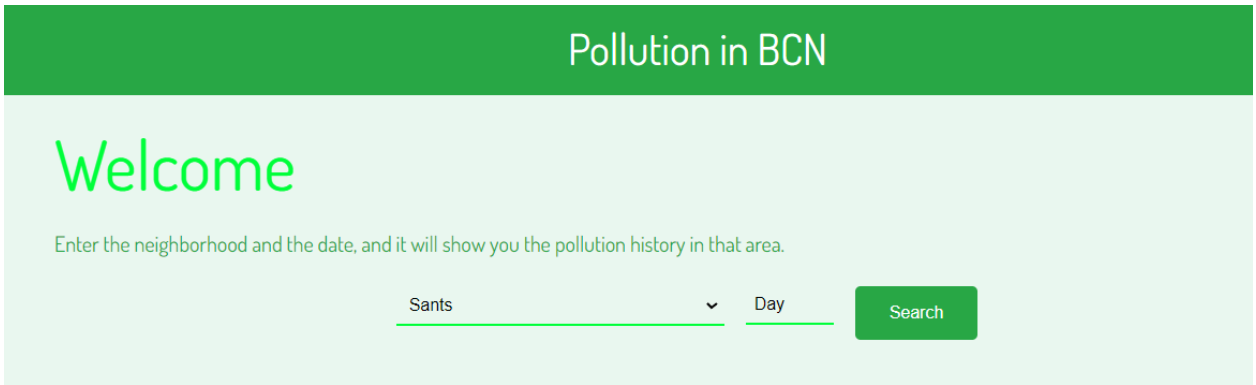
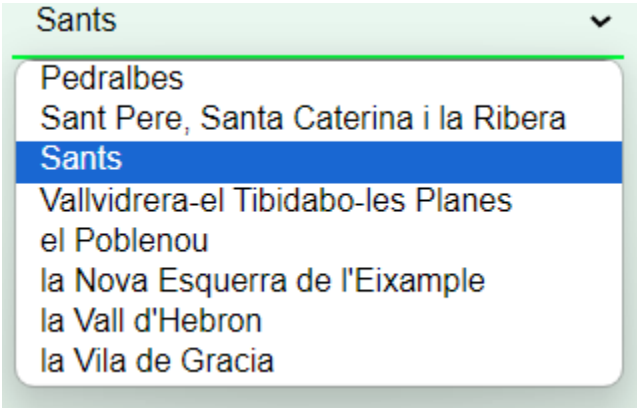
Calidad de Aire en la ciudad de Barcelona

By Jhester John Calma and
John Carlo Rodriguez

Web interface:

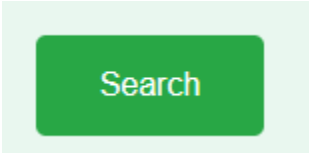


Neighborhood selection:



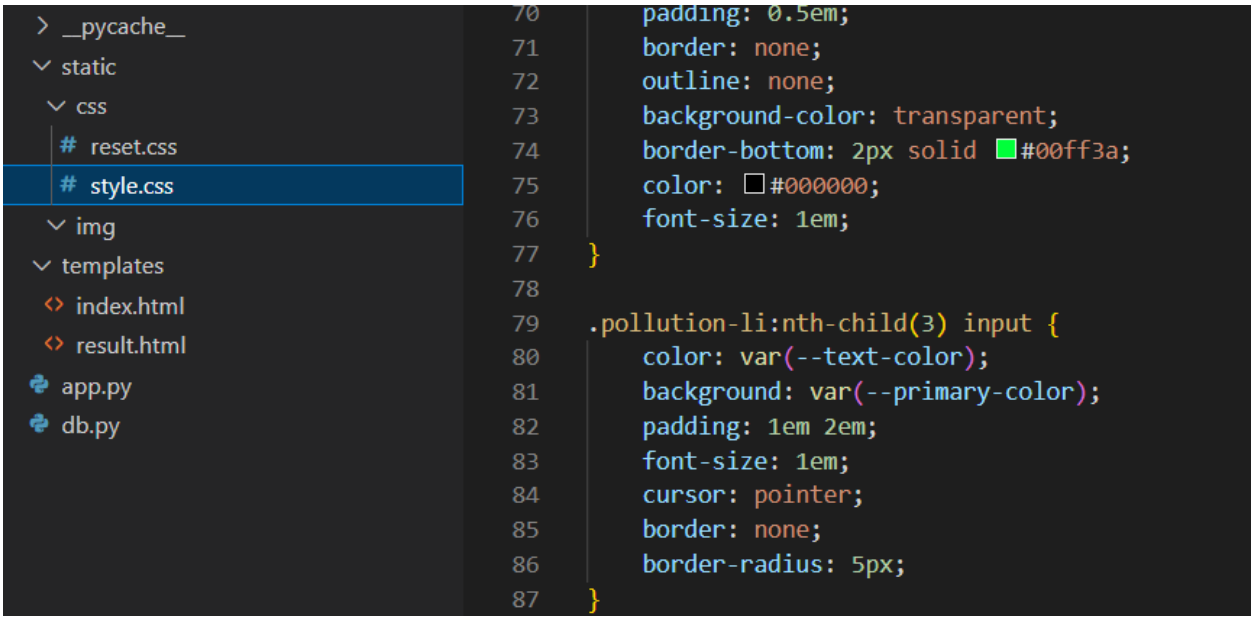
Day selection (from 1-31)

Search bar:



Style.css:

This CSS defines a coherent and responsive layout system for a web page, using CSS variables for themes, Google Fonts for custom typography, and modern CSS techniques like Flexbox for layout management. Provides a consistent look and feel across multiple components, such as headers, forms, tables, and error messages.



Reset.css: (reset stylesheet)

This reset stylesheet ensures that browsers start from a consistent base by removing default styles applied by browsers. This allows developers to have more control over the styling of their web pages and helps avoid cross-browser inconsistencies.

```
static
  css
    reset.css
    style.css
  img
templates
  index.html
  result.html
app.py
db.py
```

```
    display: block;
}
body {
    line-height: 1;
}
ol, ul {
    list-style: none;
}
blockquote, q {
    quotes: none;
}
blockquote:before, blockquote:after,
```

Index.html:

***Layout**: The page layout includes a header, main content section, and a form for user input.*

***Form**: The form allows users to select a neighborhood from a dropdown and enter a day of the month to query pollution data.*

***Styling**: The page links to reset and custom stylesheets for consistent styling.*

***Flask Templating**: Uses Flask's templating engine to dynamically generate options for the neighborhood dropdown and display error messages.*

This HTML structure is designed to provide a user-friendly interface for querying pollution data in different neighborhoods of Barcelona.

```
<body>
    <div id="wrapper">
        <div class="container-pollution">
            <main class="container-form main-bcn">
                <div class="div-text-container">
                    <h2>Welcome</h2>
                    <p>Enter the neighborhood and the date to see history in that area.</p>
                </div>
                <div class="container-form">
                    <form action="/result" method="post">
                        <ul class="form-pollutions-ul">
                            <li class="pollution-li">
                                <select name="neighborhood" id="neighborhood">
                                    {% for neighborhood in neighborhoods %}
                                        <option value="{{ neighborhood }}">{{ neighborhood }}</option>
                                    {% endfor %}
                                </select>
                            </li>
                            <li class="pollution-li">
                                <input type="number" name="date" id="date" placeholder="Day" min="1" max="31"required>
                            </li>
                            <li class="pollution-li">
```

Result.html:

This HTML template is used to display the results of a pollution data search for a specific neighborhood and day in Barcelona. The page shows the results in a table format and includes a link to go back to the main menu.

***Page Title**: The title of the webpage is "Search Result".*

***Header:** The header contains the main title "Pollution in BCN".*

***Results Display**: The main content area displays the search results in a table format. It dynamically shows the neighborhood and date based on the user's input.*

***Table Structure**: The table includes columns for contaminant name, result value, and unit of measurement. Each row represents a result for a specific contaminant.*

***Back Link**: There is a link to navigate back to the main menu.*

App.py:

This Flask application allows users to input a neighborhood and a day of the month to view pollution data.

This Flask application provides a web interface to query and display pollution data for Barcelona neighborhoods. The application:

- *Has a home page (index.html) where users can select a neighborhood and enter a date.*
- *Handles form submissions and retrieves pollution data from a database.*
- *Displays the results in a table format along with generated statistics and a chart on a result page (result.html).*
- *Provides error handling to inform users if no data is found.*

```python
from flask import Flask, render_template, request
import db

app = Flask(__name__)

@app.route("/")
def index():
    return render_template("index.html", neighborhoods=db.get_neighborhood_list())

@app.route("/result", methods=["POST"])
def result():
    data = request.form
    neighborhood, date = data["neighborhood"], int(data["date"])
    results = db.search_results(neighborhood, date)

    if results:
        stats = db.calculate_statistics(results)
        image_path = db.generate_chart(stats, neighborhood)
        return render_template("result.html", parameters={"neighborhood": neighborhood, "date": date}, results=res
    else:
        return render_template("index.html", neighborhoods=db.get_neighborhood_list(), error_message="No results f
```

Db.py:

This Python code is part of a Flask application that interacts with a MongoDB database to retrieve and process pollution data for neighborhoods in Barcelona.

- *connect_database: Connects to MongoDB.*
- *search_results: Retrieves pollution data for a given neighborhood and date.*
- *get_neighborhood_list: Gets a list of neighborhoods.*
- *calculate_statistics: Computes mean, max, and min statistics for the pollution data.*
- *generate_chart: Placeholder for a function to generate charts.*

This code sets up the necessary functions to interact with the MongoDB database, retrieve pollution data, calculate statistics, and potentially generate charts. These functions will be called by the Flask routes to provide data for the web application.

```python
import numpy as np
from pymongo import MongoClient
from pymongo.server_api import ServerApi
from collections import defaultdict

def connect_database():
    client = MongoClient("mongodb+srv://JCARLO:123@contaminacionbcn.eiic72l.mongodb.net/?retryWrites=true&w=majority&appNa
    return client["ContaminacionBCN"], client

def search_results(neighborhood, date):
    db, client = connect_database()
    station_id = db.Estaciones.find_one({"Nom_barri": neighborhood}).get("Estacio")
    results = []

    if station_id:
        for result in db.CalidadAire.find({"ESTACIO": station_id, "DIA": date}):
            contaminant = db.Contaminantes.find_one({"Codi_Contaminant": result["CODI_CONTAMINANT"]})
            results.append({"result": result["H12"], "desc": contaminant["Desc_Contaminant"], "unit": contaminant["Unitats

    client.close()
    return results if results else None

def get_neighborhood_list():
    db, client = connect_database()
```