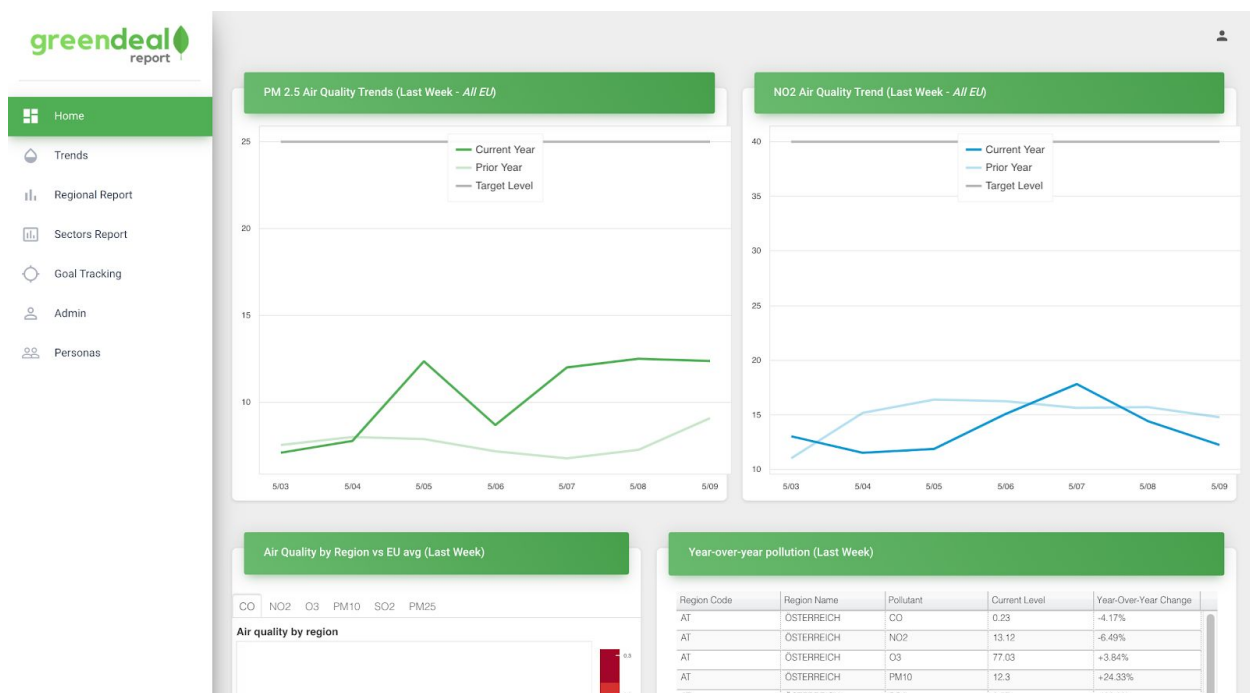


Figure 6. The Green Deal Dashboard homepage

If the user would like to drill down further, they can click on pages like the Trends page to change the specified time range and region to see how air quality has changed over time. This view proves to be especially relevant during the time this paper was written since shelter-in-place orders have been in effect across the EU due to COVID-19. With only a few clicks a Green Deal dashboard user can see that most air quality pollutants have decreased

year-over-year during the month of April (Figure 7).



Figure 7. Air quality trends view by as measured by pollutant during the month of April

Green Deal dashboard users can also gain more granular insights on air quality via map views. For example, on the Goal Tracking page, users can view readings from individual pollution reading stations across the EU and see by simple color coded bubbles whether that a particular area's readings are above the pollutant goals for that year or below. The size of the bubble also indicates the population of that region so the user can see how much impact the air quality is having on citizens. During the month of April, 2020, forest fires consumed 6,000 hectares of Biebrza National Park. The impact of those fires can be clearly seen in the PM2.5

readings of the trends map during that time (Figure 8).

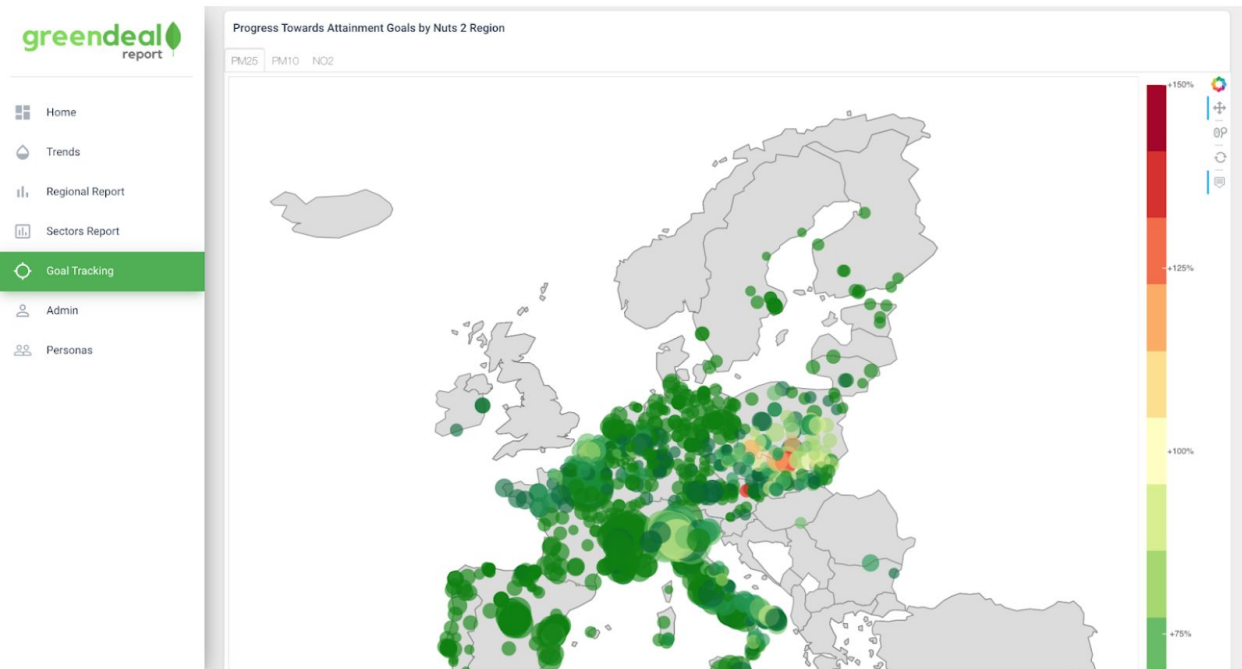


Figure 8. The Goal Tracking view shows air quality readings vs the EU's target pollution levels. High levels of PM2.5 can be seen in Poland where forest fires were active in April, 2020.

While gaining insights on current air quality levels is useful for tracking progress, commissioners and citizens also need to understand the leading causes of emissions. The Sectors report of the dashboard gives insights on which industries are emitting each major pollutant (Figure 9).

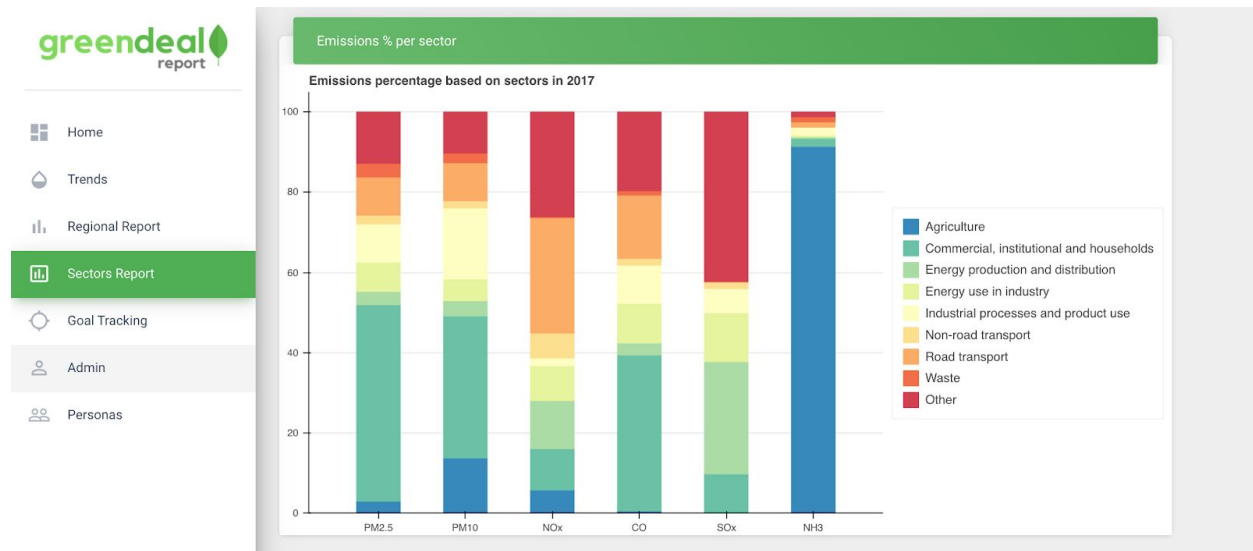


Figure 9. The Sectors report shows which industries are emitting each pollutant type in aggregate and in trended views. Users can filter these views by country and year.

While the Green Deal dashboard has already provided useful insights on the impact of shelter-in-place orders and recent forest fires on air quality, its value will increase as more users leverage it to answer questions they may have around the impacts of policy changes or their own personal actions.

The next progression of this dashboard would be to extend it to add additional data sources which pertain to other initiatives of the Green Deal such as water quality and clean food production. In addition, this dashboard could serve as a helpful tool for European citizens to feel more connected to their impact on climate change. In order to support a large number of users, further infrastructure development work would be necessary to increase the speed of the platform such as: database optimisations, setting up a CDN, implementing a headless front-end, etc. Once this work is done, this platform will be able to act as a robust tool for EU commissioners and citizens to monitor their progress towards their goal to become the first climate-neutral continent by 2050.

References

- Alder, J.R. Hostetler, S.W. (2015) Web based visualization of large climate data sets. *Elsevier Environmental Modeling And Software* - 68
- Bugwandeen, K. Ungerer, M. (2019) Exploring the design of performance dashboards in relation to achieving organisational strategic goals - 163
- Copernicus (2020). Air Quality in Europe. Retrieved from:
http://macc-raq-op.meteo.fr/index.php?category=ensemble&subensemble=hourly_ensemble&d ate=LAST&calculation-model=ENSEMBLE&species=o3&level=SFC&offset=000
- Directorate-General for Environment. (2019). Eurobarometer - Attitudes of Europeans towards Air Quality. Retrieved from:
<https://ec.europa.eu/commfrontoffice/publicopinion/index.cfm/Survey/getSurveyDetail/instrumets/SPECIAL/surveyKy/2239>
- The European Commission. (2019). The European Green Deal. Retrieved from:
https://ec.europa.eu/info/sites/info/files/european-green-deal-communication_en.pdf
- Eurostat. (2019). NUTS - Nomenclature Of Territorial Units For Statistics. Retrieved from:
<https://ec.europa.eu/eurostat/web/nuts/background>
- Kean, Dave. (2019). Satellite Data Offers New Hope For Taming Oil's Methane Emissions. Retrieved from:
<https://www.euractiv.com/section/energy/news/satellite-data-offers-new-hope-for-taming-oils-methane-emissions/>
- Presthus, W, Canales, C. (2015) Business intelligence dashboard design. A case study of a large logistics company. *NOKOBIT Vol 23* - 12
- My Climate. (2020). What is climate neutrality? Retrieved from My Climate:
<https://www.myclimate.org/information/faq/faq-detail/detail/News/what-is-climate-neutrality/>
- QGIS Tutorials (2020). Performing Spatial Joins (QGIS3). Retrieved from:
https://www.qgistutorials.com/en/docs/3/performing_spatial_joins.html
- Tominski, Christian, Donges, Jonathan F, Nocke, Thomas (2011) Information Visualization in Climate Research. *2011 15th International Conference on Information Visualisation* London, UK
-

Zeng, Yu-Ren, Chang, Yue Shan, Fang, You Hao (2019) Data Visualization for Air Quality Analysis on Bigdata Platform. *2019 International Conference on System Science and Engineering (ICSSE)* Ho Chi Minh City, Vietnam

Journal Selection

Identifying a viable journal for the project is imperative to overarching success and the respective audience that the target application can reach. Journal selection was made based on three overarching factors:

- (1) Geographical proximity to the target user demographic,
 - (2) Overall topic synergy derived from the research and data driving the product solution,
- and
- (3) Pricing and submission requirements for eligibility

Journal of Environmental Planning

We chose the Journal of Environmental Planning and Management as the target publication for the effort described in this document. This journal is backed by the broader organization, Taylor and Francis Online. The publication has a global reach, with contributions from leading authors and is published annually. Topics covered in the journal align well with the product objectives of the development project in focus. The international scope of the publication may help to inspire other geographies across the globe to learn from and emulate a similar endeavor to be net emission neutral.

The topic objective aims to accumulate prioritized data from a diverse set of resources and produce a unified, simple, and measurable dashboard of data output for commissioners. This objective aligns with the aim and scope of the Journal of Environmental Planning and Management. Synergies exist between the product scope and journal with a mutual emphasis on promoting an enhanced perspective on environmental issues, developing knowledge on the causes of environmental change, and implementing technology and innovation as a solution. We employed a particular focus on the journal's innovative management methods category.

Lastly, pricing and submission requirements were analyzed to verify compatibility and feasibility. The Journal of Environmental Planning and Management supports format-free submissions, with open scholarly and reference styles accepted. The verification process also proved to be reasonable, with eight total steps to publications:

- (1) Identify specific Taylor & Francis Journal (Journal of Environmental Planning and Management)
- (2) Write a draft of the article
- (3) Verify submission requirements (clarity, structure, length, format, references, permissions, and sharing policies)
- (4) Assessment by the journal editor
- (5) Peer review
- (6) Journal acceptance decision
- (7) Approval or denial
- (8) Published paper

There are no submission or publication fees required for the Journal of Environmental Planning and Management.

Given the journals subject matter, credibility, global publication scope, and lack of publishing charge, The Journal of Environmental Planning and Management meet the project's objectives as a publication target.

EU Datathon 2020

An alternative approach towards publication is to enter the application into a contest. The EU is hosting the EU Datathon 2020 which encourages entrants to submit applications and ideas aimed at address one of 4 areas:

1. A European Green Deal
2. An Economy that Works for Everyone
3. A New Push for European Democracy
4. A Europe Fit for the Digital Age

The contest requires that the application make direct use of the data sources that are already being used in the application. As this platform directly addresses the Green Deal, we have entered the application into this contest in the "Green Deal" category.

Changes from original requirements

Data Sources

Initially we were targeting a single data set from each of 3 data providers as mentioned in the M1 report which were Copernicus, EEA and Eurostat. We were planning to get RDF-data stored in the EEA (European Environment Agency) data source, REST-based data stored in the EuroStat data source and CSV-based data stored in the Copernicus data source. As we made more progress in the project, we realised the importance of additional datasets from each data provider in order to add more value to the dashboard. Currently we are pulling following data sets:

- Copernicus:
 - Satellite images of air quality data via REST requests with NetCDF file payloads
- EEA:
 - Observation station location via REST requests with CSV payload and Air quality sensor data from observation stations via REST requests with CSV payloads
 - Emissions by sector via SPARQL requests with RDF payloads
 - Air quality goals added manually for entire Europe
- Eurostat:
 - Region boundary polygons via REST requests with a GeoPandas shape file payloads
 - Population information via REST requests with CSV payloads

Plots and Dashboard

We added more plots and dashboards than were initially proposed in the original requirement section. Changes in plots and dashboards are:

- We added a bubble map in the Goal Tracking page which shows progress towards attainment goals by nuts 2 region. Here the bubble size relates to population in that region which shows how much population is under attainment or above attainment goals. This plot shows how this platform is able to merge data from 2 different data sources (goals from EEA and population from Eurostat) to show something meaningful in the dashboard.
- Sectors dashboard was added which shows emissions based on 9 major sectors. Initially it shows complete europe data but a region filter allows the user to see plots for a particular region. Below is the list of plots added in this dashboard:

- Emission percentage based on sectors in 2017: This shows emissions percentages for all 9 sectors for 6 pollutants which are PM2.5, PM10, NOx, CO, SOx and NH3.
- Yearly emissions of main pollutant: This shows the overall emission trends of 6 pollutants which are PM2.5, PM10, NOx, CO, SOx and NH3 from 1990 till 2017.
- It includes 6 more plots which shows how the total emission trend shown in above plot is distributed among sectors for PM2.5, PM10, NOx, CO, SOx and NH3 from 1990 till 2017.

Technical Stack

- **Simplified data flow**

Initially we were thinking of pulling data from various data sources and putting it to a temporary data store and then pulling it from there and putting it into our database. While there can be benefits to temporary data stores, we decided to simplify the process and started pulling data from data sources and putting it directly to the database.

Visualizations

- **Removed Angular**

We started with bootstrap and we did not feel the need to move to angular. We are able to show UI features we required with bootstrap.

- **Added Bokeh**

Bokeh is an interactive visualization library for modern web browsers. It is helpful in making interactive plots, dashboards and data applications. Charting elements can easily be plotted and can easily be surfaced without writing a lot of UI code.

External API

An external API is an API designed for access by a larger population as well as web developers. Because of the benefits below we decided to create external API for our application:

- This provides a way to visualize the data we are getting from the server side.
- With API design we have decoupled logic from representation.
- Using JSON provides a more familiar interface for developers.

AWS

We have used AWS EC2 to process and load data as a part of the ETL pipeline instead of AWS ECS Fargate. This was done because the incremental data load process was simple enough to run on a very small or free tier instance. One of the data sources required us to use AWS Auto-Scaling with AWS EC2 spot fleet instances to scale the data load as well.

We did end up using AWS ECS Fargate to host the web server (<https://www.greendealdashboard.com>). This helped our architecture because our webserver is built using a docker image. To deploy the docker image AWS ECS Fargate seemed to be an ideal candidate. This webserver is load balanced and can be scaled up very easily by changing options within AWS console.

To host the website we have used AWS Route53 where we registered a domain name: greendealdashboard.com and created a route to the Load balanced web server endpoint.

Application Design

Logical Software Design

This project requires an application which brings data from different sources together and provides visualizations to the end-user for analyzing EU air pollution data. The architecture supporting this application is divided into 3 layers:

1. **Model/Data**

This layer is responsible for fetching and storing data from multiple external sources at a scheduled interval based on the data refresh interval of each particular external source. The Model layer is designed so that data values from various sources are tied together using foreign key relationships between the data. A diagram of the data's entity relationship model is included in the appendix ([Appendix C - ERM](#)).

Backend Database

Using settings in the Django framework, the backend database source can be configured to run on a publicly hosted service or in a private subnet. The initial prototype was developed using a local Docker instance with Postgres as well as using AWS RDS also with Postgres. This database serves as the only data store for this project.

Data Transformation

As data is loaded, it is transformed according to the specifics of the application including, but not limited to: scaling, setting/changing data types, correcting errors, and adding calculated fields. The processing time spent at this point is meant to alleviate the burden of data processing for subsequent layers in the architecture. For example, 78 EEA sector codes collected through the RDF data source are categorized into 9 major sector groups which are displayed in the application. Other data transformations include the averaging

of hourly satellite images over a day, as well as, removing ground observation station pollution measurements that are labeled as inaccurate.

Update Interval Scheduling and Storage

Each unique data source is updated at intervals that align with the data source's update schedule. All the incremental data updates can be handled within a single AWS EC2 instance except one datasource. To get observation station reading data, we needed to scale the data loading process. We have used AWS EC2 Spot fleet with AWS Autoscaling to load data from this datasource. The number of concurrent instances which will be loading the data can be controlled within the Autoscaling options. This has been kept as a fixed number of instances and can be configured to scale up or down using the depth of AWS SQS from which the worker nodes are listening to the messages and processing them. Configuring the autoscale up and down is out of scope of this project but can be easily achieved with the current architecture.

2. **Controller**

This layer is responsible for retrieving data from the various sources stored in the backend database and consolidating them into data structures which can be used by the front end for visualization. The Controller hides the complexity of accessing the wide variety of data stored in various tables in the application database. The Controller allows the front-end to focus on presenting meaningful visualizations based on the application's data avoiding the complexity of navigating the specifics of the database.

REST API

The controller exposes backend data through a REST API. This API is used by the front-end of the application but can also be accessed directly through HTTP requests.

3. **View**

The View layer is the front end UI of the application. This layer is responsible for presenting data to the user, authenticating access to the application, and administration tasks performed by the application administrator.

End-User Visualizations

The View layer allows the user to navigate the site and filter data which is presented in multiple charts and visualizations required to do EU air pollution analysis. The appendix includes front-end UI wireframe examples that are presented by the View layer.

Authentication

Users will not self-register for access to the services of the application. Instead, an application administrator will manually create new user accounts. After an account is

established, a user can access the application using an assigned user id and password.

Application Administration

The application design includes the ability for each user to be assigned to a ‘persona’ which will dictate the default data that is presented to the user. Each user is also assigned a default region for which data is pre-filtered. The application supports a process that allows an application administrator to create and change ‘personas’ and assign ‘personas’ and default regions to users.

Superadmin

The application supports a ‘superadmin’ user role that will be used to navigate the database structure. This role is primarily for developers of the application.

Technical Architecture Design

The technical architecture diagram below summarizes the applications and services used to deliver the application services:

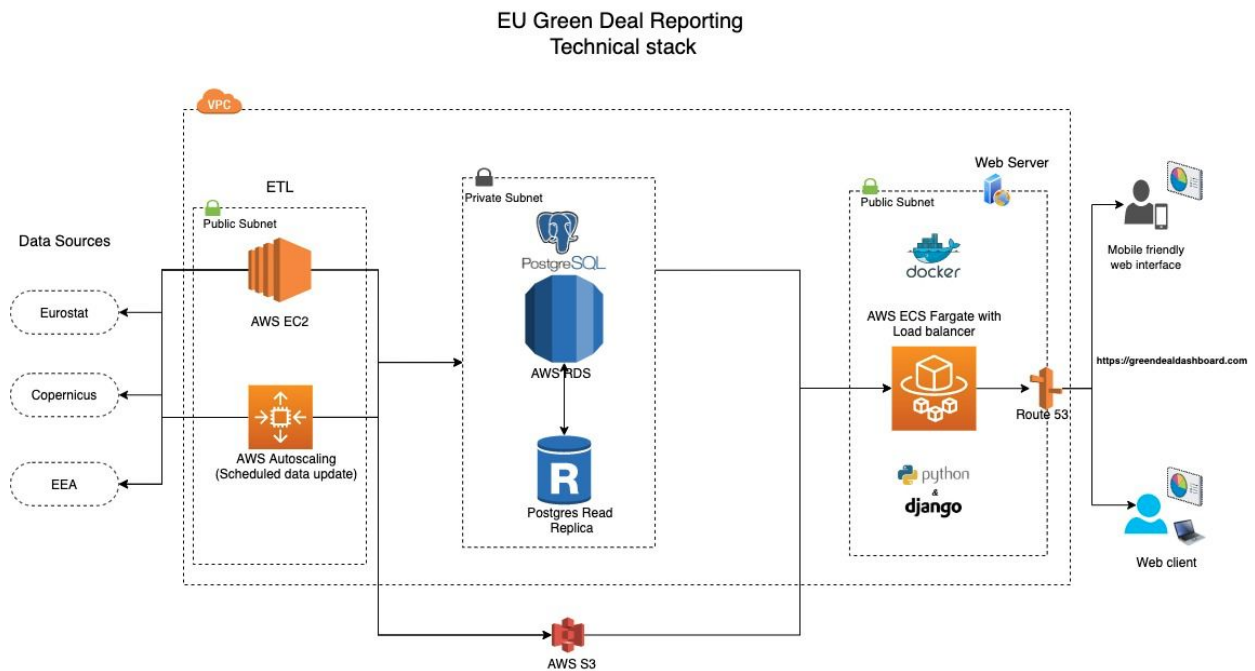


Figure 1. Technical Stack

Technical component	Service/Platform
Language	Python (Customer request)

Cloud Provider	AWS
Database	AWS RDS (Postgres)
DNS Hosting	Route 53
Data ingestion	AWS EC2, AWS Autoscaling
Raw Storage	AWS S3
Web server	Python Django, AWS ECS Fargate (Load balanced)
Front-end	Javascript, jQuery, Bokeh, Bootstrap
Desktop Browser Support	Chrome 80+

Developers Manual

A developers manual is included in [Appendix I - Developers Manual](#) that explains how to set up the application in your local environment.

Testing

The application uses the Django TestCase library to run unit tests and the Django coverage plugin to check our test coverage. Instructions on how to run the tests and generate the test coverage report can be found in [Appendix G](#).

Coverage report: 81%

Module	statements	missing	excluded	coverage ↓
/eugreendeal/airpollution/models/models_nuts.py	100	52	0	48%
/eugreendeal/airpollution/models/models_copernicus.py	58	21	0	64%
/eugreendeal/airpollution/models/models_eea.py	45	12	0	73%
/eugreendeal/airpollution/models/models_observations.py	208	52	0	75%
/eugreendeal/eugreendeal/settings.py	32	8	0	75%
/eugreendeal/manage.py	12	2	0	83%
/eugreendeal/airpollution/models/models_pollutants.py	121	13	0	89%
/eugreendeal/airpollution/models/models.py	20	2	0	90%
/eugreendeal/airpollution/__init__.py	0	0	0	100%
/eugreendeal/airpollution/admin.py	47	0	0	100%
/eugreendeal/airpollution/apps.py	3	0	0	100%
/eugreendeal/airpollution/management/__init__.py	0	0	0	100%
/eugreendeal/airpollution/migrations/0001_initial.py	8	0	0	100%
/eugreendeal/airpollution/migrations/__init__.py	0	0	0	100%
/eugreendeal/airpollution/models/__init__.py	7	0	0	100%
/eugreendeal/airpollution/models/models_eurostat_population.py	15	0	0	100%
/eugreendeal/airpollution/tests/__init__.py	0	0	0	100%
/eugreendeal/airpollution/tests/eeamodeltests.py	13	0	0	100%
/eugreendeal/airpollution/tests/eurostatmodeltests.py	12	0	0	100%
/eugreendeal/airpollution/tests/nutsregions_tests.py	18	0	0	100%
/eugreendeal/airpollution/tests/observations_tests.py	36	0	0	100%
/eugreendeal/airpollution/tests/pollutants_tests.py	65	0	0	100%
/eugreendeal/airpollution/urls.py	16	0	0	100%
/eugreendeal/eugreendeal/__init__.py	0	0	0	100%
/eugreendeal/eugreendeal/urls.py	4	0	0	100%
Total	840	162	0	81%

coverage.py v5.1, created at 2020-05-02 20:38

Figure 2. Test Coverage

Documentation

We used the Sphinx document generator to generate documentation. The document is in the code repository under `eugreendeal/docs/index.html`. Instructions on how to generate the documents can be found in [Appendix H](#).

The screenshot shows a documentation page for the 'airpollution.management.commands.load_pollutants module'. On the left is a navigation sidebar for 'EUGreendeal' with a search bar and a 'CONTENTS' section listing various packages and submodules. The main content area displays the module name, author (Mark McDonald), and details for the `airpollution.management.commands.load_pollutants.Command` class, including its base class (`django.core.management.base.BaseCommand`), methods like `add_arguments(parser)` and `handle(*args, **options)`, and a `help` string. Below this, the `airpollution.management.commands.populate_db module` is also shown with its class details.

Figure 3. Document Example

Project Software Tools

Our team used a variety of software products and everything we needed was able to be done with the free versions of the software.

Communication

We used Slack as a primary method of communicating and sharing information on a daily basis. For team meetings we used Zoom because it supports audio, video, and screen sharing.

Document Creation

We used Google Docs to create documents and Google Slides for PowerPoint-style presentations. These were used because they allowed all team members to collaborate on the same documents at the same time, and see modifications made by others in real-time. Documentation for the source code is embedded in the python modules and can be exported as a navigable html library using a module like MkDocs.

Date Coordination

For team meetings with and without the customer, we used Google calendar to keep everyone aware of the details.

Project Management

For managing the project, we used Trello. It's a lightweight way to create, assign, and track our tasks. We considered using Jira instead of Trello but found that Jira provided more functionality than we needed.

Mockups and Diagrams

We used Invision for basic wireframes and Astah for UML and sequence diagrams.

Software Source Control and CI/CD

We used Bitbucket for our code repository. It not only provides a repository but also CI/CD pipeline as a service.

Estimates and Planning

The project spans a time period from March 1 through May 13.

Efforts are planned as depicted in the timeline below.

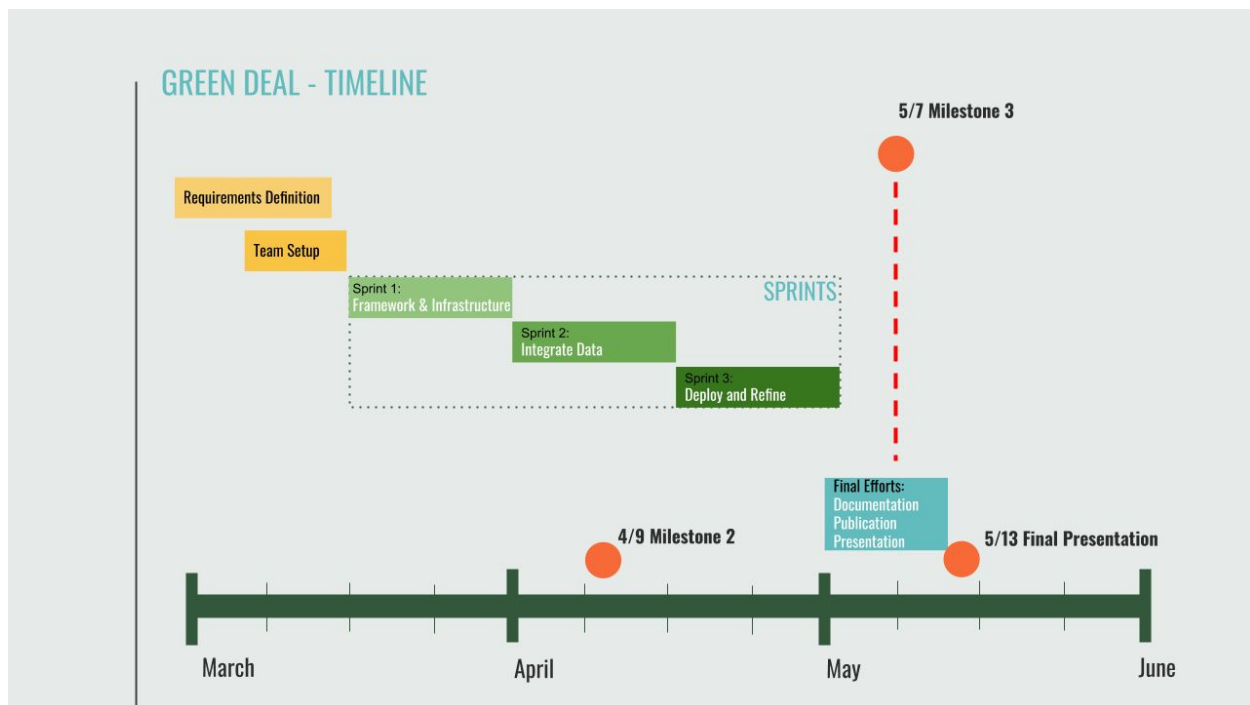


Figure 4. Timeline

Sprints

Three 2-week sprints are set up to handle 3 basic objectives of technical setup, integration of data and refining the deliverable and including stretch goals.

Final Cleanup

A final period of time is reserved to finalize documentation, prepare the publication and work on the final presentation.

Effort Estimation

The appendix includes a detailed listing of items to address before the project is complete ([Appendix D - Sprint Estimations](#)).

Effort times are defined by the number of days a person will need to work on this. Because this

is a class-project and we are not working normal work-days, a 'day' is whatever a person can do in a day along-side their usual professional workload. These days may not be sequential and may be spread over the time of the sprint. Efforts exclude project meeting/administrative time and time spent learning new technologies. This also excludes time spent in class and time preparing milestone documentation and presentations.

Team Approach

The team worked incredibly well together. Despite the fact that the customer (Dave Dowey) resides in Europe, the team had no difficulty interacting with him. Dave has been very generous with his time and joined the team on multiple Zoom meetings and also interacted via email frequently. The biggest limitation was available time for meetings, given the only available slots for all members were on the weekends.

As far as decisions go, the team discussed issues collectively and made all decisions in a collaborative fashion. All team members checked into Trello and Slack regularly and participated. The team loosely followed Agile as the primary development methodology. Initially, a Kanban approach was followed, but later switched to Scrum using sprints when coding commenced. Although the team planned the project in 2-week epics, weekly sprint targets were established to address current epics.

Development Process and Lessons Learned Reflection

The following section reviews the development process and the team's experiences throughout the process.

Requirements

Gathering a complete and clear list of requirements proved much more challenging than any team members expected. The customer presented us with a concept and the flexibility to implement the concept as we saw fit. Lacking a firm grasp of the landscape, the ultimate end

user and subject matter, we struggled to take advantage of the lack of limitations given to us. We found it difficult to define a set of requirements that described a useful product satisfying the broad vision. The effort consumed a great deal of time, motivation, morale and energy from the team. Later, we overcame this slow start as we became more familiar with the environment and the end vision became clear.

We set a key scoping element by limiting the application to being a prototype that hosts air pollution data only. The EU Green Deal includes targets for pollution, food, energy and transportation, but including all of these would have been impossible in our time frame. This simple decision helped to make the project a success.

Planning

Spending more time refining requirements put pressure on our planning timeline. Even with a list of requirements, we still lacked detailed understanding of what the data would look like, how to get it, the quality of the data and the storage requirements for the data. We also did not understand the fields we would use to link data sources together. This prevented us from preparing a detailed database design and later caused issues as the abundance of data we collected slowed data retrieval. We initially chose to keep all the data we found since we weren't sure which data would ultimately be necessary. Later, we were forced to curtail the data and adjust the collection process to focus only on the data we needed. This caused database and software changes. Fortunately, our high level MVC application design approach worked well and did not need redesign.

Spending more time on data design would have saved time and avoided rework as well as stress.

Technology Stack

We were fortunate to pick an initial technology stack that did not require significant changes along the way. We knew that the application would be web-based and that users would need to be authenticated. We also realized that the platform would be extended in the future. This made Django a simple framework choice especially since a critical mass of the team was already familiarly with Python. Using AWS to publish the application was also a clear choice as multiple team members had knowledge of the AWS landscape.

To support visualizations, we chose Bokeh. No team members had much experience with Bokeh but its simplicity and customizability made it a compelling choice. We never experienced any problems with this choice and all became comfortable using it.

Estimates

We broke our effort down into 3 2-weeks sprints that covered Framework & Infrastructure, Data Integration and Deployment. This high level outlook helped us break the work into targeted efforts with distinct milestones at the end of each sprint. Internally, we broke the 2-week sprints into weekly efforts which further helped us to manage the effort in smaller increments. Although some tasks slipped at times and new tasks were introduced, we were successfully able to manage meeting the high level timeline requirements. We did not focus on the hours spent, but rather the outputs required each week. By tracking the output of efforts rather than the time spent, we saved project management time and provided team members the flexibility to individually manage their own commitments. For this project, this output-based measurement approach made sense since we weren't paying people by the hour.

Summary

Overall, we all considered our effort a success. We could have done more planning to make the development effort easier. We were fortunate to have a team that worked well together which made up for a lack of detailed initial planning and the slow start during the requirements gathering phase.

Risks

Unavoidable risks

- The application will rely on the accuracy, completeness, and correctness of data from chosen sources. Data pulled from these sources is assumed to be correct. Errors in the source data will be propagated to this application. It is also expected that sensors may have problems and not report their data 100% of the time.
- We are reliant upon the various data resources and have no control of any outages, planned or unplanned, that they might experience.
- Changes in source data could break visualization and create unexpected behavior without warning.
- Because of the vast amounts of data, the speed of the response is of concern. This is mitigated by pre-processing the data when possible.
- Regional structures (NUTS regions) might change over time.
- It's possible that countries in the European Union change over time, such as on February 1, 2020 when the United Kingdom left. The ramifications of changes to the data is impossible to determine beforehand.

- Since the application is targeted towards EU policy makers, errors in the application or poor representation of the data could lead to inappropriate decisions and/or policies.
- The application is designed to summarize complex scientific data. This may lead to hidden correlations in data that may be valuable. Detailed analysis on trends observed in the application is necessary before taking action.

Intentional risks

- Currently, the European Union (EU) has 27 member states and 24 official languages. By only having the website in English and not providing a translation service, we are taking the risk of the site not being adopted by some people solely on the language barrier. The good news is English is the most widely used language in the EU government and is understood by a little over 50% of the population.
- We are taking a performance/availability risk by not rate-limiting any of our services. So this application is susceptible to a DDOS attack.

Appendix A - Requirements

Objective

Build a reporting tool that consolidates data from diverse sources and presents information to the user in a manner tailed to a users' persona. User personas are defined by an application admin that will predetermine the visualizations and pre-filtered values that are presented to all users who are assigned the same persona. Multiple personas can be created but a user can only be assigned to one persona.

The user request is summarized into the following two primary areas of focus:

- **Combined Data Sources:** Organize information from various sources into a single datastore. Allow new data sources to be added by extending the application.
- **Intuitive GUI:** Intuitive and friendly end-user design / cockpit-like format. Tailor visual presentation of data based on user's needs.

The ability to combine these two points into a cohesive user experience makes this application unique.

Target User

The target users of this web application are the Commissioners and Cabinet members from the 28 cabinets in the EU.

There is no public access to this website.

User stories

The requirements outlined in this document have been created to support the following list of user stories.

1. A Director-General, I would like to be able to filter a general overview of air quality for the past year

2. A Director-General, I would like to quickly be able to see which regions are on track to hit their Green Deal air quality goals
3. As a new commissioner to the EU, I would like to see an overview of the available data without any kind of setup
4. As a European & International Carbon Markets commissioner, I would like to see if carbon is increasing or decreasing in the EU
5. As a European & International Carbon Markets commissioner, I would like to see which regions are releasing the most carbon and which producing the least
6. As the Road Transport commissioner, I would like to see how much carbon monoxide and nitrogen dioxide is being produced each year and how the EU is tracking towards that goal
7. As a Clean Air commissioner, I would like to see which pollutants are and are not on track to hit the EU's Green Deal air quality attainment goals
8. As a regional commissioner, I would like to filter based on the countries I am currently interested in.
9. As an auditing commissioner, I would like to be able to generate a report on how each NUTS 3 region is tracking towards its air quality goals.
10. As an International Relations commissioner, I would like to view what air pollution levels are on the edge of the EU's territory to see if neighboring countries are creating pollution that is affecting the EU
11. As an admin, I would like to be able to create user persona/user roles.
12. As an admin, I would like to create users and associate them to a user persona/role.
13. As an admin, I would like to be able to view all the default dashboards and filter the charts based on the dimension filters available.
14. As an admin, I would like to build customized dashboards with multiple charts. Each such dashboard can be assigned to a user persona/user role. Each chart would have a pre-defined filtered dataset shown under that chart.

15. As an admin, I would like to add charts with pre-applied filters to customized dashboards intended to be created for each user persona.
16. As an admin, I would like to be able to see air pollution data from Copernicus (Comma-Separated Values [CSV] based data source), EEA (Resource Description Framework [RDF] based data source) and Eurostat (Representational State Transfer [REST] based data source).
17. As an application maintainer, I would like to be able to use the RDF based data ingestor module to point to a different source than EEA and ingest the data given that the schema of the new data being ingested is the same as specified in the documentation. The same goes for REST based data ingestor and CSV based data ingestor.

Requirements Summary

Front-End

The proposed application will be accessible via a web front-end. Upon visiting the page and logging in, a user will be presented with a default cockpit that includes an EU-wide overview of environmental data. As a proof-of-concept, the delivered application will restrict this data to 'air quality' topics. The cockpit's visualizations and arrangement will be tailored to the user's characteristics as defined by their user id. This data is presented in "tiles" of information where each tile is a type of visualization. Proposed visualization types include maps, charts and KPI's:

Maps

Map visualizations will show overlays of pollution information. The user will be able to select different time frames for which to show the overlays and also select different types of pollution.

Charts

Chart visualizations will present air-quality trends that can be filtered and clicked from drill-down information.

KPIs

KPI visualizations will display a limited list of key figures that include levels, values or percentages.

These ‘tiles’ include various features depending on the visualization type. Features include, but are not limited to:

- Drill down
- Change time periods displayed
- Add/Remove data dimensions of the content displayed

Using these features, the user will be able to create a display with data of their interest, yet in a curated fashion so that the user only sees reasonable combinations of data that are consistent across visualization and other user roles.

When each visualization is presented, data is pulled from the central data store to create an initial display of data. This data pulled will contain additional data that can be used to reflect subsequent user-driven filters placed on the visualization. The initial display of data may take a few seconds, but subsequent filtering should be perceived as instantaneous. The application will allow new visualizations to be programmatically added.

Two additional cockpits are proposed:

Regional Cockpit

The regional cockpit will present region-focused visualizations with data for a selected region or regions.

Trends Cockpit

The trends cockpit will present visualization focused on trends and target attainments.

Examples of the proposed visualizations and cockpits can be found under the “Software Design” section of this document.

Roles

The application will support an administrative user role. This role will be able to create new users and assign users to a user role. The admin can create user roles based on the interest of the user. Each user role can be configured by the admin to display selected visualization with selected filtered items to users of a role. Assignment to a role provides the user with an experience that is aligned with job-relevant objectives.

Back-End

In order to supply the web page with data, a backend is proposed which will accumulate relevant data from 3 primary sources:

1. Copernicus
2. EEA
3. Eurostat

The application will support the addition of new external data sources. This effort will require technical application support, but the effort to make these extensions will be minimized by the application design.

A constellation of AWS services will be used to update the central data store, process and clean the data, store the data and deliver the data to the web page. A more detailed overview of the landscape is available in the “Software Design” section of this document. The design of the application will allow extensibility for new external data sources. Extensibility is explained in detail in the Non-Functional Requirements section 2.5 below.

Technical

Python will be the language of choice with Javascript for front-end development. The delivered application will use Chrome v80 as the target browser.

Detailed Functional Requirements

The list of requirements below is designed to cover the application request in detail and elaborate on the summary above. Each main section of the requirements is tied to the application requests.

The requirements below are structured in a hierarchical manner that is arranged to match the general application architecture (see section: Application Architecture section of this document). This structure will also serve as a template for detailed application design.

The detailed requirements are divided into the following sections:

- 1.0 Functional Requirements
 - 1.1 Data
 - 1.2 Middle Layer
 - 1.3 Front-End
- 2.0 Non-Functional Requirements
- 3.0 Stretch Goals
- 4.0 Out of Scope
- 5.0 Other Considerations

Each requirement includes the following attributes:

Description: A short description of the requirement

Effort/Duration: The estimated amount of effort. These are not sequential time periods and will be converted to time during specification. The effort estimate includes detailed design, coding, testing, and documentation.

Acceptance Criteria: A description of the required output or the state after the requirement has been met.

Prerequisites: The efforts necessary before the requirement can be met.

1.0 DETAILED FUNCTIONAL REQUIREMENTS

1.0	<p>Description: A user will have access to a web-based platform that consolidates data from diverse external data sources and presents a series of summarized visualizations that are designed to meet the needs of the specific user. The user can update the data displayed in order to fine-tune the initial display into something that interests the user.</p> <p>Effort/Duration: N/A <covers length of project></p> <p>Acceptance Criteria: Acceptance of all sub-requirements</p> <p>Prerequisites: Requirements are complete and understood.</p>
-----	---

FUNCTIONAL REQUIREMENTS: DATA

Addresses customer requirement “Combine Data Sources” and “Data Dimensions”

1.1	<p>Description: The user will have access to data from a diverse set of external data sources. These data sources can be expanded to include additional new data sources.</p> <p>Effort/Duration: 9</p> <p>Acceptance Criteria: Multiple data sources are included in the application such that data from these external sources is extracted, transformed and loaded into a central database of the application. Provide the services necessary to extract defined raw data dimensions from defined sources. This includes Python programs that will access each data source via the externally facing interfaces provided by each source. These are interface programs that send a query request to the data source and collect the returned data. This data will be saved into the centralized application database in a separate step. Interfaces are able to collect the catalog of required data elements for each of the defined data sources. This can be measured by checking off each of the cataloged data elements and ensuring they can be accessed via one of the interfaces created.</p> <p>Prerequisites: EDA complete and all required data elements cataloged. This serves as the checklist to ensure all data can be accessed and measure the completion of this task.</p>
-----	---

Build External Data Interfaces

1.1.1	<p>Description: Data will be collected from external sources using interfaces. These interfaces will use a common structure so that interfaces to new data sources can be built making the application expandable. Interfaces will be built using Python. These interfaces will collect the specified data as defined in the data catalog specified in EDA section of this document. These interfaces will serve as API's</p>
-------	--

internal to the application and have a common structure so that structure of each call for data is the same for all data sources. Defining this common structure will be an iterative process defined as each data source is understood and the specific data needed to identify the target data is understood.

The “data api” will be a project-specific module that can be imported into other modules. A class will be available in this module for every data source. To access the API, a program will need to instantiate an instance of the class to access the data access methods for the data source to which the class is dedicated.

Each API should accommodate the following arguments:

- Datasource
- Data Dimensions:
 - **Time Dimensions:** A list of time dimensions that should be returned. Yearly, monthly, daily, hourly. With hours being the most granular unit. Multiple time values can be supplied in a list.
 - **Time Values:** A list of lists, where each list represents the time values corresponding to the ‘Time’ dimensions. The elements in this list will be the same as the ‘time dimensions’. Each list contains scalar values of the time periods to collect. An empty list will return all available times.
 - **Copernicus pollution unit:** A list of Pollution types: Ozone, CO2, PM 2.5, PM 10, NO2, SO2
 - **Region:** A list of NUTS-specified regional level or levels that should be returned. (Static region locations updated manually when new data is available provided by <https://ec.europa.eu/eurostat/>.)
 - **Include_children:** Boolean value that indicates if all NUTS available regions below the region/s listed in ‘Region’ should be returned. Defaults to False.
 - **Attainment data:** Boolean value that indicates whether or not attainment data should be included in returned values. (Pollution goal data updated manually when new data is available [provided by the EEA](#))

Data for each API shall be returned in a standard dictionary structure with the following keys:

- **Time**
 - Labels - labels for each time dimension
 - Values - the values for each time dimension

	<ul style="list-style-type: none"> - Region <ul style="list-style-type: none"> - Labels - labels for each region dimension - Values - the values for each region dimension - Kind <ul style="list-style-type: none"> - A single value that describes the kind of data (e.g. - pollution) - Elements <ul style="list-style-type: none"> - Labels - labels for each element (e.g. "Carbon Dioxide") - Values - the key value for each element (e.g. - "CO2") - Data <ul style="list-style-type: none"> - The data points - Shape <ul style="list-style-type: none"> - The array shape of the data (time, region, elements) <p>Effort/Duration: 4 Acceptance Criteria: An API definition that can be extended to numerous data sources. Prerequisites: A catalog of data sources described in the EDA.</p>
1.1.1.1	<p>Description: Build Python interfaces to collect the RDF-data stored in the EEA (European Environment Agency) data sources. This interface will follow the API specification outlined in the EDA.</p> <p>Effort/Duration: 1 Acceptance Criteria: An API exists that can deliver all the data required by the application as specified in the data catalog in the EDA section of this document related to the EEA data sources. Prerequisites:</p> <ul style="list-style-type: none"> - Understanding of RDF and SPARQL. - A completed API specification that the interface will implement as specified in section 1.2.1 of this document. - A catalog of data that should be collected from the data source as defined in the EDA section of this document.
1.1.1.2	<p>Description: Build Python interfaces to collect the REST-based data stored in the EuroStat data sources. This interface will follow the API specification outlined in 1.2.1.</p> <p>Effort/Duration: 1 Acceptance Criteria: An API exists that can deliver all the data required by the application as specified in the data catalog in the EDA section of this document related to the EuroStat data sources. Prerequisites:</p> <ul style="list-style-type: none"> - A completed API specification that the interface will implement as specified

	<p>in section 1.2.1 of this document.</p> <ul style="list-style-type: none"> - A catalog of data that should be collected from the data source as defined in the EDA section of this document.
1.1.1.3	<p>Description: Build Python interfaces to collect the CSV-based and binary data stored in the Copernicus data sources. This interface will follow the API specification outlined in 1.2.1.</p> <p>Effort/Duration: 1</p> <p>Acceptance Criteria: An API exists that can deliver all the data required by the application as specified in the data catalog in the EDA section of this document related to the Copernicus data sources.</p> <p>Prerequisites:</p> <ul style="list-style-type: none"> - A completed API specification that the interface will implement as specified in section 1.2.1 of this document. - A catalog of data that should be collected from the data source as defined in THE EDA section of this document.
Database	
1.1.3	<p>Description: A Centralized RDBMS database will store application data. This database will serve as the central data repository for the application described in this document. This database may consist of several tables that have not yet been defined. The definition of these tables will depend on the detailed application design and performance under real-World conditions. This database may not be fully normalized in an effort to place emphasis on the speed of access. This database will only include data that is collected through the interfaces described in section 1.2.2 of this document. No other data will exist in this store and no other sources will provide information to this database. The data in this database will only be accessible to the application described in this document and will not be directly available through an API.</p> <p>Effort/Duration: 3</p> <p>Acceptance Criteria: Should be able to store data from various sources</p> <p>Prerequisites: Should know RDBMS</p>
1.1.3.1	<p>Description: Data entities will be created based on the application ontology</p> <p>Effort/Duration:1</p> <p>Acceptance Criteria: Complete data can be represented in RDBMS</p> <p>Prerequisites: EDA complete</p>
1.1.3.2	<p>Description: ERM will be created based on defined entities</p> <p>Effort/Duration:1</p>

	<p>Acceptance Criteria: All relations between entities are captured in the model</p> <p>Prerequisites: Data entities defined. Metadata defined from data sources.</p>
1.1.3.3	<p>Description: RDBMS schemas will be built to reflect the data necessary in the application as defined in the EDA.</p> <p>Effort/Duration: 1</p> <p>Acceptance Criteria: Data from various sources can be put into schemas</p> <p>Prerequisites: Data sources should be analyzed</p>
Build Data Processing Logic	
1.1.4	<p>Description: An Extract/Transform/Load (ETL) series of services will be created.</p> <p>Effort/Duration: 9</p> <p>Acceptance Criteria: Data should be loaded to RDBMS</p> <p>Prerequisites: RDBMS and Data sources are available</p>
1.1.4.1	<p>Description: An AWS service will be used to extract data. This service will use the APIs created in section 1.2.1 which standardize access to the data sources used by this Application as described in the EDA section of this document. A scheduler will access the services established in this section to collect data on intervals specified by the update frequency of the source.</p> <p>Effort/Duration: 2</p> <p>Acceptance Criteria: AWS can extract all data necessary to populate the application central data store described in section 1.2.3.</p> <p>Prerequisites: All APIs are completed which are designed to access data from the application's external data sources.</p>
1.1.4.2	<p>Description: An AWS service will be configured to transform data (clean, calculate and extend data). Initially, no ML/DL transformations are foreseen. Shall these be added later, they would extend this requirement. Currently, the exact transformations to existing data are unknown until data extracts are complete and the resulting data is analyzed. These transformations may include data formats, data corrections, data scaling, standardization/normalization of data or possibly calculations based on combinations of data. This effort includes defining the transformations necessary as well as documenting the transformations.</p> <p>Effort/Duration: 4</p> <p>Acceptance Criteria: Services created can prepare all data so that it can be stored in the centralized application database and their use and processes are documented.</p> <p>Prerequisites: Data should be accessible as defined in section 1.2.4.1.</p>
1.1.4.3	<p>Description: AWS services will be configured to load data. This effort includes storing transformed data into the application database.</p> <p>Effort/Duration: 2</p>

	<p>Acceptance Criteria: Data should be loaded to the application database and available for access by the middle-layer APIs described in section 1.3.</p> <p>Prerequisites: RDBMS should be available for loading data as described in section 1.2.3.</p>
1.1.4.4	<p>Description: AWS services will be configured to schedule ETL processing. This includes defining the update interval for data elements and setting up the AWS service to execute API calls and subsequent data loads for data updated in the external sources.</p> <p>Effort/Duration: 3</p> <p>Acceptance Criteria: Data should be updated at a defined frequency from the external source.</p> <p>Prerequisites: Frequencies should be defined for various data sources that need to be updated.</p>

FUNCTIONAL REQUIREMENTS: MIDDLE-LAYER	
Enables customer requirement "Intuitive GUI"	
1.2	<p>Description: Use Django web-server framework will be configured to handle and present data. The webserver framework for this application will be python-based as the result of a Python focus requested by the customer.</p> <p>Effort/Duration: 7</p> <p>Acceptance Criteria: A basic framework has been put in place with a designated web address where site pages and respective logic can be stored. To complete this task, only a simple "Hello World"-like the response is necessary.</p> <p>Prerequisites: A web hosting service must be determined. We anticipate that AWS EC3 will be used.</p>
1.2.1	<p>Description: ORM Models will connect front and back ends. The web server logic will require a series of ORM definitions to hold data captured from the database which will ultimately be used to present data. We anticipate that an ORM model will be required for each distinct visualization defined (see Front-End section of this document for the visualizations proposed.)</p> <p>Effort/Duration: 3</p> <p>Acceptance Criteria: An ORM is defined for each visualization and mapping defined to each ORM's data source. Be able to query tables using ORM objects and allow access to the database data and schema using abstractions.</p> <p>Prerequisites: Database schema needs to be finalized before this step.</p>
1.2.2	<p>Description: A middle-layer API service will be built to support front-end UI design.</p>

	<p>The structure and definition of these API's should mirror the structure defined in section 1.2.1. In that section, access to each data source is defined. In this section, access to the internal application database is defined. These two structures should be the same with only minor deviation.</p> <p>Effort/Duration: 3</p> <p>Acceptance Criteria: All data used by the front-end application is available through a defined API. Each API parameter is documented and graceful error handling is implemented so that the application will not crash if called data does not exist.</p> <p>Prerequisites: ORM models are complete to serve as data objects for API.</p>
1.2.3	<p>Description: A publicly accessible endpoint/website will be provided. AWS EC2 service will be used to host the Django webserver.</p> <p>Effort/Duration: 1</p> <p>Acceptance Criteria: URL address is publicly accessible.</p> <p>Prerequisites: Need to have AWS account</p>
1.2.4	<p>Description: The application will support an application user role. A user will not have access to any administrative feature of the application.</p> <p>Effort/Duration: 1</p> <p>Acceptance Criteria: User role is available that allows the user to view pages in the application with no access to any administrative tasks (like assigning new users or creating roles.)</p> <p>Prerequisites: None</p>
1.2.5	<p>Description: The application will support an application administrator role.</p> <p>Effort/Duration: 1</p> <p>Acceptance Criteria: New user creation will allow the administrator role to be selected which allows the admin user to create new users and create new roles. User's setup as administrator will have an admin attribute assigned to their user id.</p> <p>Prerequisites: None</p>
1.2.6	<p>Description: The application will support user identity through login. Each user will be required to login before being presented with data. The user will not self-register for access and will instead need to request access which will be granted by an application administrator.</p> <p>Effort/Duration: 1</p> <p>Acceptance Criteria: Login screen is available. Upon login, user-specific meta-data is available which can be used by the front-end to determine how data is displayed.</p> <p>Prerequisites: None</p>
1.2.6.1	<p>Description: The application will support the addition of a new user by an admin user. Before using the application, a user will require a user ID. This ID is established by a centralized administrator user. The administrator will require a user-administration page that is only accessible to the administrator.</p>

	<p>Effort/Duration: 1 Acceptance Criteria: A Prerequisites:</p>
1.2.6.2	<p>Description: The application will support the creation and maintenance of user roles. An administrative user will have the ability to create new user roles which can later be assigned to users. The user's role will define what the user sees when logging in to the application.</p> <p>Each role will include the following list of maintainable characteristics:</p> <ul style="list-style-type: none"> - The number and arrangement of specifically-filtered visualizations on the home page cockpit. - Each visualization assigned to a role can be further refined with the visualizations default time dimension, data content selected and default region selection. - The location of each visualization can also be defined by the admin user. <p>Examples of roles are:</p> <ul style="list-style-type: none"> - General (the layman) - Transport - Economy - Top-Level (Commissioner) - Goal-Attainment - Regional (specific region) <p>Effort/Duration: 3 Acceptance Criteria: An administrative user has access to a User Management section where new users can be added and the region and default visualizations can be defined. All visualizations available in the application will be presented dynamically so that the new specific process is necessary to make a visualization available to the administrative user when assigning a role. Prerequisites: An admin role is available within the platform.</p>
1.2.6.3	<p>Description: Each user will be assigned a user role. When creating a new user, the administrative user will assign each user with a role. Only one role is required, but including multiple roles can be supported if possible (multiple role assignments is not a requirement but can optionally be included).</p> <p>Effort/Duration: 1 Acceptance Criteria: The new user process includes the ability to assign a role. Prerequisites: Admin user is available in the application.</p>

FUNCTIONAL REQUIREMENTS: FRONT-END	
Addresses customer requirement "Intuitive GUI"	
1.3	<p>Description: Air Quality Reporting Dashboard will be created for users</p> <p>Effort/Duration: N/A <covers length of project - non contiguous effort></p> <p>Acceptance Criteria: Customer acceptance of look and feel as well as data</p> <p>Prerequisites: Backend and middle layers functional with data defined and available. Required completed ORM model for front-end data handling.</p>
Visualizations	
1.3.1	<p>Description: Front-end visual elements will be created that include dimensions such as: time series, pollutant, NUTS 3 regions, and country regions. Additional statistical figures like population density levels may be included. The data presented in each visual element is determined by the user role.</p> <p>Effort/Duration: 10</p> <p>Acceptance Criteria: A completed set of visual element sketches that present visual representations of air quality in the EU.</p> <p>Prerequisites: The available data on air-quality is cataloged as described in the EDA section of this document.</p>
1.3.1.1	<p>Description: A map overlay visualization (see wireframe) will be created. The pollutant type can be overlaid onto a European map. Such a visualization should include the option to see the overlay for different time periods. Each map will only contain a single overlay but will distinguish the intensity of the pollutant. When first presented to the user, the configuration of the visualization is defined by the user role.</p> <p>Effort/Duration: 3</p> <p>Acceptance Criteria: Sketches that show the possible map overlay options which include the set of pollutants and possible pollutant sources which are in the supporting data set.</p> <p>Prerequisites: The data available for map overlays are cataloged.</p>
1.3.1.2	<p>Description: A hierarchy of drill down-capable chart visualizations (see wireframe) will be created including drill-down by pollutant-type and region. The drill-down may display an updated chart or the detailed underlying data depending on the level of available detail in the chart. Chart data can be downloaded by the user to a local CSV-type file format. When first presented to the user, the configuration of the visualization is defined by the user role.</p> <p>Effort/Duration: 3</p>

	<p>Acceptance Criteria: A set of sketches that show the possible charting options which cover the air-quality data supported by the application.</p> <p>Prerequisites: A catalog of air-quality data from all sources.</p>
1.3.1.3	<p>Description: A table/tabular visualization (see wireframe) will be created. Tabular visualization contains traditional row/column formats. Table data can be downloaded by the user to a local file in a CSV-like format. When first presented to the user, the configuration of the visualization is defined by the user role.</p> <p>Effort/Duration: 3</p> <p>Acceptance Criteria: A set of sketches that show the possible tables which will be available in the application.</p> <p>Prerequisites: A catalog of air-quality data from all sources.</p>
1.3.1.4	<p>Description: A KPI visualization (see wireframe) will be created. KPI visualizations contain a specific data point of a single dimension and will contain a single value or a pair of values. (e.g. - current Ozone measure, target attainment percentage). When first presented to the user, the configuration of the visualization is defined by the user role.</p> <p>Effort/Duration: 1</p> <p>Acceptance Criteria: A set of sketches that show the KPI visualizations supported by the application. As the list of possible displayed KPI's may belong, a list of KPIs that are displayed may be provided with a single example of the visual representation of the data.</p> <p>Prerequisites: A catalog of air-quality data from all sources.</p>
1.3.1.5	<p>Description: A filtering and navigation structure will be created giving the user the ability to navigate to new visualizations as well as to refine the data in the current visualizations.</p> <p>Effort/Duration: 3</p> <p>Acceptance Criteria: A navigation structure design is available that enables the user to reach the data available in the application with a minimal number of steps.</p> <p>Prerequisites: The set visualizations and the data which each visualization can display is available.</p>
Dashboards	
1.3.2	<p>Description: A default dashboard landing page will be created. This default page's visualizations and filtered settings will be defined based on the user's user role. Includes: EU-level Air quality trends based on year region and pollutant, Goal attainment, Map overlay. The default landing page will be configured to show the</p>

	<p>user a high-level overview of current EU-wide data. The default page provides the user with an expectation of the visualizations possible and guides the user into refining the page to explore additional data. When a user visits the dashboard, the visualizations presented and their content will be defined by the role assigned to the user.</p> <p>Effort/Duration: 4 Acceptance Criteria: A wireframe is available that shows the landing page which covers EU-wide information. Prerequisites: Visualizations are defined.</p>
1.3.3	<p>Description: A dashboard will be created that focuses on Trends. The trend visualizations will be defined by the user's user role. Users will be able to analyze trends (time series analysis) for region, pollutant and pollutant sources. Such a dashboard focuses on timeline trends for various pollutants and sources over a period of time for a selected regional level. When a user visits the dashboard, the visualizations presented and their content will be defined by the role assigned to the user.</p> <p>Effort/Duration: 3 Acceptance Criteria: A wireframe is available that shows a page of trend-like visualizations that can be refined to display various dimensions of data supported by the application. Prerequisites: Visualizations are defined.</p>
1.3.4	<p>Description: A dashboard page that focuses on Regional information will be created. The visualizations and content will be defined by the user's user role. Users will be able to analyze data per region. The Regional Dashboard will focus on various different visualizations that pertain to a chosen region. This view may contain any combination of charts, KPIs and maps only specific to a region. When a user visits the dashboard, the visualizations presented and their content will be defined by the role assigned to the user.</p> <p>Effort/Duration: 3 Acceptance Criteria: A wireframe is available that shows a page of visualizations covering a user-selected region. Prerequisites: Visualizations are defined.</p>
Authentication	
1.3.5	<p>Description: Users will access the application using a login id and password. Each user will require an id to access the application. This account will be tied to a user role which will drive the visualizations displayed to the user including the configuration of each visualization. Upon visiting the page, the user is presented with a login screen or the ability to follow a process for requesting access. Security will be</p>

	<p>limited to off-the-shelf security tools available. No enhanced or special security will be provided. No adherence to GDPR will be included. No cookies will be used. No application state will be saved locally or centrally meaning that any filters or visualization settings set during a session will be saved for a subsequent visit.</p> <p>Effort/Duration: 1</p> <p>Acceptance Criteria: Users can log in and a default region is selected after login as well as a selected dashboard.</p> <p>Prerequisites: None</p>
1.3.6	<p>Description: User ID request. The user will have the ability to follow a process for requesting a new user account. Requests are routed to the admin user of the account. When requesting access, the user must identify his/her role.</p> <p>Effort/Duration: 1</p> <p>Acceptance Criteria: The login page has a link to a 'new user request' option that allows the user to self-select a role. Completed requests are routed to the admin user.</p> <p>Prerequisites: None</p>

2.0 NON-FUNCTIONAL REQUIREMENTS	
2.1	<p>Description: Intuitive GUI. The application should be simple for users to understand and navigate and allow users to understand the data available as well as the data being presented.</p>
2.2	<p>Description: 4-9's of availability. 99.99%</p>
2.3	<p>Description: Data Refresh. Copernicus data will be refreshed once per day (at midnight). All other data sources will be updated manually when new data is available. This includes Attainment Data and updated NUTS 3 regions. Country regions will be inherited by the plotting library we use (e.g. Plotly or matplotlib basemap).</p>
2.4	<p>Description: Speed. When each visualization is created, data is pulled from the central data store. This data will contain the data necessary to display the visualization plus additional data that may be shown in the visualization after the user makes changes to the visualization filter. The initial display of data may take a few seconds, but subsequent filtering should be perceived as instantaneous.</p>
2.5	<p>Description: The application should be built so that it can be extended for additional data sources. Data will originate from 3 primary types of data stores: RDF based data source, REST-based data source and CSV based data source.</p>

	<p>When adding a new data source that includes one of these 3 data types, the following efforts will be necessary to create a new interface:</p> <ol style="list-style-type: none"> 1. For each time series in the new data source, a mapping of data fields and conversion of values will be necessary. This effort must ensure that newly interfaced time series data is supported by and understood by the application and that the values have the proper type supported by the application. 2. The regional dimension of data must be aligned with the application. The regions identified in the source data must be mapped to the existing regions supported by the application. When interfacing to the external data, the interface will map the source region to the respective region in the application's database. This will require an identification of the source's relevant fields and values and any values not aligned to the application will need to be mapped accordingly. 3. The addition of new dimensions will require a more extensive application change including changes to the database schema. Adding new dimensions is not part of the application's extensibility.
2.6	<p>Description: Mobile-Friendly. The application web page should be mobile friendly and designed so that pages render in a usable manner on smaller mobile devices.</p>
2.7	<p>Description: Target user. The target users of this web application are the Commissioners and Cabinet members from the 28 cabinets in the EU. They will be given credentials from admin to access the website based on their roles. There is no public access to this website.</p>
<p>EDA</p>	
2.8	<p>Description: Exploratory Data Analysis (EDA). This effort represents the work of learning the data that is available in the data sources in more detail. This includes not only what the data represents, but also the names of the fields in the data. Other information includes the size of the data sources, how frequently they are updated, how reliable the data is and if it contains errors, the key fields in data stores, how data is linked together with other data sources and the latency associated with accessing individual or groups of records. Data will not be collected 'just-in-case' since the cost of including unused data is expected to drive increased costs and access times.</p> <p>Effort/Duration: 3</p> <p>Acceptance Criteria: Data sources and elements cataloged and access points to data sources identified. A listing of the data sources with the following items:</p> <ul style="list-style-type: none"> - Name of data source - How to access the data source - How long records take to access

	<ul style="list-style-type: none"> - What the data represents - The data types of data elements - An estimate of the accuracy of the data - The frequency which the data is updated - A mapping of record names and descriptions - Any metadata associated with datasets
<p>2.8.1</p>	<p>Description: The data sources will be restricted to the following:</p> <ul style="list-style-type: none"> - EEA - EuroStat - Copernicus <p>No other datasets will be explored or included in the solution. It is possible that not all data from each of the sources will be included in the application, but rather, a subset that represents air-quality related information. It is possible that data such as population density levels may also be included but only to the extent that it can be related to air quality.</p> <p>Effort/Duration: Part of EDA which has 3 points of effort.</p> <p>Acceptance Criteria: Sources of data are cataloged as described in the EDA section of this document.</p> <p>Prerequisites: Project Objective Defined. An understanding of what the project should deliver is necessary before the relevant data can be selected.</p>
<p>2.8.2</p>	<p>Description: Restrict data elements to air-quality topics. The application will focus on air-quality information only. No other environmental aspects will be covered. This includes the pollutants in the air as well as the sources. Define and document data elements required from each data source. Understand the air-quality relevant data tables and fields that will be useful in application. The following is a list of pollutants that will be included in the data:</p> <p style="margin-left: 40px;"> H4 - Methane CH4_CO2E - Methane (CO2 equivalent) CO2 - Carbon dioxide GHG - Greenhouse gases (CO2, N2O in CO2 equivalent, CH4 in CO2 equivalent, HFC in CO2 equivalent, PFC in CO2 equivalent, SF6 in CO2 equivalent, NF3 in CO2 equivalent) HFC_CO2E - Hydrofluorocarbons (CO2 equivalent) HFC_PFC_NSP_CO2E - Hydrofluorocarbons and perfluorocarbons - not specified mix (CO2 equivalent) N2O - Nitrous oxide N2O_CO2E - Nitrous oxide (CO2 equivalent) NF3_CO2E - Nitrogen trifluoride (CO2 equivalent) PFC_CO2E - Perfluorocarbons (CO2 equivalent) SF6_CO2E - Sulphur hexafluoride (CO2 equivalent) </p> <hr style="width: 20%; margin-left: 40px;"/> <p>Air-quality data will not include the source of pollutants. The source list is 170 elements</p>

	<p>long which contributes to a complexity not intended for the high-level audience. Forming visualizations to properly interpret such a long list is beyond the time-scope for this project.</p> <p>The data collected will include map overlay data for each of the pollutants as well as absolute values of the pollutant data. These data points will be available for all regions that are published. These regions are defined by NUTS 3 regions in the EU. There are several levels of regional representations that have changed over the years. The complexity of tracking these changes and displaying values by region would be complex to deliver visually. Instead, the only NUTS 3 regions that will be represented are the latest regions along with country regions.</p> <p>Effort/Duration: 3 Acceptance Criteria: Catalog of data sources, tables and elements is available including data descriptions Prerequisites: Data sources defined</p>
--	---

3.0 STRETCH GOALS	
3.0	Description: Stretch Goals
3.1	<p>Description: (Stretch goals) Searching keywords should update the dashboard Effort/Duration: 5 Prerequisites: Complete dashboard pipeline should work before this. Acceptance Criteria: User should be able to search keywords and dashboard should update accordingly.</p>
3.2	<p>Description: (Stretch goals) Forecasting data. Using ML/DL to forecast or project values. This goal carries a risk of setting expectations that are not validated by an expert. Considering the topic this may have political ramifications or even societal implications if the forecasts are not accurate and the data is relied upon. Effort/Duration: 5 Prerequisites: Should be done with Data part of this project. Acceptance Criteria: Projected values should follow the ML/DL</p>
3.3	<p>Description: (Stretch goals) Authenticated access to application / User Customization. In the event this goal is pursued, an application security strategy will be required. Currently, security is out of scope. Effort/Duration: 5</p>

	<p>Prerequisites: Default dashboard should be available.</p> <p>Acceptance Criteria: Users should be able to login and should have a personalized default page.</p>
--	---

4.0 OUT OF SCOPE	
4.0	Description: Out Of Scope
4.1	<p>Topic: Chatbot</p> <p>Description: A chatbot feature will not be included in the scope of the application. Including a chatbot could be included in a second version of the product, but for the initial design, it was determined that a chatbot would lead to a poor user experience since only limited free-form user requests would likely be met with a desired response. Instead, the focus is on ensuring that the user will get an expectation of the output at the moment of landing on the home page and guided to a refined solution through filters and links.</p>
4.2	<p>Topic: User-defined customizations</p> <p>Description: The user will not have the ability to customize any default visualization settings. The user can update defaulted elements, but any filtered updates selected by the user are not saved. The environment presented to the user will be defined by the role set under the user's ID. Each user of the same role will see the same set of visualizations. No cookies will be used to capture the user state.</p>
4.3	<p>Topic: Multi-language support</p> <p>Description: In this project scope we decided to provide English language support only. Multi-language support could be a future enhancement to this project.</p>
4.4	<p>Topic: Security</p> <p>Description: No data security issues will be considered. The platform contains publicly available information and no personal user data. User authentication is being done but no personal information is saved.</p>
4.5	<p>Topic: Stand-Alone Application</p> <p>Description: The application will be web-enabled with no stand-alone application.</p>
4.6	Topic: Air Pollutant Sources

	<p>Description: The source of air pollutants will not be covered in the application. The list of pollutants is 170 elements long which will make the intuitiveness of the application difficult to realize in the timeframe allocated for the project.</p>
4.7	<p>Topic: API Description: No API will be provided by the application which allows direct access to the data shown in the web interface. The only way to access the data is through the web presentation and any respective download options supplied by certain views.</p>
4.8	<p>Topic: GDPR Compliance Description: No consideration for GDPR compliance will be included. To the extent that any default user authentication service which is selected provides GDPR by default, it will be available, but no review or audit of the application will be done to guarantee adherence to GDPR guidelines.</p>

OTHER	
5.1	<p>Topic: Site Availability Description: The site will be available for review during a limited time and terminated after the semester is over. At this point, the publicly hosted presence will be taken down. At the customer's request, the application can be set up on a customer supplied AWS environment provided that the credentials for the environment are provided at the onset of the project.</p>
5.2	<p>Topic: Documentation Description: Application code and service configurations will be documented including any setup scripts. All documentation will be made publicly available and will be at the published destination (BitBucket) for a period of 1 year after the project is completed.</p>

Prioritization

Must:

- *Have visualizations for air quality data*
- *Support time-series trends (down to daily) for air quality data*
- *Display regional data at a country level*
- *Compare actual data versus attainment goals*
- *Mobile-friendly web application*

- *Be built with Python*
- *Support user logins*
- *An administrator will set up users*
- *Have unique views per user type*

Should:

- *Display data at a NUTS 3 regional level*
- *Indicate outliers by region by pollutant levels*

Could:

- *Visualize a time series playback of pollution levels across Europe*
- *Store state of filters when users login the next time they visit the application*

Won't:

- *Support languages beyond English*
- *Implement GDPR compliance*
- *Create an API*
- *Support a chatbot*
- *Support user-created customizations*
- *Be available to the general public*

Appendix B - UI Wireframes

Green Deal Dashboard

Home

Trends

Regional Report

Goal Tracking

Air Quality vs Other Data

Air Quality Trends

● 2019 ● 2020 Last 4 months

Pollutant	Change
Ozone	-5%
NO2	-4%
SO2	-9%
CO2	-11%
PM10	+1%
P2.5	+2%

Recent Pollution Spikes

Today

Location	Pollutant	Increase
Rieti, IT	NO2	+323%
Roma, IT	NO2	+309%
Latina, IT	NO2	+302%
Dobrich, BG	CO	+283%
Shumen, BG	CO	+282%

Air Quality By Region

Last 4 months

Actual vs 2020 Goal

Full Report

Year to Date Air Pollution

2%

Above Air Quality Goals

Ozone	-7%
NO2	-7%
SO2	-4%
CO	-7%
PM10	+1%
PM2.5	+2%

Page 57

Green Deal Dashboard

Home

Trends

Regional Report

Goal Tracking

Air Quality vs Other Data

Air Quality Trends

All Regions

Weekly Data

Compare to prior year

More

Nov 1, 19 - Feb 1, 20

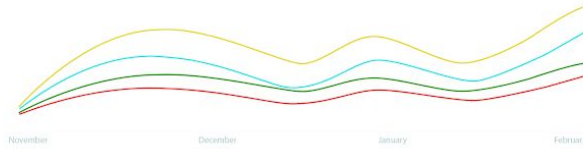
Air Quality Index -2%

2019 2020



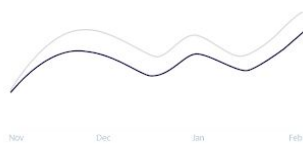
Pollutant Comparison

NO2 Ozone PM2.5 SO2



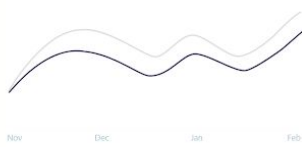
Ozone -5%

2019 2020



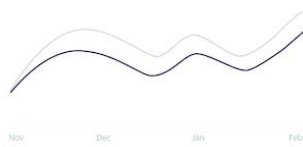
NO2 +4%

2019 2020



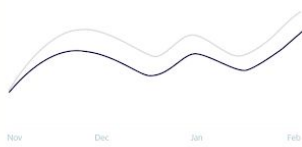
SO2 -9%

2019 2020



CO -11%

2019 2020



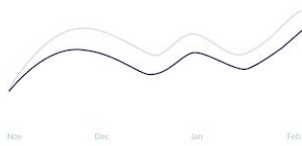
PM10 +1%

2019 2020



PM2.5 +2%

2019 2020



Air Quality Time Lapse



Play Time Lapse

Trend Summary

Export

Pollutant	Level	Prior Yr Level	Change
Ozone	62	65	-8%
NO2	101	97	+4%
SO2	89	97	-9%
CO	4876	5479	-11%
PM10	32	32	+1%

Green Deal Dashboard

- Home
- Trends
- Regional Report
- Goal Tracking
- Air Quality vs Other Data

Air Quality Trends

All Regions

All Pollutants

Nov 1, 19 - Feb 1, 20

Air Quality Trends



Recent Pollution Spikes

Location	Pollutant	Increase
Rieti, IT	NO2	+323%
Roma, IT	NO2	+309%
Latina, IT	NO2	+302%
Dobrich, BG	CO	+283%
Loren Ipsum, XX	NO2	+273%
Loren Ipsum, XX	NO2	+269%
Loren Ipsum, XX	NO2	+262%
Loren Ipsum, XX	CO	+253%
Loren Ipsum, XX	CO	+252%

Top Pollution Reductions

Location	Pollutant	Increase
Rieti, IT	SO2	-211%
Roma, IT	SO2	-200%
Latina, IT	PM2.5	-199%
Dobrich, BG	PM10	-198%
Loren Ipsum, XX	CO	-197%
Loren Ipsum, XX	CO	-196%
Loren Ipsum, XX	SO2	-180%
Loren Ipsum, XX	CO	-172%
Loren Ipsum, XX	Ozone	-170%

Green Deal Dashboard

- Home
- Trends
- Regional Report
- Goal Tracking
- Air Quality vs Other Data

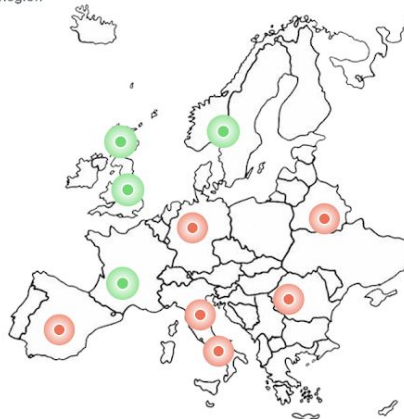
Air Quality Trends

All Regions ▾

Actual Air Quality vs Attainment Goals

Pollutant	Avg Attainment	2017 Actuals	2018 Actuals	2019 Actuals
Ozone	5800	5420 -7%	5420 -7%	5420 -7%
NO2	40	42 +5%	42 +5%	42 +5%
SO2	3	3 +0%	3 +0%	3 +0%
CO	0	0 +0%	0 +0%	0 +0%
PM10	50	51 +2%	51 +2%	51 +2%
PM2.5	25	25 +8%	25 +8%	25 +8%
Average Performance		+1.3%	+1.3%	+1.3%

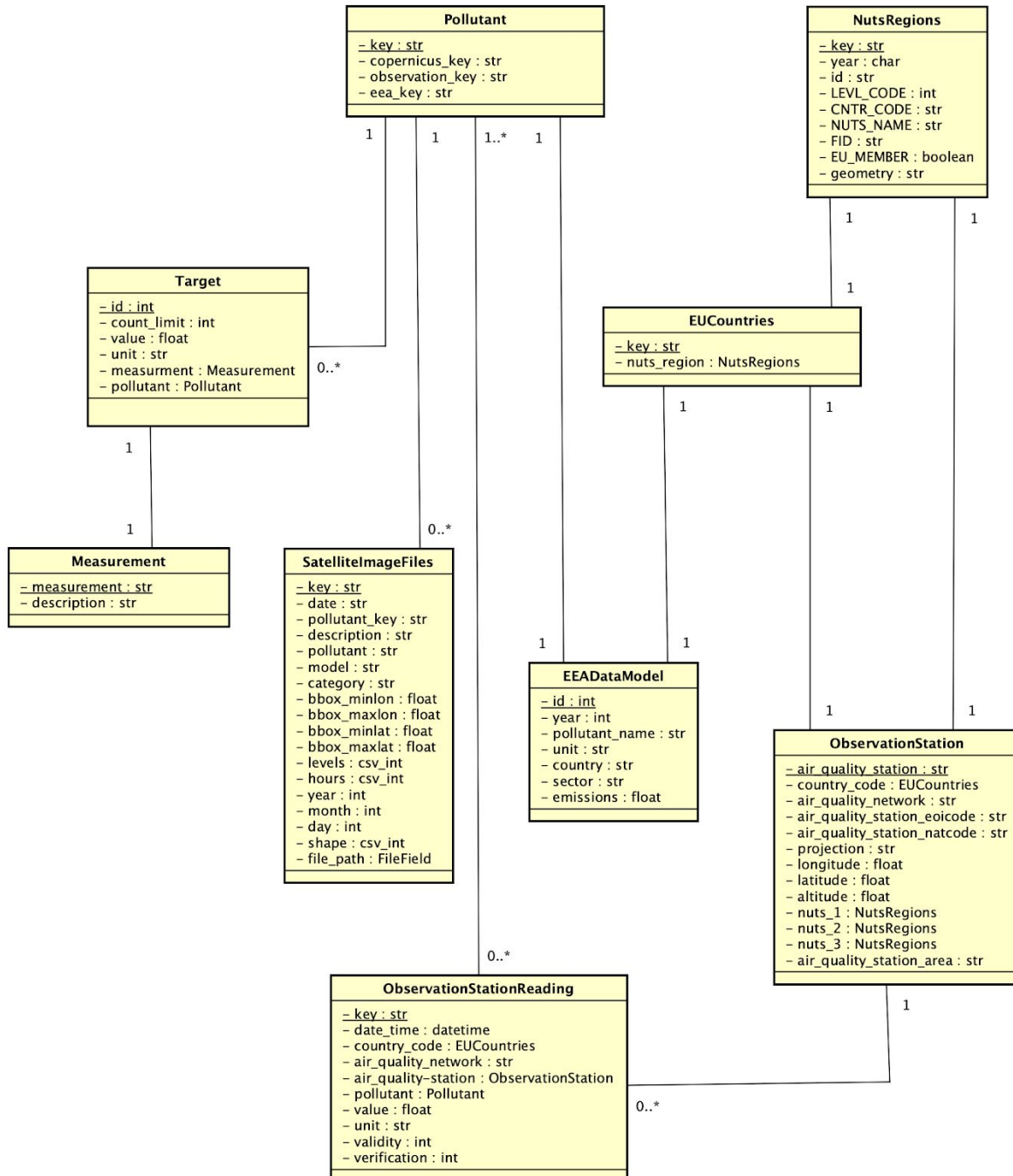
Progress Towards Attainment Goals by Region



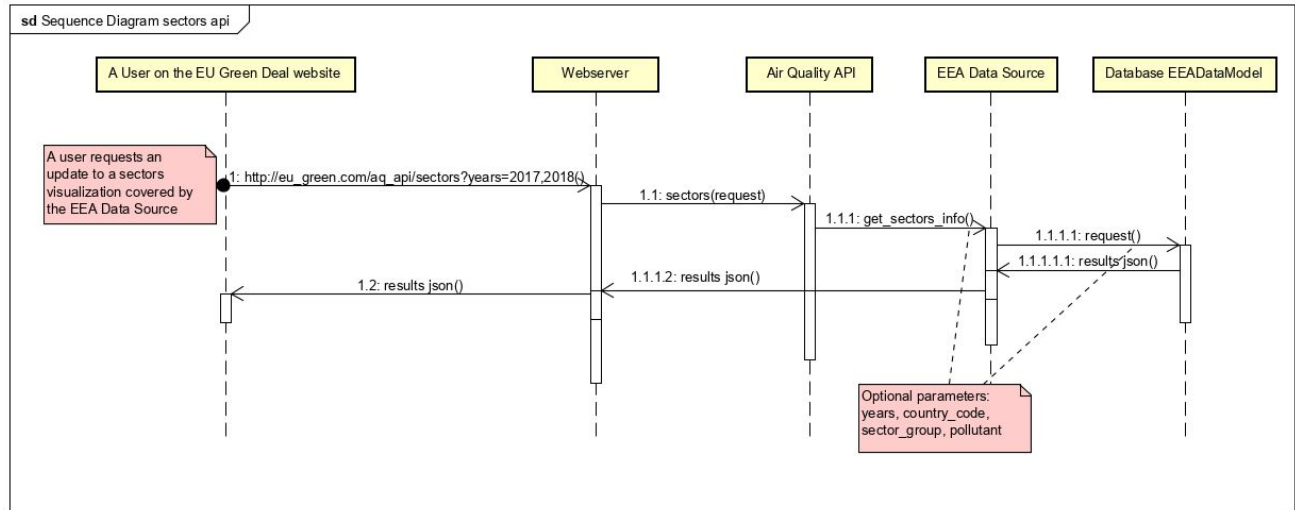
Compliance by Region

Location	Pollutant	2017 Actuals	2018 Actuals	2019 Actuals
London, UK	SO2	5420 -7%	5420 -7%	5420 -7%
Darlington, UK	SO2	42 +5%	42 +5%	42 +5%
Loren Ipsum, XX	PM2.5	3 +0%	3 +0%	3 +0%
Loren Ipsum, XX	PM10	0 +0%	0 +0%	0 +0%
Loren Ipsum, XX	CO	51 +2%	51 +2%	51 +2%
Loren Ipsum, XX	Ozone	25 +8%	25 +8%	25 +8%

Appendix C - ERM



Appendix D - Sequence Diagram



Sequence diagram for when a visualization gets updated.

Appendix E - Sprint Estimations

Sprint 1 - Framework and Infrastructure (2-weeks)		
Description	Est. Effort (person-days)	Actual Effort (person-days)
Setup Django Framework	4	2
Implement Design Template for Site	3	3
Setup Git process	1	1
Enable User Authentication	1	1
Design Admin Process and Pages	4	3
Design Persona Process and Pages	6	5
Enable Docker-based Postgres Database	2	1
Design Extensible Model for Data	3	2
Make Web design mobile friendly	4	5
Create Model and Data Load for Regions	5	5
Create Model and Data Load for EEA Annual Pollution	6	6
Create Model and Data Load for Ground Observation Stations	7	10
Create Model and Data Load for Satellite Images	4	5
Create Model and Data Load for Pollution Targets	3	1
TOTAL	53	50
Sprint 2 - Integrate Data (2-weeks)		
Description	Est. Effort	Actual Effort

	(person-days)	(person-days)
Document API. The API will sit between the front end and the back end data. The API will aggregate data from various data sources hosted in the application.	4	2
Provide API data access: Regions	4	2.5
Provide API data access: Observation Stations	3	5
Provide API data access: Satellite Images	1	3
Provide API data access: Pollution Targets	2	1
API Function: Annual Pollution	2	.5
API Function: Maps	1	.5
API Function: Daily Pollution Values	3	.5
API Function: Annual Pollution Values	2	.5
API Function: Satellite Images	1	.5
Design extensibility into visualizations	5	0
Landing Page	2	3
Add DataSource for EEA Population Data by Region	3	4
Enhance Mobile experience	2	3
Extend Map Visualization Filters (pollutant/country)	3	3
Document Data Model	1	1
Add Line Chart visualization	2	1
Use Line Chart to plot EEA Year Pollution data	2	3
Add map visualizations to home page and	2	2

dashboards		
TOTAL	45	36
Sprint 3 - Deploy - Refine (2-weeks)		
Description	Est. Effort (person-days)	Actual Effort (person-days)
Clean and Document Data Loading and Model Code	5	3
Clean and Document API Code	5	2
Clean and Document Visualization Management	5	1
Clean and Document Views Code	5	1
Refine Visualizations	15	20
Implement Python documentation	10	10
Deploy to AWS	3	5
Adjust Application for Flexible deployment locations	2	3
TOTAL	50	45
Final Effort - Documentation / Publication / Presentation (2-weeks)		
Description	Est. Effort (person-days)	Actual Effort (person-days)
Finalize End User and Developer Documentation	15	8
Complete Paper for publication	20	2
Prepare Final Presentation	20	10
TOTAL	50	20

Appendix F - API Documentation

Summary

The Air Quality API exposes all the information required to successfully visualize air quality data associated with the EU Green Deal.

[/aq_api/daily](#)

Provides pollutant levels in units of micro g/m3 by region, pollutant, and date.

Note: This API method works for dates after July 15, 2015

[/aq_api/annual](#)

Provides pollutant levels in units of micro g/m3 by pollutant, country, and year.

Note: This API method only provides annual data from [START DATE] through [END DATE]

[/aq_api/targets](#)

Provides a list of target emission levels by region, pollutant type, and year.

[/aq_api/sectors](#)

Provides pollutant levels in Gg (1000 tonnes) by sector, pollutant, country, and year.

Note: This API method works from dates 1990 through 2017

[/aq_api/region_info](#)

Provides a list of NUTS regions and associated information.

[/aq_api/region_boundaries](#)

Provides a list bounding shapes for NUTS regions.

All responses are formatted in JSON.

Responses

- **OK** indicating the API request was successful.
- **INVALID_REQUEST** indicating the API request was malformed
- **OUT_OF_BOUNDS_TIME** indicating the API dates are outside of the supported time range for that request
- **REQUEST_DENIED** indicating the API did not complete the request.
- **UNKNOWN_ERROR** indicating an unknown error.

Method details

/aq_api/daily

Provides pollutant levels in units of micro g/m3 by region, pollutant, and date.

Example request:

http://localhost:8000/aq_api/daily?version=v1®ions=de,fr,be&pollutants=o3,co&start-date=2020-03-19&end-date=2020-03-19

Request parameters:

parameter	type	description
version	String	Default=None. API version (e.g - 'v1'). If blank, default to the current version.
pollutants	[String]	A list of pollutant-types to be returned. Options include: <ul style="list-style-type: none"> • o3 Ozone • co: Carbon monoxide • no2 Nitrous dioxide • so2 Sulfur dioxide • pm25 Particulate matter (droplets that are 2.5 microns or less)

		<ul style="list-style-type: none"> • pm10 Particulate matter 10 (droplets that are 10 microns or less) • pans Peroxyacyl nitrates • nmvoc Non-methane volatile organic compounds • no Nitrous oxide • nh3 Gaseous ammonia
regions	[String]	<p>Default=None. This can be any NUTS code for NUTS0, NUTS1, NUTS2 or NUTS3. If blank, all regions are returned.</p> <p>Options include</p> <p>NUTS 1 (countries):</p> <ul style="list-style-type: none"> • at Austria • cz Czech Republic • de Germany • dk Denmark • ee Estonia • fi Finland • fr France • el Greece • hu Hungary • hr Croatia • ie Ireland • it Italy • lt Lithuania • lu Luxembourg • lv Latvia • mt Malta • nl Netherlands • pl Poland • pt Portugal • ro Romania • es Spain • se Sweden • si Slovenia • sk Slovakia • al Albania • me Montenegro • mk North Macedonia • rs Serbia • tr Turkey

		<ul style="list-style-type: none"> • ch Switzerland • is Iceland • li Liechtenstein • no Norway • uk United Kingdom <p>NUTS 2 examples: (Full list here)</p> <ul style="list-style-type: none"> • at-11 Burgenland • de-94 Weser-Ems <p>NUTS 3 examples: (Full list here)</p> <ul style="list-style-type: none"> • at-111 Mittelburgenland • be-341 Arr. Arlon
start-date	String	The start date of the range to be returned: YYYY-MM-DD (eg 2020-01-01)
end-date	String	The end date of the range to be returned (inclusive): YYYY-MM-DD (eg 2020-01-01)

Response values:

parameter	type	description
region	String	Regions codes will match the region codes from the request.
pollutant	String	Pollutant codes will match the pollutant codes from the request.
date	String	The dates of the corresponding air quality values.
avg-level	Float	The average pollutant level in units of micro g/m3.
ytd-level	Float	The average pollutant level in units of micro g/m3 up until this date

```
{
  "de" : {
    "2020-03-19" : {
```

```
    "o3" : {
      "day-avg-level": 66.2,
      "ytd-avg-level": 67.4
    },
    "co" : {
      "day-avg-level": 5000.3,
      "ytd-avg-level": 67.6
    }
  },
  "2020-03-20" : {
    "o3" : {
      "day-avg-level": 69.2,
      "ytd-avg-level": 67.4
    },
    "co" : {
      "day-avg-level": 5001.3,
      "ytd-avg-level": 67.6
    }
  }
}
```

/aq_api/annual

Provides pollutant levels in units of micro g/m3 by pollutant, country, and year.

Example request:

For all pollutants, all countries, all years:

http://localhost:8000/aq_api/annual?version=v1

For a set of countries, pollutants and/or years:

http://localhost:8000/aq_api/annual?version=v1&countries=de,fr,be&years=2016,2017,2018&pollutants=o3,co

parameter	type	description
version	String	Default=None. API version (e.g - 'v1'). If blank, default to the current version.
countries (optional)	[String]	<p>Default=None. If blank, all countries are returned.</p> <p>A list of nuts regions to be returned. Options include:</p> <ul style="list-style-type: none"> • at Austria • cz Czech Republic • de Germany • dk Denmark • ee Estonia • fi Finland • fr France • el Greece • hu Hungary • hr Croatia • ie Ireland • it Italy • lt Lithuania • lu Luxembourg • lv Latvia • mt Malta • nl Netherlands • pl Poland • pt Portugal • ro Romania • es Spain • se Sweden • si Slovenia • sk Slovakia

		<ul style="list-style-type: none"> • al Albania • me Montenegro • mk North Macedonia • rs Serbia • tr Turkey • ch Switzerland • is Iceland • li Liechtenstein • no Norway • uk United Kingdom
years (optional)	[Int]	Default=None. 4-digit year/s for which the target value is being requested. If blank, all available years are returned.
pollutants (optional)	[String]	A list of pollutant-types to be returned. Options include: <ul style="list-style-type: none"> • o3 Ozone • co: Carbon monoxide • no2 Nitrous dioxide • so2 Sulfur dioxide • pm25 Particulate matter (droplets that are 2.5 microns or less) • pm10 Particulate matter 10 (droplets that are 10 microns or less) • pans Peroxyacyl nitrates • nmvoc Non-methane volatile organic compounds • no Nitrous oxide • nh3 Gaseous ammonia

Response values:

key	type	description
country	String	Country code will match the country codes from the request.
year	Int	4-digit value of the year of the target
pollutant	String	Pollutant codes will match the pollutant codes from the request.


```
{
  "de": {
    "2011": {
      "co": 50000,
      "no2": 50000,
      "o3": 50000,
      "pm10": 50000,
      "pm25": 50000,
      "so2": 50000
    },
    "2012": {
      "c6h6": 50000,
      "co": 50000,
      "no2": 50000,
      "o3": 50000,
      "pm10": 50000,
      "pm25": 50000,
      "so2": 50000
    }
  }
}
```

/aq_api/targets

Provides a list of target emission levels by region, pollutant type, and year.

Example request:

For all pollutants, all regions, all years:

http://localhost:8000/aq_api/targets?version=v1

For a set of regions, pollutants and/or years:

http://localhost:8000/aq_api/targets?version=v1®ions=de,fr,be&years=2016,2017,2018&pollutants=o3,co

parameter	type	description
version	String	Default=None. API version (e.g - 'v1'). If blank, default to the current version.
regions (optional)	[String]	Default=None. This can be any NUTS code for NUTS0, NUTS1, NUTS2 or NUTS3. If blank, all regions are returned. Options include NUTS 1 (countries): <ul style="list-style-type: none"> • at Austria • cz Czech Republic • de Germany • dk Denmark • ee Estonia • fi Finland • fr France • el Greece • hu Hungary • hr Croatia • ie Ireland • it Italy • lt Lithuania • lu Luxembourg • lv Latvia • mt Malta • nl Netherlands • pl Poland • pt Portugal

		<ul style="list-style-type: none"> • ro Romania • es Spain • se Sweden • si Slovenia • sk Slovakia • al Albania • me Montenegro • mk North Macedonia • rs Serbia • tr Turkey • ch Switzerland • is Iceland • li Liechtenstein • no Norway • uk United Kingdom <p>NUTS 2 examples: (Full list here)</p> <ul style="list-style-type: none"> • at-11 Burgenland • de-94 Weser-Ems <p>NUTS 3 examples: (Full list here)</p> <ul style="list-style-type: none"> • at-111 Mittelburgenland • be-341 Arr. Arlon
years (optional)	[Int]	<p>Default=None.</p> <p>4-digit year/s for which the target value is being requested. If blank, all available years are returned.</p>
pollutants (optional)	[String]	<p>Default = None.</p> <p>If None, all available pollutant values are returned.</p> <p>Any number of the following pollutants can be included in the list. The values are not case-sensitive.</p> <p>A list of pollutant-types to be returned. Options include:</p> <ul style="list-style-type: none"> • c6h6 Benzene • co: Carbon monoxide • no2 Nitrous dioxide • o3 Ozone • pm10 Particulate matter 10 (droplets that are 10 microns or less)

		<ul style="list-style-type: none"> • pm25 Particulate matter (droplets that are 2.5 microns or less) • so2 Sulfur dioxide <p>(This list represents the available pollutants in the published targets)</p>
--	--	---

Response values:

key	type	description
region	String	Region code will match the region codes from the request.
year	Int	4-digit value of the year of the target
pollutant	String	Pollutant codes will match the pollutant codes from the request.

```
{'DE':
  {2019:
    {'PM25':
      {'calendar_year': {'value': 25.0, 'count_limit': 25.0, 'unit': 'ug/m3'}},
      'PM10':
        {'day': {'value': 50.0, 'count_limit': 50.0, 'unit': 'ug/m3'},
         'calendar_year': {'value': 40.0, 'count_limit': 40.0, 'unit': 'ug/m3'}},
      'O3':
        {'max_8hour_mean': {'value': 120.0, 'count_limit': 120.0, 'unit': 'ug/m3'}},
      'NO2':
        {'hour': {'value': 200.0, 'count_limit': 200.0, 'unit': 'ug/m3'},
         'calendar_year': {'value': 40.0, 'count_limit': 40.0, 'unit': 'ug/m3'}},
      'CO': {},
      'SO2': {},
      'PANS': {},
      'NMVOC': {},
      'NO': {},
      'NH3': {},
      'BIRCHPOLLEN': {},
      'OLIVEPOLLEN': {}},
  }
```

```
'GRASSPOLLEN': {},
'RAGWEEDPOLLEN': {},
'PB': {},
'NI': {},
'CR': {},
'CD': {},
'CU': {},
'AS': {},
'NOX': {},
'SOX': {}},
2020: { ...
```

/aq_api/sectors

Provides emissions data by sector, pollutant, country, and year.

Note: This API works for dates 1990 through 2017

Example request:

For all pollutants, all countries, all sectors, and all years:

http://localhost:8000/aq_api/sectors?version=v1

For a set of pollutants, countries, sectors, and/or years:

http://localhost:8000/aq_api/sectors?version=v1&countries=de,fr,be&years=2016,2017,2018&pollutants=o3,co§ors=oil-gas-production

parameter	type	description
-----------	------	-------------

version	String	Default=None. API version (e.g - 'v1'). If blank, default to the current version.
countries (optional)	[String]	<p>Default=None. If blank, all countries are returned.</p> <p>A list of countries to be returned. Options include:</p> <ul style="list-style-type: none"> • at Austria • cz Czech Republic • de Germany • dk Denmark • ee Estonia • fi Finland • fr France • el Greece • hu Hungary • hr Croatia • ie Ireland • it Italy • lt Lithuania • lu Luxembourg • lv Latvia • mt Malta • nl Netherlands • pl Poland • pt Portugal • ro Romania • es Spain • se Sweden • si Slovenia • sk Slovakia • al Albania • me Montenegro • mk North Macedonia • rs Serbia • tr Turkey • ch Switzerland • is Iceland • li Liechtenstein • no Norway • uk United Kingdom

<p>years (optional)</p>	<p>[Int]</p>	<p>Default=None. 4-digit year/s for which the target value is being requested. If blank, all available years are returned.</p>
<p>pollutants (optional)</p>	<p>[String]</p>	<p>A list of pollutant-types to be returned. Options include:</p> <ul style="list-style-type: none"> ● co: Carbon monoxide ● pm25 Particulate matter (droplets that are 2.5 microns or less) ● pm10 Particulate matter 10 (droplets that are 10 microns or less) ● nmvoc Non-methane volatile organic compounds ● no Nitrous oxide ● so Sulfur oxide
<p>sectors (optional)</p>	<p>[String]</p>	<p>The following sector codes aggregate the emissions from the group of emissions in their corresponding list</p> <ol style="list-style-type: none"> 1. oil-gas-production: <ul style="list-style-type: none"> ○ Manufacture of solid fuels and other energy industries ○ Fugitive emission from solid fuels: Coal mining and handling ○ Fugitive emission from solid fuels: Solid fuel transformation ○ Other fugitive emissions from solid fuels ○ Fugitive emissions oil: Refining / storage ○ Distribution of oil products ○ Other fugitive emissions from energy production ○ Cement production ○ Fugitive emissions from natural gas ○ Venting and flaring 2. mfg-combustion: <ul style="list-style-type: none"> ○ Stationary combustion in manufacturing industries and construction: Iron and steel ○ Stationary combustion in manufacturing industries and construction: Non-ferrous metals ○ Stationary combustion in manufacturing industries and construction: Chemicals ○ Stationary combustion in manufacturing industries and construction: Pulp ○ Stationary combustion in manufacturing industries and construction: Food processing

		<ul style="list-style-type: none"> ○ Stationary combustion in manufacturing industries and construction: Non-metallic minerals ○ Mobile Combustion in manufacturing industries and construction ○ Stationary combustion in manufacturing industries and construction: Other <p>3. aviation:</p> <ul style="list-style-type: none"> ○ International aviation LTO (civil) Domestic aviation LTO (civil) ○ International aviation cruise (civil) ○ Domestic aviation cruise (civil) International maritime navigation <p>4. road-transport:</p> <ul style="list-style-type: none"> ○ Road transport: Passenger cars Road transport: Light duty vehicles ○ Road transport: Heavy duty vehicles and buses ○ Road transport: Mopeds & motorcycles ○ Road transport: Gasoline evaporation ○ Road transport: Automobile tyre and brake wear ○ Road transport: Automobile road abrasion <p>Railways</p> <p>5. water-transport:</p> <ul style="list-style-type: none"> ○ International inland waterways National navigation (shipping) <p>6. other-transport:</p> <ul style="list-style-type: none"> ○ Multilateral operations Transport (fuel used) <p>7. agriculture:</p> <ul style="list-style-type: none"> ○ Agriculture/Forestry/Fishing: Stationary ○ Agriculture/Forestry/Fishing: Off-road vehicles and other machinery ○ Agriculture/Forestry/Fishing: National fishing ○ Manure management - Dairy cattle ○ Manure management - Non-dairy cattle Manure management - Sheep ○ Manure management - Swine Manure management - Buffalo ○ Manure management - Goats Manure management - Horses ○ Manure management - Mules and asses Manure management - Laying hens ○ Manure management - Broilers Manure management - Turkeys ○ Manure management - Other poultry Manure
--	--	--

		<p>management - Other animals</p> <ul style="list-style-type: none"> ○ Inorganic N-fertilizers (includes also urea application) ○ Animal manure applied to soils Sewage sludge applied to soils ○ Other organic fertilisers applied to soils (including compost) ○ Urine and dung deposited by grazing animals ○ Crop residues applied to soils Indirect emissions from managed soils ○ Farm-level agricultural operations including storage Off-farm storage ○ Cultivated crops Use of pesticides ○ Field burning of agricultural residues Agriculture othe <p>8. chemical-mineral-production:</p> <ul style="list-style-type: none"> ○ Ammonia production Nitric acid production Adipic acid production ○ Carbide production Titanium dioxide production Soda ash production ○ Chemical industry: Other Iron and steel production ○ Ferroalloys production Aluminium production Magnesium production ○ Lead production Zinc production Copper production ○ Nickel production Other metal production ○ Lime production Glass production ○ Quarrying and mining of minerals other than coal <p>9. residential:</p> <ul style="list-style-type: none"> ○ Residential: Household and gardening (mobile) ○ Domestic solvent use including fungicides Road paving with asphalt ○ <p>10. waste-treatment:</p> <ul style="list-style-type: none"> ○ Biological treatment of waste - Composting ○ Biological treatment of waste - Anaerobic digestion at biogas facilities ○ Municipal waste incineration Industrial waste incineration ○ Hazardous waste incineration Clinical waste incineration ○ Sewage sludge incineration Cremation Other waste incineration
--	--	--

		<ul style="list-style-type: none"> ○ Open burning of waste Domestic wastewater handling ○ Industrial wastewater handling Other wastewater handling ○ Other waste Other (included in national total for entire territory) ○ Biological treatment of waste - Solid waste disposal on land <p>11. construction:</p> <ul style="list-style-type: none"> ○ Construction and demolition Storage Other mineral products ○ Asphalt roofing Coating applications Degreasing Dry cleaning <p>12. other:</p> <ul style="list-style-type: none"> ○ Pipeline transport Other Commercial/institutional: Stationary ○ Commercial/institutional: Mobile Residential: Stationary ○ Other stationary (including military) ○ Chemical products Printing Other solvent use Other product use ○ Pulp and paper industry Food and beverages industry ○ Other industrial processes Wood processing Production of POPs ○ Consumption of POPs and heavy metals (e.g. electrical and scientific equipment) ○ National total for the entire territory (based on fuel sold) ○ National total for compliance assessment (please specify all details in the IIR) ○ Other not included in national total of the entire territory <p>13. natural-emissions:</p> <ul style="list-style-type: none"> ○ Volcanoes Forest fires Other natural emissions
--	--	---

Response values:

key	type	description
country	String	Country code will match the country codes from the request.

year	String	4-digit value of the year of the target
pollutant	String	Pollutant codes will match the pollutant codes from the request.
sector	String	The sector code will match the sector codes from the request.
emission-level	Float	The emission level for that particular pollutant in Gg (1000 tonnes)

```
{
  "de": {
    "2017": {
      "co": {
        "aviation" : 1.32,
        "construction" : 0.48
      },
      "pm25": {
        "aviation" : 0.44,
        "construction" : 0.43
      }
    }
  }
}
```

/aq_api/region_info

Provides a list of NUTS regions and associated information.

Example request:

For all regions:

http://localhost:8000/aq_api/region_info?version=v1

For a given level:

http://localhost:8000/aq_api/region_info?version=v1&levels=1

Request parameters:

parameter	type	description
version	String	Default=None. API version (e.g - 'v1'). If blank, default to the current version.
levels (optional)	[Int]	The NUTS level to be returned: <ul style="list-style-type: none"> • 0 = NUTS0 (countries) • 1 = NUTS1 • 2 = NUTS2 • 3 = NUTS3 <p>If not supplied or None, all levels are returned</p>

Response values:

key	type	description
region	String	The region code of the NUTS region
level	Int	The NUTS level of the region
name	String	The name of the region
country	String	The country that the region is located in

Example response:

```
{
```

```
0 :
  { "de", {
    "name" : "Germany",
    "country_code" : "de"},
    { "it", {
    "name" : "Italy",
    "country_code" : "it"},
    ...
  }
  ,
  1 : [ {} , {} .. ],
}
```

/aq_api/region_boundaries

Provides a list bounding shapes for NUTS regions.

Example request:

For all regions:

http://localhost:8000/aq_api/region_boundaries?version=v1

For a set of regions

http://localhost:8000/aq_api/region_boundaries?version=v1®ions=de,fr

Request parameters:

parameter	type	description
version	String	Default=None. API version (e.g - 'v1'). If blank, default to the current version.

level (optional)	Integer	Optional. Nuts level. If None, all levels are returned.
regions (optional)	[String]	<p>Default=None. List of codes for regions to include. If blank all regions are included.</p> <p>Options include</p> <p>NUTS 0 (countries):</p> <ul style="list-style-type: none"> • at Austria • cz Czech Republic • de Germany • dk Denmark • ee Estonia • fi Finland • fr France • el Greece • hu Hungary • hr Croatia • ie Ireland • it Italy • lt Lithuania • lu Luxembourg • lv Latvia • mt Malta • nl Netherlands • pl Poland • pt Portugal • ro Romania • es Spain • se Sweden • si Slovenia • sk Slovakia • al Albania • me Montenegro • mk North Macedonia • rs Serbia • tr Turkey • ch Switzerland • is Iceland • li Liechtenstein • no Norway

		<ul style="list-style-type: none"> uk United Kingdom
--	--	---

Response values:

key	type	description
region	String	Region code will match the region code from the request.
level	Int	The NUTS level of the region
name	String	The name of the region
country_code	String	The country that the region is located in
geography	String	The region's bounding shape in latitude and longitude pairs

Example response:

```

{
  0 :
    { "de", {
      "name" : "Germany",
      "country_code" : "de",
      "geography" : "POLYGON ((21.478999999999992
45.192999999999978, 21.358000000000005 44.822000000000027,
22.012000000000005 44.601999999999968, 22.015999999999982
44.598999999999966, 22.500000000000000....." }},
    { "it", {
      "name" : "Italy",
      "country_code" : "it",
      "geography" : "POLYGON ((21.478999999999992
45.192999999999978, 21.358000000000005 44.822000000000027,

```

```
22.0120000000000005 44.601999999999968, 22.015999999999982
44.598999999999966, 22.500000000000000....."} ,
    ...
  }
  ,
  1 : [ {} , {} .. ],
}
```

Region-time API response

```
{
  "de" : {
    "2020-03-19" : {
      "o3" : {
        "annual-target" : .7,
        "values" : [.5656,...,5656]
        "ytd-avg" : .5212,
      }
      "co" : {
        "annual-target" : 12,
        "values" : [14.564,..14.564]
        "ytd-avg" : 12.12,
      }
    }
    "2020-03-20" : {
      "o3" : {
        "annual-target" : .7,
        "values" : [.4911,...,4911]
        "ytd-avg" : 50.005,
      }
    }
  }
}
```



```
        "co" : {  
            "annual-target" : 12,  
            "values" : [12.212,.., 12.212]  
            "ytd-avg" : 12.09,  
        }  
    }  
}
```

Appendix G - Tests and Code Coverage

We're using the django test TestCase to run our unit tests and the django coverage plugin to check our test coverage. These are the instructions on how to run the tests and generate the test coverage report.

In order to run the tests, the application will need a local sqlite3 database.

Make a db.sqlite3 file:

1. In settings.py, comment out the main database settings
2. Un-comment the part to make sqlite3 the database
3. python manage.py makemigrations
4. python manage.py migrate
5. Now you can change the comments back in settings.py because there's a setting in settings.py that will automatically use the local sqlite3 database if "test" is in the command

In the base eugreendeal directory, these will run the tests

```
> ./manage.py test airpollution.tests.eeamodeltests  
> ./manage.py test airpollution.tests.eurostatmodeltests  
> ./manage.py test airpollution.tests.nutsregions_tests  
> ./manage.py test airpollution.tests.pollutants_tests  
> ./manage.py test airpollution.tests.tests  
> ./manage.py test airpollution.tests.observations_tests
```

We're using the django coverage plugin to check our test coverage

Install the django coverage plugin using:

```
> pip install django_coverage_plugin
```

On the same directory level as eugreendeal

1. > mkdir eucoverage
2. > cd eucoverage

Run the tests using the -a flag (append)

```
> coverage run -a ../eugreendeal/manage.py test  
airpollution.tests.eeamodeltests --keepdb
```

```
> coverage run -a ../eugreendeal/manage.py test  
airpollution.tests.eurostatmodeltests --keepdb
```

```
> coverage run -a ../eugreendeal/manage.py test  
airpollution.tests.observations_tests --keepdb
```

```
> coverage run -a ../eugreendeal/manage.py test  
airpollution.tests.pollutants_tests --keepdb
```

```
> coverage run -a ../eugreendeal/manage.py test  
airpollution.tests.nutsregions_tests --keepdb
```

```
> coverage run -a ../eugreendeal/manage.py test airpollution.tests.tests  
--keepdb
```

View the coverage results on the screen. The major logic isn't in the views so they are ignored.

```
> coverage report --include=../eugreendeal/* --omit */views/*
```

Make html files of the results

```
> coverage html --include=../eugreendeal/* --omit */views/*
```

If html files are made, you can see the report by opening the `eucoverage/htmlcov/index.html` file in your browser

Coverage report: 81%

Module	statements	missing	excluded	coverage ↓
/eugreendeal/airpollution/models/models_nuts.py	100	52	0	48%
/eugreendeal/airpollution/models/models_copernicus.py	58	21	0	64%
/eugreendeal/airpollution/models/models_eea.py	45	12	0	73%
/eugreendeal/airpollution/models/models_observations.py	208	52	0	75%
/eugreendeal/eugreendeal/settings.py	32	8	0	75%
/eugreendeal/manage.py	12	2	0	83%
/eugreendeal/airpollution/models/models_pollutants.py	121	13	0	89%
/eugreendeal/airpollution/models/models.py	20	2	0	90%
/eugreendeal/airpollution/__init__.py	0	0	0	100%
/eugreendeal/airpollution/admin.py	47	0	0	100%
/eugreendeal/airpollution/apps.py	3	0	0	100%
/eugreendeal/airpollution/management/__init__.py	0	0	0	100%
/eugreendeal/airpollution/migrations/0001_initial.py	8	0	0	100%
/eugreendeal/airpollution/migrations/__init__.py	0	0	0	100%
/eugreendeal/airpollution/models/__init__.py	7	0	0	100%
/eugreendeal/airpollution/models/models_eurostat_population.py	15	0	0	100%
/eugreendeal/airpollution/tests/__init__.py	0	0	0	100%
/eugreendeal/airpollution/tests/eeamodeltests.py	13	0	0	100%
/eugreendeal/airpollution/tests/eurostatmodeltests.py	12	0	0	100%
/eugreendeal/airpollution/tests/nutsregions_tests.py	18	0	0	100%
/eugreendeal/airpollution/tests/observations_tests.py	36	0	0	100%
/eugreendeal/airpollution/tests/pollutants_tests.py	65	0	0	100%
/eugreendeal/airpollution/urls.py	16	0	0	100%
/eugreendeal/eugreendeal/__init__.py	0	0	0	100%
/eugreendeal/eugreendeal/urls.py	4	0	0	100%
Total	840	162	0	81%

coverage.py v5.1, created at 2020-05-02 20:38

Appendix H - Application Documentation

The application uses the Sphinx document generator to create documentation. The documents are included in with the code under eugreendeal/docs. When changes are made to the code base, the documents can be regenerated using the following instructions:

```
> brew install sphinx-doc  
> pip install sphinx_rtd_theme
```

On the same level as the eugreendeal directory

1.

```
> mkdir eudocs
```
2.

```
> cd eudocs
```
3.

```
> sphinx-quickstart
```

Modify the conf.py as follows:

```
# Path setup section  
import os  
import sys  
sys.path.insert(0, os.path.abspath('.'))  
sys.path.insert(0, os.path.join(os.path.abspath('.'), '../eugreendeal'))  
  
os.environ['DJANGO_SETTINGS_MODULE'] = 'eugreendeal.settings'  
import django  
django.setup()  
  
# automodule error fix:  
extensions = ['sphinx.ext.autodoc']  
  
# html_theme = 'alabaster'  
html_theme = 'sphinx_rtd_theme'
```

Modify the index.rst file by adding a new line and modules below the toctree caption line.

It should look like this:

```
.. toctree::  
   :maxdepth: 2  
   :caption: Contents:  
  
   modules
```

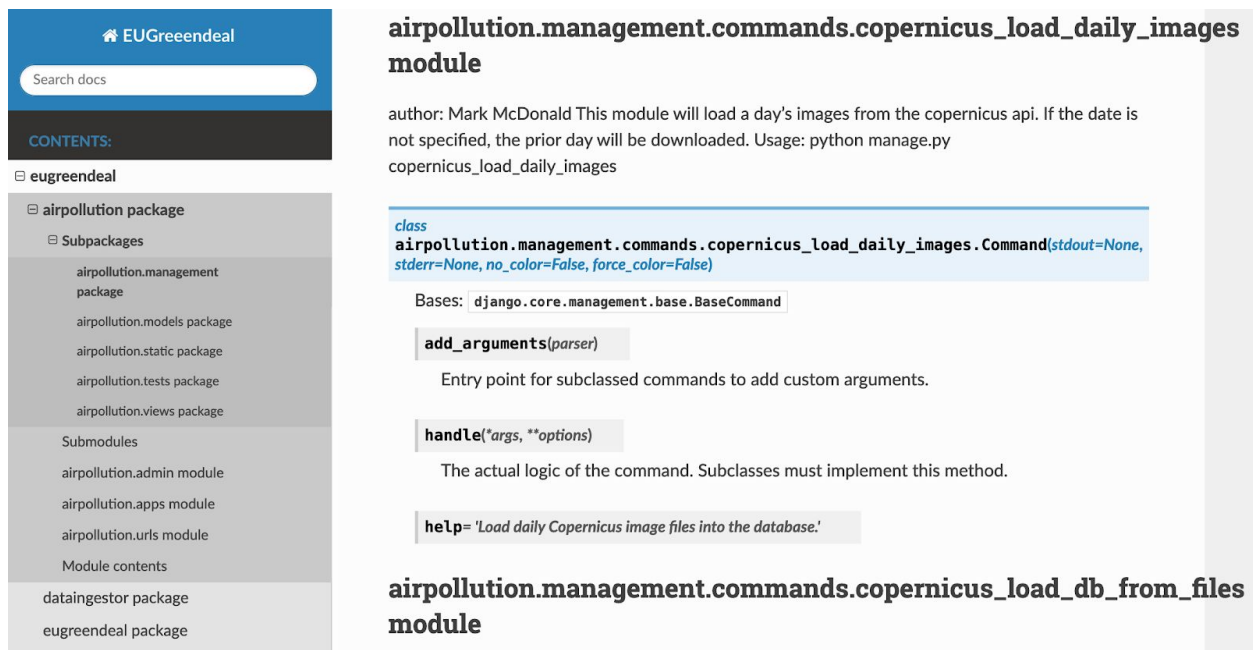
Make the rst files, ignoring the migrations

```
> sphinx-apidoc -o . ../eugreendeal ../airpollution/migrations/*.*
```

Generate the files by using the command:

```
> make html
```

View the document by opening the eudocs/docs/_build/html/index.html file in your browser.



The screenshot shows a web browser displaying a Django documentation page. On the left is a navigation sidebar with a search bar and a tree view of the project structure. The main content area displays the documentation for the `airpollution.management.commands.copernicus_load_daily_images` module. The page includes an author note, a class definition for `airpollution.management.commands.copernicus_load_daily_images.Command`, its base class `django.core.management.base.BaseCommand`, and two methods: `add_arguments(parser)` and `handle(*args, **options)`. A `help` attribute is also shown.

airpollution.management.commands.copernicus_load_daily_images module

author: Mark McDonald This module will load a day's images from the copernicus api. If the date is not specified, the prior day will be downloaded. Usage: python manage.py copernicus_load_daily_images

class
`airpollution.management.commands.copernicus_load_daily_images.Command(stdout=None, stderr=None, no_color=False, force_color=False)`

Bases: `django.core.management.base.BaseCommand`

add_arguments(parser)
Entry point for subclassed commands to add custom arguments.

handle(*args, **options)
The actual logic of the command. Subclasses must implement this method.

help= 'Load daily Copernicus image files into the database.'

airpollution.management.commands.copernicus_load_db_from_files module

Appendix I - Developers Manual

Index

[Overview](#)

[Installation](#)

[Clone Repository](#)

[Install Dependencies](#)

[Python](#)

[Install pip](#)

[Pipenv](#)

[Psycopg2](#)

[SpatialIndex](#)

[Docker](#)

[Setup Local Database](#)

[Activate Environment](#)

[Activate the pipenv environment for the project. This will install all of the required packaged from the specified within Pipfile.lock](#)

[Delete Existing Migrations](#)

[Setup Tables in Local Database](#)

[Create a Superuser](#)

[Create a Default Persona](#)

[Launch Application](#)

[Load Data](#)

[Run Docker Image \(Optional\)](#)

[Architecture / Extending Functionality](#)

[Architecture Overview](#)

[Extending Application](#)

[Data Model](#)

[Data Ingestion](#)

[Data Access \(API\)](#)

[Views](#)

Overview

The EU Green Deal application described in this document is designed to collect data from a variety of sources and present consolidated results in an easy-to-understand manner. Although the data collected is scientific in nature, the application is intended for non-scientific users to understand trends and patterns specifically in relation to environmental issues and more specifically in relation to the EU Green Deal Project.

The EU Green Deal provides a roadmap for environment improvement through a roadmap of actions is an EU initiative that sets a variety of targets that are aimed at becoming climate neutral by 2050 by pursuing 4 main goals:

- Zero pollution
- Affordable secure energy
- Smarter transport
- High-Quality Food

The application described in this document is a prototype application that focused on the “Zero Pollution” target by reporting air quality measurements and comparing them to targets set forth in the Green Deal agreement.

The technical structure of the application is based on a Django Web Server (which is based on Python). Data for the application is stored in a PostgreSQL database which can be hosted in a local Docker container or any cloud service of your choice.

Knowledge of Django (and Python) is strongly recommended before attempting to extend the application; however, the application can be downloaded and run locally without any knowledge of Django.

Installation

The following steps describe the detailed steps necessary to install the application locally and prepare it for extended development. If you run into problems during any of these steps and simply want to run the prototype application, you can run the entire application as a Docker container locally. See [Run Docker Image \(Optional\)](#) for details.

IMPORTANT NOTE: This application was developed on a Mac OS platform and has been tested to work on a Mac and should also work on Linux. Installation on Windows may require special handling of the GeoPandas installation. Please refer to the documentation for

GeoPandas installation on Windows if you run into a problem installing the dependencies using Windows. Using the Docker instance of the application should be possible to run on any platform that supports Docker. To do this, follow the instructions under [Run Docker Image \(Optional\)](#).

Clone Repository

Access to the repository is restricted and not public. If you do not have an account which can access the application, please request access via email to mcdomx@gmail.com. Please include a desired user ID for the account that will be created for you. Once you have an account established, you can clone the repository using:

```
> git clone  
https://bitbucket.org/capstoneeureporting/eugreendeal/src/master/
```

Install Dependencies

Python

The application requires Python 3.7. Please refer to the Python installation documentation if you do not have 3.7 installed locally. To check your Python version:

```
> python --version
```

Any 3.7 version will work.

Install pip

You're welcome to install required packages using other installers, but we recommend using pip. [Instructions to install pip can be found here.](#)

Pipenv

The application uses pipenv to manage dependencies. To use pipenv, you will first need to install the python module:

```
> pip install pipenv
```

Psycopg2

Check to make sure you have the PostgreSQL adapter installed:

```
> pip freeze | grep psycopg2
```

If it isn't installed already you can install it using one of these commands:

```
> pipenv install psycopg2-binary
```

ALTERNATIVE: If you have issues installing via pipenv, you can install with:

```
> brew install postgresql
```

SpatialIndex

SpatialIndex is a library used to calculate data that is rendered in some of the visualizations in the prototype application.

```
> brew install spatialindex
```

Docker

The initial setup will be done using a Docker database. This requires docker to be installed locally. Instructions for installing Docker can be found at <https://docs.docker.com/get-docker/>.

Setup Local Database

The application database can be installed using a local Docker Postgres container.

Navigate to the application './eugreendeal/infrastructure' directory:

```
> docker-compose up -d db
```

The first time this is run, the proper container will be downloaded. This may take several minutes.

Activate Environment

Activate the pipenv environment for the project. This will install all of the required packages from the specified within Pipfile.lock

From the './eugreendeal/' directory:

```
> pipenv install
```

When prompted, enter the following to activate the virtual environment:

```
> pipenv shell
```

Delete Existing Migrations

Delete all the Django migrations from airpollution/migrations. Leave the __init__.py file in place. The migrations are located in the './eugreendeal/airpollution/migrations' directory.

Setup Tables in Local Database

Once the Docker database container is up and running, the application's data tables can be created:

From the './eugreendeal/' directory:

```
> python manage.py makemigrations  
> python manage.py migrate
```

Create a Superuser

As a superuser, you have the ability to navigate to table views in order to see the data stored in the application.

From the './eugreendeal/' directory:

```
> python manage.py createsuperuser
```

Follow the prompts to set up a username and password.

Create a Default Persona

The application needs a default persona for a non superuser to be assigned a persona. To do this, we need to create a default persona if this is the first time the application is being initialized.

From the `../eugreendeal/` directory:

```
> python manage.py createdefaultpersona
```

Launch Application

At this point, you can launch the application to see that it is functioning properly.

From the `../eugreendeal/` directory:

```
> python manage.py runserver
```

Navigate to the site at:

<http://localhost:8000/>

To see the Django administrative console, navigate to:

<http://localhost:8000/superadmin/>

IMPORTANT NOTE: At this point, you have not populated the database with any data so you'll get a number of errors if you try to load in plots. Please follow the instructions to load in data below:

Load Data

The application relies on a large amount of data to provide meaningful analysis and reports. For testing, a limited set of data can be loaded.

First, shut down the application using (Ctrl + C). Then run the following command

```
> python manage.py populate_db
```

IMPORTANT NOTE: this process can still take a few hours to complete. Please be patient.

Run Docker Image (Optional)

If you face any compatibility issues while installing any dependency or due to some reason you are unable to bring up the Django server, you can opt to build a local container and run it as a docker image along with the docker postgres instance. To do that, navigate to the project home directory and execute:

```
> docker build -t eugdserver .
```

Once the build is complete, you can bring up the entire infrastructure by navigating into ../eugreendeal/infrastructure directory and executing:

```
> docker-compose up -d
```

To bring stop the docker instance:

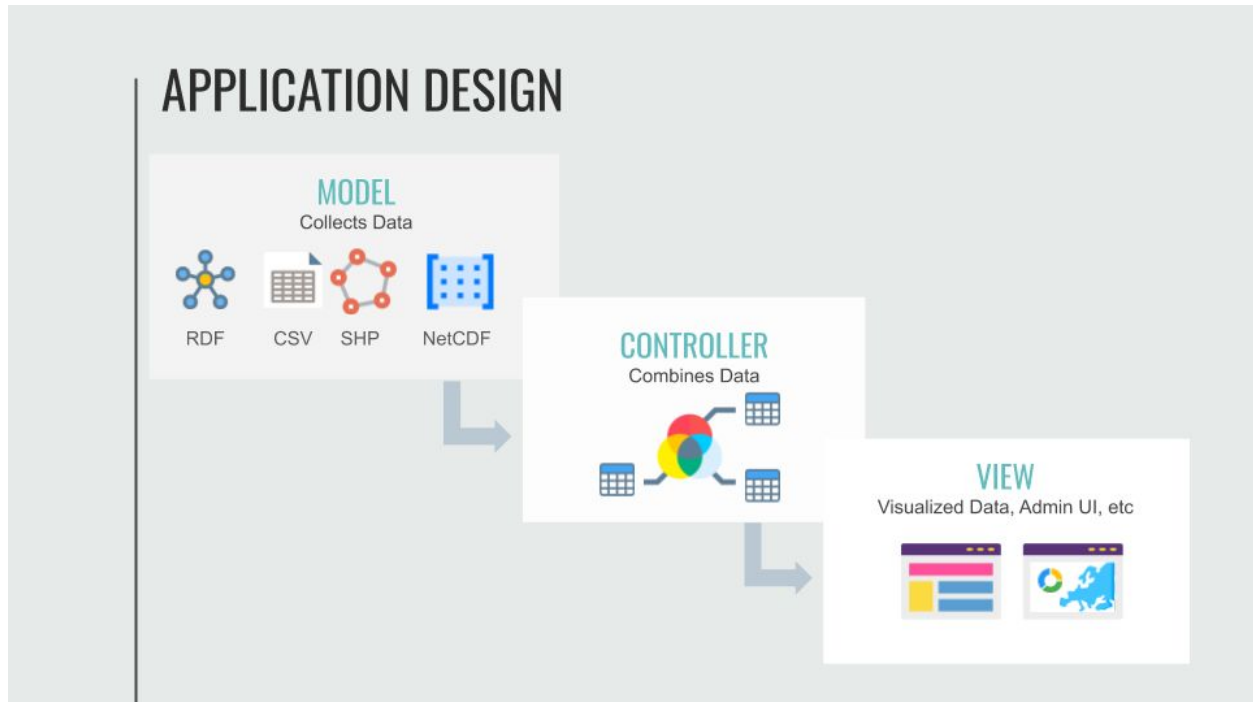
```
> docker-compose down
```

Architecture / Extending Functionality

The application is based on a Django Web Server which is python dependent. The application allows for the extension of new data sources and visualizations. Knowledge of Django is highly recommended and knowledge of Python is essential to extend the application.

Architecture Overview

The application follows a traditional MVC design approach.



Extending Application

Extending the application requires new components to be built which include:

- Creating DB tables for new data
- Collecting and loading external data
- Creating accessor functions to access the table data
- Creating API functions to consolidate data for front-end use
- Creating visualizations
- Placing the visualizations on web pages

If new data is not necessary, then only new visualizations must be created and you can start at the [Data Access \(API\)](#) or [Views](#) section of this document. Data for your new visualization may be readily accessible via existing API functions, so you may not need to create any functions to access data specific to your visualization. It is advisable to start with the [Views](#) section and go back and add API functions as you find that they are necessary to build your visualization.

A summary of the extendable classes are listed in the table below:

Note: Examples for each class are included in the prototype application.

APPLICATION DESIGN

The following components require configuration to extend the application to collect new data and present it on a web page.

Model Collects Data		Controller Combines Data		View Presents Data
Data Ingestor	Data Model	API	View	HTML Page
<p>Handles the specifics of communicating with the external data source and bringing the data to the application.</p> <p>A new class is created for each data source which extends the DataSource.py class</p> <p>A Django loading process can be triggered by including a Command function in the 'management' directory.</p>	<p>Django uses an ORM wrapper for database tables. For each table, a new data model must be created.</p> <p>Each model extends the Django models.Model class.</p> <p>Each model class also includes basic data accessors for data in the model.</p>	<p>An API layer consolidates data from one or many Data Models into simplified functions that can be used by the front-end components.</p> <p>Functions in the API layer should call the data accessor functions included in the model layer.</p> <p>An initial API is included in the prototype application in the ' directory.</p>	<p>Each visualization corresponds to a views.py file stored in the './greendeal/airpollution/views/' directory.</p> <p>Functions called in these files should return JSON objects that can be handled by a Django webpage which will use Jinja to render the data.</p>	<p>Pages are built using Jinja templating. New visualization can be created in a templated html page which can be included in a respective dashboard.</p>

Django uses a predefined directory structure to store files of types. Subdirectories under each directory can be created, but files of similar types must be kept under the 'model', 'views' and 'templates' directories. This is important as Django will look for files under certain directories to perform application management tasks.

A list of the directories used is presented below:

DIRECTORY STRUCTURE

Directory Structure	Description	Files used to extend application
<ul style="list-style-type: none"> <ul style="list-style-type: none"> airpollution <ul style="list-style-type: none"> management media models templates views dataingestor eugreendeal infrastructure 	<ul style="list-style-type: none"> Prototype Application Includes load commands Stores downloaded files Includes a model file for each data table Holds the HTML pages Holds the application logic for creating pages Holds python files used to gather external data Main Project Settings Docker Database Configuration 	<ul style="list-style-type: none"> Load commands Data Model HTML Page API and View Data Ingestor

Data Model

If a new presentation of data requires new data to be externally collected, a new data model must be created. This will likely require some knowledge of how Django's ORM works, but you can follow the templates created in the prototype application to cover most cases.

Step 1:

Create a new model class python file in the './eugreendeal/airpollution/models' directory.

Step 2:

In that file, create your new class inheriting from the Django models.Model class. (use examples from the base application as a template.)

For each field that will be used in the application, create a variable from the models class.

Step 3:

In the new class, create data accessor functions which the API will use to get data from the new model. These functions should specialize in collecting summarized data efficiently from the model. No accessor functions should access other models. Functions that combine data are in the API layer described later (see [Data Access](#)).

Data Ingestion

After a model is created, the logic for collecting external data can be started. The design of this task will vary depending on what datasource you will use and how that data is accessed. Of the steps necessary to extend the application, creating a process to load external data is likely to require the most time, effort and skill.

A new class is required to handle the data collection task. This new class must inherit from the DataSource class which can be found in the './eugreendeal/dataingestor/' directory. The DataSource class is specific to this application and not a Django class. The DataSource class only requires a `load_data()` and a `load_dummy_data()` function to be implemented but you can, and should, create others to build the process.

These functions are expected to execute the process for collecting external data and saving it into the application database. The `load_data()` function can accept a 'kwargs' dictionary argument allowing flexibility for accepted arguments. These arguments will likely be necessary to set parameters for restricting the data load to a specific date or for only certain elements.

Since the variety of datasources is extremely large, no specific instructions are included but a variety of working examples are provided with the prototype application.

Data Access (API)

In this section, we review the API layer and its purpose.

Access to each data source's table data is created in the model (see [Data Model](#)). Those functions are basic data accessors and provide a benefit by making fast data queries without requiring the API to know the detailed model structure.

The API layer provides intelligence for the application. Here, data may be combined from several calls to one of many data model accessors and organized for concise presentation to

the front end. Functions in the API layer will be called by views, each of which are designed to serve a specific visualization (see [Views](#)). The flow of data requests is simplified below:

Front-end HTML → my_view → the_application_API → my_model(s)

The prototype application includes an API script in the 'views' directory called 'aq_api_v1.py'. Existing functions can be re-used or additional functions can be added. You may also create a new api.py file to organize your work more effectively.

Creating an API function is dependent on the tasks necessary and follows no specific formatting requirements in the application. The main purpose of the API layer is to prevent any views from needing direct access to the data model objects as well as preventing any views from needing to combine data from various functions. The primary guideline to follow is that API functions should use model accessor functions to make calls to the database. Generally, the API should not include functions that directly call a database table.

Views

If the application already has the data you need, you may be able to start at this step for extending the application. If you find that you need a new API function during this step, new API functions should be created (see [Data Access](#)).

Views are a Django term and refer to Python scripts that collect data and render a web page or a component of a web page with that data. For the purposes of extending this application, each new view should represent a new visualization and should collect the data necessary for the visualization.

Step 1:

Create a new views_<name of view>.py file in the 'views' directory. The contents of this file should be a named function that will be called to render your visualization. The function should return accept a single argument of 'request' and return a 'render' object.

```
def my_visualization_view(request):  
    logic and code  
    return render(request, 'airpollution/my_vis.html', dict(data1='my_data', data2='more_data'))
```

This will render the page passing the dictionary of data into the page which can be accessed using jinja templating.

Alternatively, you can return a JSON object if your page uses javascript to render the page.

```
def my_visualization_view(request):  
    ..logic and code  
    return JsonResponse(dict(data1='somedata', data2='moredata'))
```

Step 2:

Create a new HTML file that will host your visualization. Copy the 'visualization_template.HTML' file as a starting point.

This new file should only include the javascript and HTML tags specific for the visualization you are creating. This new HTML file inherits from the site's layout and navigation design.

Step 3:

Create a route to your visualization.

During development, you may want to see your progress. To create a route to your visualization without rendering an entire dashboard page, you can create a route in the 'eugreendeal/airpollution/urls.py' file by adding a route to the 'urlpatterns' list. Follow the logic presented in the template. Your added item should follow the pattern below:

```
path("my_route_name", views.my_visualization_view, name="my_visualization_name"),
```

You will be able to reach your page at:

http://localhost:8080/my_route_name

Later, this route will be used to place your visualization into an existing dashboard.

Step 4:

Once the visualization is completed, it can be integrated into a dashboard page. Select the dashboard page to include the visualization into and find the location for your page based on the dashboard layout. Insert the call to your page's route where applicable.

New visualizations are quite simple to insert into HTML. First add a div with a unique id to reference. This is where your visualization will be placed. For example;

```
<div id="reg-rep-ch-1"> </div>
```

The div is populated using a JavaScript function in the dashboard page. The following `update()` function is called in the dashboard page when the page is first loaded and when the page's selections are changed. The function performs the following:

- Selects the specified div in the page
- Initially places a a preloader gif image in that div
- If it receives a successful JSON response from the API, then it populates the div with the Bokeh plot embedded within the JSON response. Note that the API function in the example below is called from the route `pollution_over_time`. This route must be included in the `urls.py` urlpatterns list.
- If the JSON response throws an error, it places an error image within that div

```
function update(country) {
    var chart_div = document.querySelector('#reg-rep-ch-1');
    chart_div.innerHTML = "<img style=\"height:300px;width:400px\" src={% static
'airpollution/assets/img/gif/pre-loader-2.gif' %}>";

    fetch('/pollution_over_time?pollutant=PM10&start_date=2020-02-29&end_date=2020-03-01&c
ountries=' + country)
        {#pollutant=PM25&start_date=2020-01-01&end_date=2020-03-22#}
        .then(function (response) {
            return response.json();
        })
        .then(function (item) {
            var chart_div = document.querySelector('#reg-rep-ch-1');
            chart_div.innerHTML = "";
            Bokeh.embed.embed_item(item, 'reg-rep-ch-1');
            chart_div.setAttribute('data-url',
'/pollution_over_time?pollutant=PM25&start_date=2020-01-01&end_date=2020-03-22');
        }).catch((error) => {
```

```
var chart_div = document.querySelector('#reg-rep-ch-1');
chart_div.innerHTML = "";
chart_div.innerHTML = "<img style=\"height:200px;width:400px\" src={% static
'airpollution/assets/img/tech-snag.png' %}>";

});
}
```

Once implemented, the above steps should yield your visualization presented in the browser window when navigating to the chosen route.