1. **Objectives for Your Project (for Portfolio)**

Here are some achievable and meaningful objectives for the **GitHub Repository Analyzer + Contributor Insights Engine**:

* ✅ Analyze activity and trends in any public GitHub repository.
* ✅ Extract contributor statistics: who contributes the most, how often, and what kind of work they do (code, issues, docs, etc.).
* ✅ Visualize repo data: commits over time, pull request trends, open/closed issues, etc.
* ✅ Score or rank contributors based on custom metrics (e.g., frequency, impact, diversity of contribution).
* ✅ Provide real-time or recent updates of repo activity.
* ✅ Allow the user to input a GitHub repo URL and get back insights.
* ✅ Learn how to use and authenticate with the GitHub REST or GraphQL API.

**2. Simple Explanation of the Project (For Anyone to Understand)**

This project is about exploring and understanding any public GitHub repository by pulling in data from GitHub. You’ll build a tool that fetches and analyzes useful information like how active the repo is, who the top contributors are, and what kind of work they’re doing. Think of it like creating a mini dashboard or engine that gives insights into the health and activity of open source projects.

**Project Plan (Step-by-Step with a Goal)**

**🎯 Goal**: Build a web-based tool that shows detailed insights for any GitHub repo the user enters.

Here’s a practical plan:

**🔹 Phase 1: Set the Foundation**

* Learn how to use the GitHub API (REST or GraphQL).
* Understand basic GitHub data: commits, issues, PRs, contributors.

**🔹 Phase 2: Build the Backend**

* Use Python (e.g., Flask or FastAPI) or Node.js to access GitHub API.
* Add authentication using a personal GitHub token.
* Write functions to fetch:
  + Commits, PRs, Issues.
  + Contributors & their contributions.

**🔹 Phase 3: Process & Analyze the Data**

* Calculate trends (commits per week, most active contributors).
* Identify contributor roles (e.g., coder, issue resolver).
* Store data in memory or a small database (e.g., SQLite).

**🔹 Phase 4: Build a Simple Frontend**

* Show charts (with Chart.js or Recharts).
* Let users input a repo URL and get insights.

**🔹 Phase 5: Polish & Deploy**

* Add error handling (e.g., bad URLs).
* Deploy to Render or Vercel.
* Write a README and add to your portfolio.

**4. How to Start the Project (With Example Repo)**

**✅ Step 1: Pick a Real Active Repo**

Let’s choose the repo: [facebook/react](https://github.com/facebook/react)  
It’s very active and widely used in the dev community.

**✅ Step 2: Access GitHub API for This Repo**

Start with the GitHub REST API (easiest to test and understand).

Use endpoints like:

* Repo info: GET https://api.github.com/repos/facebook/react
* Contributors: GET https://api.github.com/repos/facebook/react/contributors
* Commits: GET https://api.github.com/repos/facebook/react/commits
* Issues: GET https://api.github.com/repos/facebook/react/issues

**✅ Step 3: Use Your GitHub Account to Access Metadata**

Create a **Personal Access Token (PAT)**:

1. Go to GitHub > Settings > Developer Settings > Tokens (Classic)
2. Generate a token with repo, read:org, and read:user scopes (you only need public\_repo access)

**✅Step 4: Play With It**

* Try different endpoints.
* Print the JSON responses.
* See what data is available (name, commits, issues, etc.).
* Save responses to files for later analysis.