Biolyt AI: Full Stack Developer Task

Introduction

Welcome to the Biolyt AI technical assessment. This task is designed to evaluate your ability to build a full-stack web application from the ground up. You will be working with real-world data from the ClinicalTrials.gov API to create an application that allows for the analysis and visualization of clinical trial information.
This project will test your skills in front-end development, back-end development, real-time API integration, and data manipulation. We are excited to see what you build!

Project Objective

The goal is to create a full-stack web application that fetches clinical trial data directly from the ClinicalTrials.gov API and presents it to the user in a meaningful and interactive way. The application will not use a database; all data will be fetched on-the-fly.

Core Features:

1. **Real-time Data Fetching:**
    - The backend will act as a proxy, fetching data directly from the ClinicalTrials.gov API in response to requests from the front-end. You can find the API documentation here.
2. **Backend API:**
    - Create a RESTful or GraphQL API that mirrors the filtering and searching capabilities of the ClinicalTrials.gov API.
    - Your API will receive requests from the front-end, construct the appropriate query for the external ClinicalTrials.gov API, fetch the data, and then pass it back to the front-end.
    - Implement endpoints for searching and filtering trials based on criteria like condition, intervention, location, and status.
3. **Frontend User Interface:**
    - A clean and intuitive user interface to display the clinical trial data.
    - A search bar and filtering options to allow users to narrow down the trial results.
    - Display trial data in a clear, tabular format.
    - Include data visualizations (e.g., charts, graphs) to represent trends in the data. For example, a bar chart showing the number of trials per country or a pie chart showing the distribution of trial statuses.
4. **Deployment:**
    - Deploy the full-stack application to a cloud platform. We recommend using Render for its ease of use with both front-end and back-end services.

Recommended Technology Stack

You have the flexibility to choose your preferred technologies, but we recommend the following

stack:

- **Frontend:** React (with Vite or Create React App), Vue.js, or Svelte.
- **Backend:** Node.js with Express.js or NestJS, Python with Django or Flask.
- **Data Visualization:** A library like Chart.js, D3.js, or Recharts.
- **Styling:** A CSS framework like Tailwind CSS, Material-UI, or styled-components.

Step-by-Step Guide

Phase 1: Backend Development

1. **Set up your Backend Project:**
    - Initialize a new Node.js or Python project.
    - Install necessary dependencies for your chosen framework (e.g., express) and for making HTTP requests (e.g., axios, node-fetch).
2. **Connect to the ClinicalTrials.gov API:**
    - Familiarize yourself with the ClinicalTrials.gov API. A good starting point is the study_fields endpoint to understand the available data fields.
    - **Example API call:** https://clinicaltrials.gov/api/query/study_fields?expr=diabetes&fields=NCTId,BriefTitle,Condition,InterventionName,OverallStatus,LocationCountry&min_rnk=1&max_rnk=100&fmt=json
3. **Create your API Endpoints:**
    - Implement an endpoint to get trials (e.g., /api/trials). This endpoint should, in turn, call the ClinicalTrials.gov API.
    - Implement logic to pass query parameters from your API to the external API. For example, a request to your /api/trials?status=Recruiting&condition=Asthma should trigger a correctly formatted request to the ClinicalTrials.gov API.
    - Implement an endpoint that provides aggregated data for your visualizations. You will need to fetch a broader set of data and then perform the aggregation on your backend before sending it to the client.

Phase 2: Frontend Development

1. **Set up your Frontend Project:**
    - Use a tool like Vite or Create React App to bootstrap your front-end application.
    - Install your chosen styling and data visualization libraries.
2. **Build the UI Components:**
    - Create a TrialTable component to display the list of trials.
    - Create a SearchBar component.
    - Create Filter components (e.g., dropdowns for status, text inputs for conditions).
    - Create Chart components for your data visualizations.
3. **Connect to your Backend API:**
    - Use fetch or a library like axios to make requests to your back-end API.
    - Fetch and display an initial list of trials when the application loads.
    - Implement the logic for the search and filter components to re-fetch data from your API based on user input.
4. **Implement Data Visualizations:**

- Use your charting library to create visualizations based on the aggregated data from your back-end.
- Ensure the charts are interactive and update when filters are applied.

Phase 3: Deployment

1. **Prepare for Deployment:**
   - Ensure your project is in a Git repository (e.g., on GitHub).
   - Create a production build of your front-end application.
2. **Deploy to Render:**
   - Sign up for a free account on Render.
   - **Deploy the Backend:** Create a new "Web Service" on Render and connect it to your GitHub repository. Configure the build command (e.g., npm install) and the start command (e.g., npm start).
   - **Deploy the Frontend:** Create a new "Static Site" on Render and connect it to your GitHub repository. Configure the build command (e.g., npm run build) and the publish directory (e.g., dist).
   - **Connect Everything:** Configure your front-end to make API requests to your deployed back-end URL.

Evaluation Criteria

We will be evaluating your submission based on the following criteria:

- **Functionality:** Does the application meet all the core feature requirements? Do the search, filtering, and visualizations work as expected?
- **Code Quality:** Is the code well-structured, readable, and maintainable? Have you followed best practices for your chosen frameworks?
- **UI/UX:** Is the user interface clean, intuitive, and responsive?
- **Problem Solving:** How did you approach the task? Did you make good technical decisions?
- **Deployment:** Is the application successfully deployed and accessible via a public URL?

Submission

Please submit the following:

1. A link to your public GitHub repository containing the source code for both the front-end and back-end.
2. A link to the live, deployed application on Render.
3. A brief README.md in your repository that explains your technical choices and provides instructions on how to run the project locally.

Good luck, and we look forward to seeing your work!