React-Flask GH Archive Analyzer

This application provides a visual analytics interface to the GH Archive dataset, allowing users to explore GitHub event data through various charts and tables. The frontend is built with React, showcasing data visualizations like heatmaps, event charts, and top contributor charts. The Flask backend serves the processed GH Archive data to the frontend.

Architecture

Frontend: A React application that makes API calls to the Flask backend to fetch data and displays it through interactive charts and tables.
Backend: A Flask application that processes and serves GH Archive data stored in JSON files.
Getting Started
These instructions will get you a copy of the project up and running on your local machine for development and testing purposes.

Build the Docker images:

Open the project directory and run the following command to build the Docker images for both the React frontend and Flask backend.

docker-compose up

Accessing the Application:

The React frontend will be accessible at http://localhost:3000.
Please wait until localhost:3000 completely loads it can take a min or so to render with the correct data.
The Flask backend API can be accessed directly at http://localhost:5000.

Application Components

React Frontend:

- App.js: The main component that routes to different parts of the application.
- CustomHeatmapTable.jsx: Displays a heatmap based on the data fetched from the Flask backend.
- EventChart.jsx: Shows a chart of GitHub events over time.
- Top20Chart.jsx: Renders a chart displaying the top 20 contributors based on the event data.

Flask Backend (app.py):

- Serves processed GH Archive data through RESTful endpoints to the React frontend.
- Handles data fetching, processing, and aggregation.
- Development

Backend Development:

For changes in the Flask application, update app.py or other backend files, and Docker will handle the recompilation.

API Endpoints

- **/events_per_hour**: Fetches data aggregated by hour.
- **/top_20_watch_events_hourly**: Returns the top 20 contributors based on the number of events.
- **/top_20_watch_events**: Provides a list of top 20 projects based on GitHub events.
- **/top_10_events_by_hour**: Provides a list of top 10 projects with highest number of Github events every hour

- **Future improvements**
- To improve the current state of application we can adopt a microservices architecture, separating data processing and API serving into distinct services for more efficient scaling. We can implement message queues, like RabbitMQ or Kafka, for asynchronous data processing, enhancing the application's data handling capacity without overloading the API server.
- We can also have spark hosted in a different docker container and Flask making calls to Spark to get data processing done.
- Container orchestration will be managed through Kubernetes, ensuring scalability, high availability, and smooth deployment processes. For data processing, we can consider using stream processing frameworks, such as Apache Kafka Streams or Apache Flink, for real-time data analysis, and distributed databases or data lakes for scalable data storage.
- On the frontend and backend, we can improve load times and SEO through server-side rendering (SSR) for React and will transition the backend API to GraphQL to optimize data fetching by the frontend. Additionally, introducing a database layer to further reduce rendering times.
- To maintain the health of the application and quickly address any issues, we can implement comprehensive monitoring and logging solutions using tools like NewRelic. Moreover, integrating distributed tracing and observability tools will provide deeper insights into the system's performance and behavior.