Web application - Proof of concept

Overview:

Our web application is built using the MERN stack (MongoDB, Express, React, Node.js) with Python handling ETL tasks. The application features a public registration page, a login page, a protected home page, and a graph view page—all demonstrating a seamless, data-driven user experience.

Technology Stack:

- **Frontend:** React, HTML, CSS, Bootstrap
- **Backend:** Node.js, Express (with REST API)
- **Database:** MongoDB (for secure user registration data)
- **ETL & Data Visualization:** Python, Pandas, Matplotlib, Seaborn
- **Cloud:** (Optional – AWS solutions can be integrated for scalability, such as AWS Redshift for large data storage)

Application Flow:

1. **Registration & Login:**
    - **Registration:** Users provide their email, password, date of birth, and field of work. This information is validated server-side using Node.js and Express before being securely stored in MongoDB.
    - **Login:** Registered users access the login page to authenticate and gain access to the home page.
2. **Home Page:**
    - Accessible only to logged-in users, the home page presents a simple interface with navigation options—including a direct link to the graph view page.
3. **Graph View Page:**
    - The graph view displays dynamic visualizations of greenhouse gas emissions data.
    - **Data Processing & Visualization:**

        - In Jupyter Lab, data was first processed from CSV files using Pandas and then loaded into an AWS Redshift database for scalable storage and fast queries.
        - A dedicated Python script connects to AWS Redshift to retrieve the transformed data and performs further calculations—such as computing the percentage of global CO2 emissions.
        - Using Matplotlib and Seaborn, the script creates a graph with a title, axis labels, and data plots. The resulting graph image is then saved on the server.
    - **Data Delivery:**
        - The server exposes a REST API endpoint that, via the Node.js child_process module, triggers the Python script to update the graph image.
        - The React frontend uses a useState hook to dynamically display the latest graph image received from the API.

Data Transformation Workflow:

- The Python ETL process reads the original CSV files (containing data like “Annual CO2 emissions” and other greenhouse gas metrics), filters and merges necessary dataframes, and computes additional statistics (e.g., percentage of global CO2 emissions).
- After generating a graph with a proper title, axis labels, and data plot, the resulting image file is stored on the server and made available via a GET HTTP request—integrated with the frontend as a dynamic “img” component.

Conclusion:

This proof of concept demonstrates our capability to create a full-stack, data-intensive web application using modern technologies. It effectively integrates user registration, secure authentication, dynamic data visualization, and advanced ETL operations—all in a cohesive platform intended to impress GreenLab’s decision makers.