Project Documentation: Code Snippets Manager

1. Technologies Used

Frontend

- **React.js (v19)**: A JavaScript library for building user interfaces. It allows for the creation of reusable UI components and manages the state of the application efficiently.
- **Vite**: A modern build tool that provides a faster and leaner development experience for modern web projects. It offers instant server start and lightning-fast Hot Module Replacement (HMR).
- **Tailwind CSS (v4)**: A utility-first CSS framework for rapidly building custom user interfaces. It allows for styling directly in the JSX markup.
- **Google OAuth (@react-oauth/google)**: Used for secure user authentication via Google accounts.
- **Lottie Files (@lottiefiles/dotlottie-react)**: For rendering lightweight, scalable animations (like the loading spinner).
- **React Syntax Highlighter**: For displaying code snippets with syntax highlighting, making them readable.

Backend

- **Node.js**: A JavaScript runtime built on Chrome's V8 JavaScript engine, allowing execution of JavaScript on the server side.
- **Express.js**: A minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.
- **MySQL (mysql2)**: A relational database management system used to store user data, folders, and snippets.
- **JWT (jsonwebtoken)**: Used for securely transmitting information between parties as a JSON object. In this project, it's used for user authentication and session management.

2. Backend Details (Express & Dependencies)

The backend is built using **Express.js**, which simplifies the process of handling HTTP requests, routing, and middleware integration.

Key Dependencies & Their Use:

- **express**: The core framework. It handles routing (e.g., GET /api/data, POST /api/folders) and middleware management.
- **cors**: Middleware to enable **Cross-Origin Resource Sharing**. Since our frontend runs on port 5173 and backend on 5000, browsers would normally block requests between them for security. cors allows the frontend to communicate with the backend.
- **mysql2**: A MySQL client for Node.js. It is used to connect to the MySQL database and execute SQL queries (e.g., SELECT, INSERT, UPDATE) to manage data.
- **jsonwebtoken (JWT)**: Used to create access tokens. When a user logs in via Google, the backend generates a signed JWT containing the user's ID. This token is sent to the frontend and included in subsequent requests to prove the user's identity (authenticate Token middleware)
- **google-auth-library**: verify the Google ID token sent from the frontend during the login process, ensuring the user is legitimately authenticated with Google.
- **dotenv**: Loads environment variables from a .env file into process.env. This keeps sensitive data like database passwords and API keys secure and out of the codebase.
- **bcryptjs**: (Included but primarily for password hashing if traditional email/password login were implemented. With Google Auth, it's less critical but good to have for future expansion).
- **express-rate-limit**: A security middleware that limits the number of requests a user can make to the API in a given timeframe, protecting against brute-force attacks or abuse.

3. Frontend Details (React.js & Optimization)

The frontend is a Single Page Application (SPA) built with **React**.

Key Dependencies:

- **react & react-dom**: The core libraries for rendering components and managing the DOM.
- **react-syntax-highlighter**: Renders code blocks with colors and formatting similar to VS Code.
- **@react-oauth/google**: Provides the GoogleLogin component and hooks to handle the OAuth flow easily.

React Hooks Used & Why:

- **useState**: Used to manage local component state.
    - *Example*: const [searchTerm, setSearchTerm] = useState('') in HomePage.jsx tracks what the user is typing in the search bar.
- **useEffect**: Used for side effects, such as fetching data or setting up event listeners.
    - *Example*: Fetching user data from the API when the component mounts, or adding a scroll event listener in Navbar.jsx.
- **useMemo**: Used for performance optimization. It "rememoizes" (caches) the result of a calculation so it's not re-computed on every render unless dependencies change.
    - *Example*: Filtering allSnippets based on searchTerm. We don't want to re-filter the entire list just because a button was clicked; only when the search term or data changes.
- **useCallback**: (Implicitly used in some patterns or good for future use) Memoizes functions to prevent them from being recreated on every render, which is crucial when passing functions to child components wrapped in React.memo.

Code Optimization Techniques:

- **React.memo**: We wrapped components like SnippetCard, Folder, and CodeSyntaxHighlighter in React.memo. This tells React: "If the props (data) passed to this component haven't changed, do not re-render it." This significantly reduces lag when scrolling through large lists.
- **Optimistic UI Updates**: In App.jsx, when creating a folder or snippet, we update the UI *immediately* with a temporary ID before the server responds. This makes the app feel instant to the user.
- **CSS Optimization**: We used will-change and transform: translateZ(0) (via the gpu-accelerated class) to offload rendering of complex animations and glassmorphism effects to the GPU, preventing main-thread jank.

---

4. Architecture & Connection

Port Numbers

- **Frontend (Port 5173)**: This is the default port for **Vite**. It's an arbitrary choice by the tool but has become a standard for modern React development.
- **Backend (Port 5000)**: A common convention for Node/Express servers. It keeps it distinct from the frontend port.

Establishing Connection

1. **CORS**: As mentioned, the backend uses the cors middleware to explicitly allow requests from http://localhost:5173.
2. **API Calls**: The frontend makes HTTP requests (GET, POST, PUT, DELETE) to http://localhost:5000/api/....
3. **Proxy (Alternative)**: In production or some dev setups, a proxy can be configured in vite.config.js so the frontend thinks it's calling its own domain, avoiding CORS
4. issues, but here we used explicit CORS for clarity.

---

5. Project Explanation (From Scratch)

1. Setup & Foundation:

- We started by initializing a **Node.js** project for the backend and a **Vite+React** project for the frontend.
- We set up a **MySQL** database to store our structured data (Users -> Folders -> Snippets).

2. Backend Development:

- We built an **Express server**.
- We created **API endpoints**:
    - /auth/google: To handle login and issue JWTs.
    - /api/data: To fetch the entire tree of folders and snippets for a user.
    - /api/folders & /api/snippets: To create, update, and delete items.
- We secured these endpoints using a middleware (authenticateToken) that checks for a valid JWT.

3. Frontend Development:

- We designed a modern, dark-themed UI using **Tailwind CSS**.
- We built reusable components: Navbar, SnippetCard, Folder, SnippetViewModal.
- We implemented **State Management** in App.jsx to hold the master userData.
- We connected the frontend to the backend using fetch.

4. Refinement & Optimization:

- We added **Google Auth** for easy login.
- We implemented **"Glassmorphism"** (blur effects) for a premium look.
- We noticed performance issues (lag) with many snippets, so we optimized using React.memo, simplified CSS shadows, and added GPU acceleration.
- We added **Optimistic Updates** so creating items feels instant.
- Finally, we added polish like the **Copy Feedback** and **Logout Confirmation**.

The result is a full-stack, secure, and highly performant application for managing code snippets.