GITXPLORE: A FULL-STACK GITHUB PROFILE EXPLORER

A deep dive into a MERN-stack application with OAuth and API integration.

Name : Aman Prakash

Email : 22f3001090@ds.study.iitm.ac.in

Introduction

This project is a practical application of Full Stack Development and Programming concepts, combining a frontend, backend, database, and third-party authentication.

The Problem:

- GitHub is a vast platform for developers, but discovering interesting profiles and repositories can be cumbersome.
- The native interface lacks streamlined social features or custom sorting.

The Solution:

- I built GitXplore, a full-stack web app to explore and interact with GitHub profiles in a more social and dynamic way.

Background & Technology Stack

Core Concepts:

- MERN Stack: A JavaScript-based stack for building full-stack applications.
- OAuth 2.0: A security standard for delegated authorization. It lets users log in with their existing GitHub account.
- REST API: The "language" our frontend, backend, and the external GitHub API use to communicate.

Technology Stack:

- Frontend: React, Tailwind CSS
- Backend: Node.js, Express.js
- Database: MongoDB
- Authentication: Passport.js

The Authentication Flow (OAuth)

- **Goal**: Securely log in users without storing their passwords.
- **Implementation**: Used Passport.js with the GitHub "strategy".
- **How it Works (Simplified):**
  1. User clicks "Login with GitHub."
  2. They are redirected to GitHub to approve the app.
  3. GitHub sends a unique code back to our Node.js server.
  4. Passport.js exchanges this code for a user profile and access token.
  5. The user is logged in, and their session is created.

The Core Feature (GitHub API)

- **Goal**: Fetch and display real-time data from GitHub.
- **Implementation**:
  1. Created backend (Express.js) routes that act as a secure "middleman."
  2. These routes use the authenticated user' s token to fetch data from the official GitHub API.
  3. This proxy approach protects our API keys and logic from being exposed on the frontend.
- **Data Fetched**: User profiles, repository lists, and repository data by programming language

Dynamic Sorting & Filtering

- **Goal**: Make the data interactive and useful.
- **Feature**: Implemented dynamic sorting for repositories.
- **How it Works (Frontend to Backend):**

  1. Frontend (React): User selects a filter (e.g., "Most Stars").
  2. API Call: The frontend re-fetches data from our backend with a query parameter.
  3. Backend (Node.js): The server logic modifies its call to the GitHub API to request the data in the specified order.

The Social Feature (A Full-Stack "Like")

- **Goal**: Add a social interaction feature, a key differentiator from plain GitHub.
- **This feature demonstrates the entire MERN stack working together:**

1. Frontend (React): A user clicks the "Like" button on another user's profile.
2. API Call: React sends a POST request to our backend.
3. Backend (Express/Node): The server (a) verifies the user is logged in and (b) finds the correct user document.
4. Database (MongoDB): The server updates the liked user's profile in our database, adding the "likedBy" ID to an array.

UI & Responsiveness

- **Goal**: Ensure the application is clean, modern, and usable on all devices.
- **Technology**: Used Tailwind CSS for utility-first styling.
- **Key UI Features:**

1. Fully responsive design that adapts from desktop to mobile.
2. Clean, card-based layout for repositories and user profiles.
3. Intuitive navigation for authenticated users to find popular repos by language.

Results & Key Challenges

- **Key Results:**

1. A fully functional and deployed full-stack (MERN) application.
2. Successful integration of a complex third-party OAuth flow.
3. A dynamic and interactive UI for exploring GitHub data.

- **Challenges Faced:**

1. GitHub API Rate Limiting: Learning to handle API limits by making efficient, conditional, or cached requests.
2. OAuth Configuration: Managing callback URLs for both development (localhost) and the deployed production site.
3. State Management: Keeping the user's authentication status and fetched data consistent across all React components.

Conclusion

- **Summary:** GitXplore successfully integrates a MERN stack, a third-party API, and secure authentication to create a useful and social developer tool.

- **Future Scope:**

  1. Cache GitHub API responses in our database to improve performance.
  2. Expand the social features (e.g., commenting on profiles).
  3. Add more detailed repository analytics.

Thank You