**Comprehensive documentation: RG Quiz App - A Resource-constrained computing prototype**

## **Project Title: RG Quiz and Reinforced Learning System**

Name: Gbenga Joshua Afolabi

Live Application: https://spectacular-begonia-a0be97.netlify.app/

GitHub Repository: https://github.com/joshuaafolabi80/africa\_deep\_tech\_challenge

1. **Problem definition and identified constraints**

The RG Quiz App addresses critical challenges hindering effective education for children in Africa, particularly in resource-constrained environments. I identified the following core problems and constraints:

* **Learning isn't fun or engaging for children:** Traditional learning methods often fail to capture children's attention, leading to disinterest and poor retention.
* **Concepts are hard to grasp without personalized help:** Students frequently struggle with complex topics, lacking access to tailored explanations that cater to their individual understanding levels.
* **Parents struggle to track progress and reward learning:** Parents often lack effective tools to monitor their children's educational progress and provide timely encouragement or rewards.
* **Need for curriculum-tailored & customizable applications:** There is a significant need for educational applications specifically designed to align with the Nigerian school curriculum, with the flexibility to be customized for individual schools or learning paths.

**Resource-constrained computing environment**

* Limited Power/Low Compute: Devices may rely on batteries, solar power, or unstable grids. Applications must minimize energy consumption and perform smoothly on minimal hardware (e.g., low-cost tablets).
* Offline Functionality: Intermittent or complete lack of internet connectivity is a major barrier. Learning must be able to continue uninterrupted without an active internet connection.
* Affordability of Internet: Even where available, the cost of mobile data can be prohibitive for many families.

1. **Alternative design explorations and final decisions**

My design process focused on building a robust, accessible and intelligent learning system that directly counters the identified problems and constraints.

**Progressive web app (PWA) as core architecture:**

* **Exploration:** I considered native mobile apps (Android/iOS) and traditional web applications.
* **Decision:** I opted for a PWA.
* **Rationale:** PWAs offer the best of both worlds – web accessibility (no app store download required) combined with native app-like features (installability to home screen, offline functionality). This directly addresses the internet affordability and accessibility constraints, ensuring continuous learning even without connectivity. It also allows for a smaller initial download size compared to many native apps.

**Frontend-centric architecture for offline capability:**

* **Exploration:** Given the challenge's focus on resource-constrained environments, I prioritized solutions that did not rely on constant internet connectivity for core features.
* **Decision:** I designed the core learning experience, including AI explanations, to be frontend-centric and fully offline.
* **Rationale:** This decision is paramount for adhering to "resource-constrained computing principles." By making AI explanations pre-built and local, I eliminated the need for constant internet connectivity, saving data costs and ensuring functionality in low-connectivity areas.

**Two-layer explanation system (basic + pre-built AI-powered deep explanation):**

* **Exploration:** I considered providing only basic pre-defined explanations. However, to offer personalized help without requiring live internet, I decided to integrate pre-generated content.
* **Decision:** I implemented a two-tiered system for explanations:

1. Instant Basic Explanation: Quick feedback from pre-defined text (currentQuestion.explanation).
2. Pre-built AI-Powered Deep Explanation: Pre-generated, detailed, and age-appropriate explanations (currentQuestion.deepExplanation) are embedded directly within the local all\_quiz\_data.json file.

* **Rationale:** This approach directly addresses the problem of "Concepts are hard to grasp without personalized help" while strictly adhering to the "offline functionality" constraint. The "AI Tutor" persona and tailored language are pre-computed and stored locally, providing a personalized learning aid without requiring an active internet connection or consuming live API credits during use.

**Curriculum alignment and customization:**

* **Exploration:** I focused on creating content that was directly relevant to the target audience.
* **Decision:** I designed the quiz data (all\_quiz\_data.json) to be structured by Class, Term, Subject, and Topic, directly aligning with the Nigerian curriculum. The data is easily extensible.
* **Rationale:** This directly addresses the need for an application tailored to the Nigerian curriculum and provides a clear pathway for customization and expansion for different schools or regions.

**User interface (UI) and user experience (UX):**

* **Exploration:** I aimed for a design that was both functional and appealing to children.
* **Decision:** I utilized Bootstrap 5 for a responsive, clean, and intuitive design. I incorporated engaging sound effects and visual feedback (celebrations, empathetic messages).
* **Rationale:** A user-friendly and engaging interface is crucial for making learning fun for children. Responsiveness ensures the app works seamlessly on low-cost tablets and smartphones, maximizing reach.

1. **Tools and technologies used**

The RG Quiz App leverages a modern and efficient technology stack, prioritizing frontend capabilities for offline functionality:

**Frontend Technologies:**

* **React:** JavaScript library for building dynamic user interfaces.
* **Vite:** A next-generation frontend tooling that provides an extremely fast development experience and optimized builds.
* **React Router DOM:** For client-side routing and navigation within the single-page application.
* **React Bootstrap:** A popular UI framework for responsive and pre-styled components, ensuring a clean and mobile-friendly interface.
* **Framer Motion:** A production-ready motion library for React, used to create smooth and engaging animations (e.g., celebration effects).
* **Howler.js:** A robust JavaScript audio library for handling sound effects and background music, enhancing user engagement.
* **VitePWA:** A Vite plugin that simplifies the creation of Progressive Web Apps, enabling offline capabilities and installability.
* **React Markdown & Remark GFM:** Libraries used to parse and render Markdown content (from the pre-built explanations) into HTML, allowing for rich, formatted explanations.

**Deployment & version control:**

* **Netlify:** A cloud platform optimized for deploying frontend frameworks and static sites. It handles the build process and provides global CDN for fast access.
* **GitHub:** Used for version control, collaboration, and hosting the project's source code.

1. **Technical implementation details and performance benchmarks**

**Project architecture (Frontend Only)**

The project is a pure frontend application, designed for maximum efficiency and offline capability. There is no active backend component for the core learning experience in the deployed application.

**PWA implementation for resource constraint (Offline & Low Compute)**

1. **VitePWA Plugin:** I configured the vite-plugin-pwa in frontend/vite.config.js to generate a Service Worker. This Service Worker is responsible for:

* **Offline Caching:** Caching all essential static assets (HTML, CSS, JS, images, audio, quiz data including pre-built AI explanations) using Workbox strategies (CacheFirst, StaleWhileRevalidate). This allows the application to function fully even when the device has no internet connection, directly addressing the "offline functionality" constraint.
* **Installability:** Enabling users to "Add to Home Screen" on mobile devices or "Install" on desktops, providing an app-like experience without app store downloads.

1. **Lightweight Footprint:** The application is designed to be ultra-light and energy-efficient. At its prototype stage, the estimated size is 46MB. This small footprint is crucial for running smoothly on low-cost tablets and devices with limited storage and processing power, aligning with "low compute" principles.
2. **Responsive Design:** Built with Bootstrap 5, the frontend dynamically adjusts its layout and components to fit various screen sizes (phones, tablets, desktops), ensuring optimal usability on diverse devices.

**Pre-built AI integration for personalized learning (Offline)**

**Data Source:** The core quiz data, including both basic explanations (explanation) and the pre-built AI-powered deep explanations (deepExplanation), is stored directly within the frontend/src/components/all\_quiz\_data.json file.

**Frontend (frontend/src/components/QuizPage.jsx):**

* When a user clicks "Still not clear? Get deeper explanation", the explanationLevel state is updated to 2.
* The component then directly accesses currentQuestion.deepExplanation (or currentQuestion.explanation as a fallback) from the locally loaded all\_quiz\_data.json.
* The TypingEffect component is used to simulate a dynamic AI response, displaying this pre-built text with an engaging typing animation.
* The deepExplanation text is rendered using ReactMarkdown to ensure proper display of any Markdown formatting (e.g., bolding) present in the pre-built explanations.

**Offline Advantage:** This implementation ensures that the "AI Tutor" feature is fully functional without any internet connection, making it a truly offline-ready digital tool.

**Curriculum Alignment & Customization**

* The quiz content is loaded from frontend/src/components/all\_quiz\_data.json. This JSON file is structured hierarchically by classId, termId, subjectId, and topicId, making it easy to manage and expand content that directly maps to the Nigerian school curriculum.
* This modular data structure allows for easy customization by schools or educators who might want to swap out or add their own sets of questions and topics.

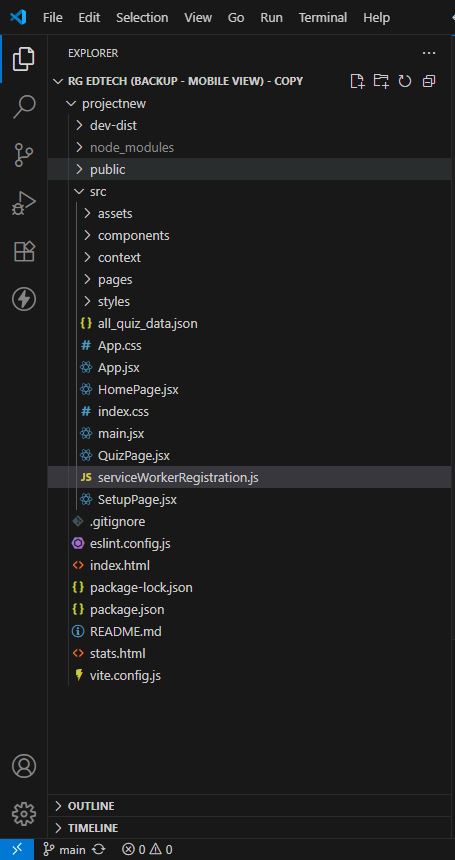
**Performance Benchmarks (Descriptive)**

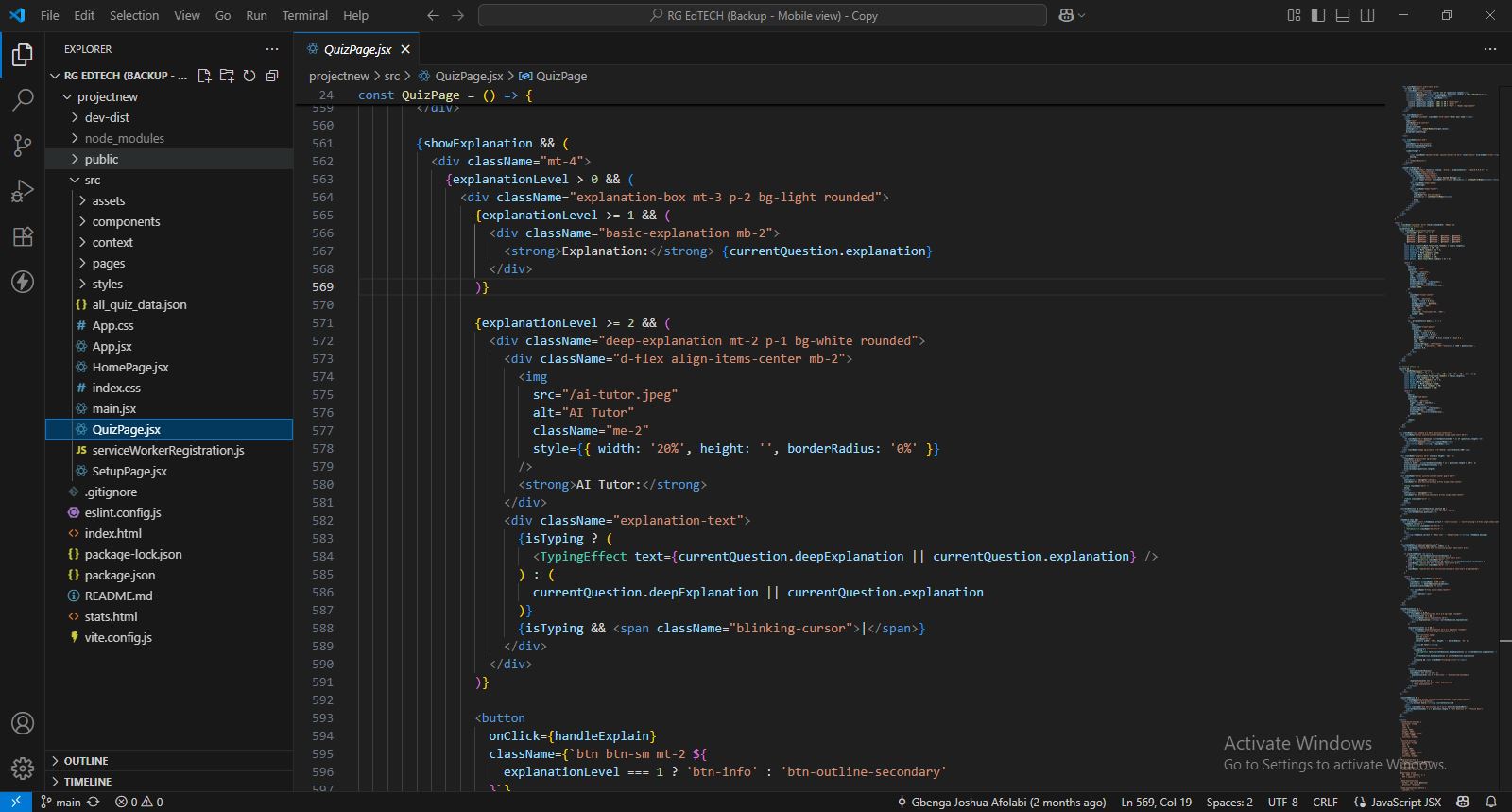
While specific numerical benchmarks (e.g., load times in milliseconds) are not provided, the design choices inherently lead to high performance:

* **Fast Load Times:** Vite's optimized build process and PWA caching ensure that subsequent loads are near-instantaneous once the app is installed or cached.
* **Energy Efficiency:** The lightweight nature (46MB) and client-side processing for most UI interactions minimize CPU and battery usage, crucial for devices with limited power. The elimination of network calls for AI explanations further reduces power consumption.
* **Smooth Operation:** The use of React for efficient UI updates and optimized state management ensures a smooth user experience, even on low-spec hardware.

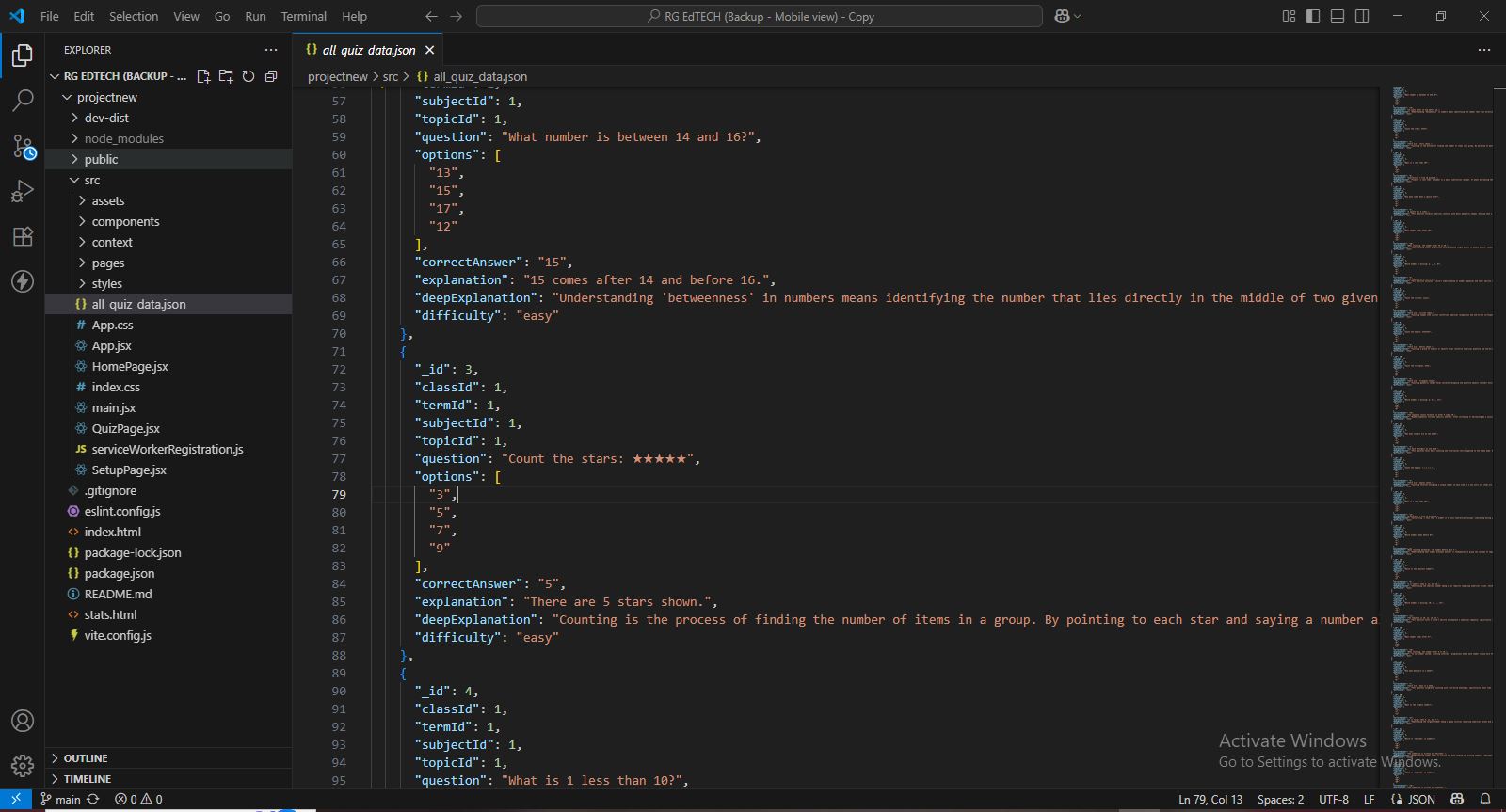
1. **Screenshots of the development process (Description)**

**Project structure:**

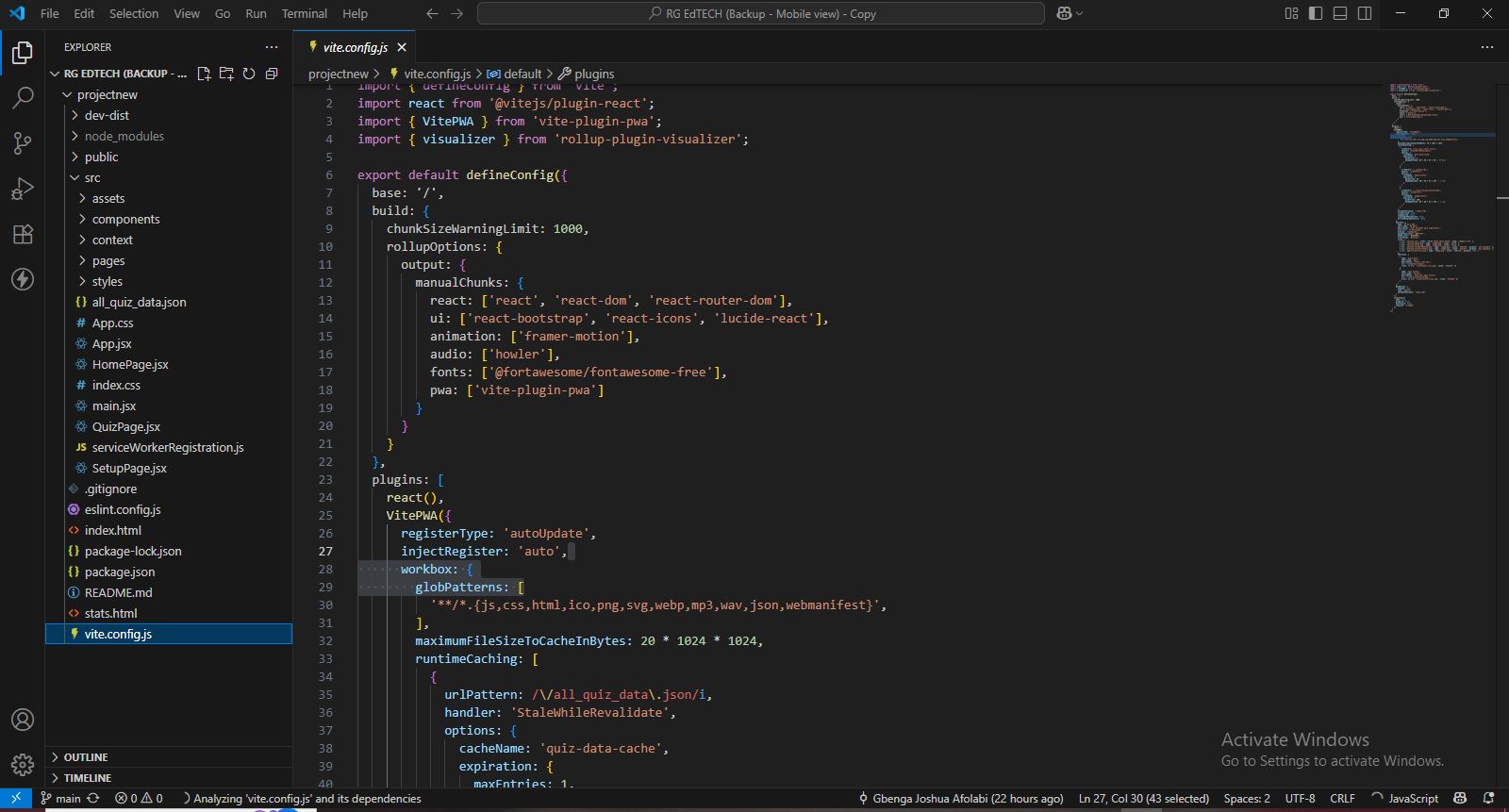
**frontend/src/components/QuizPage.jsx:**



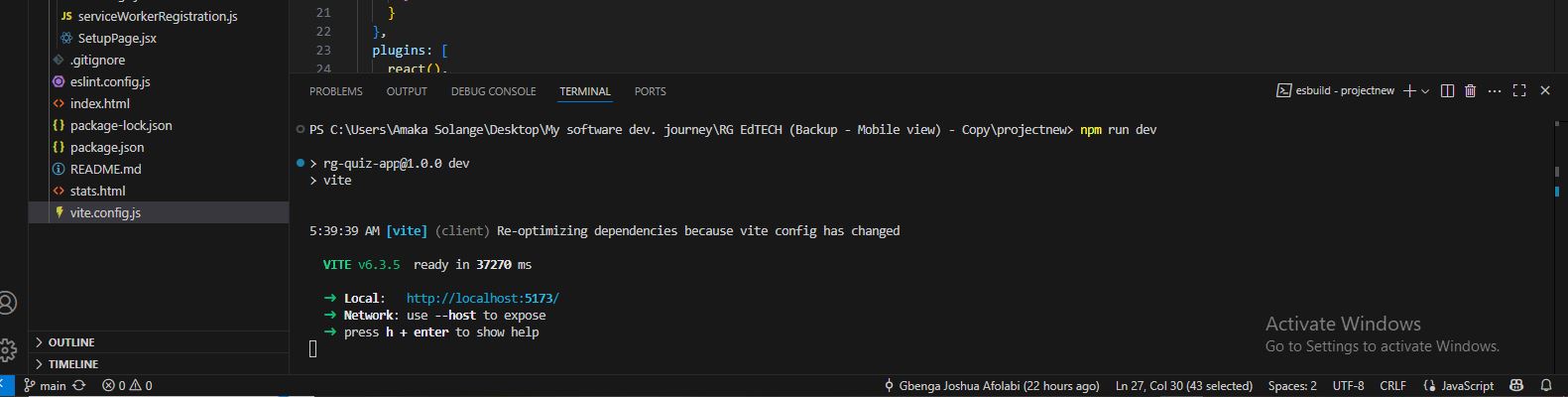
**frontend/src/components/all\_quiz\_data.json:**



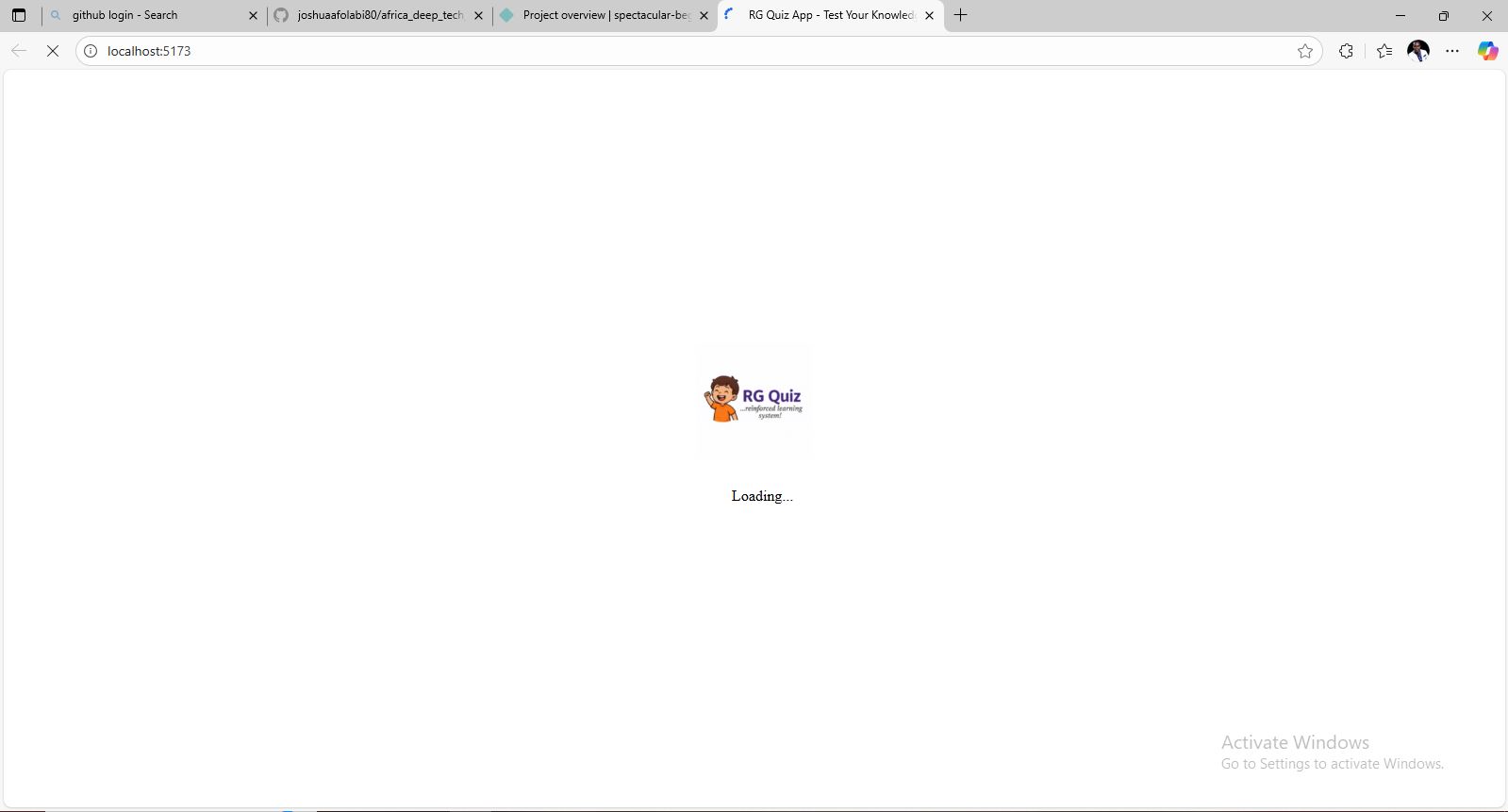
**frontend/vite.config.js:**



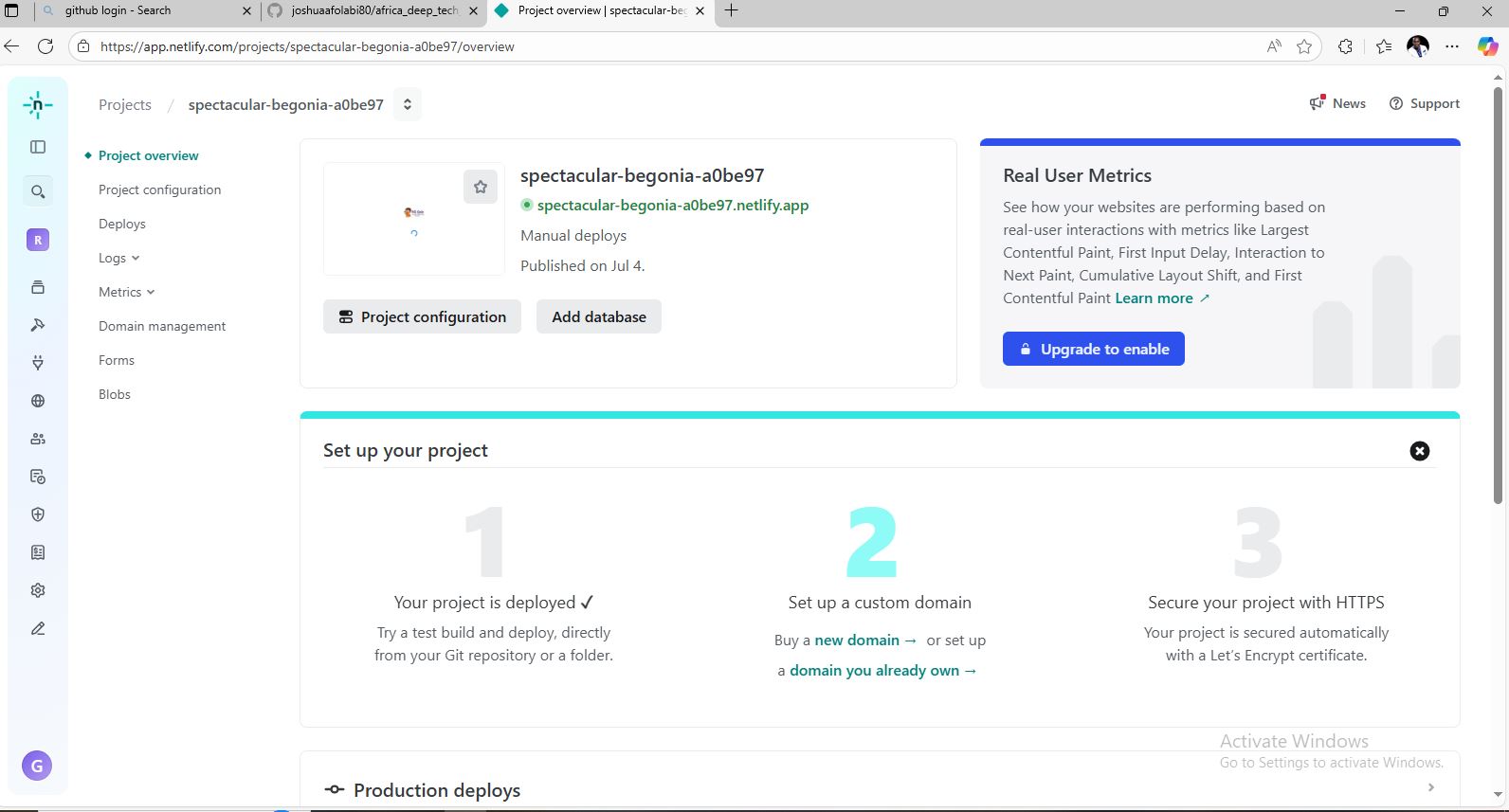
**Local development (npm run dev):**

****

**Application running in a browser:**

****

**Netlify deployment:**

****