INTERNSHIP PROJECT DOCUMENT
ON “CODE EDITOR”

Submitted by
(Intern)
Sai Kiran Annoju

Submitted to
Founder - Kanduri Abhinay
CEO & CTO - Rithin Varma

INTRODUCTION:

A Code Editor is an essential tool for developers, enabling them to write, edit, and debug code efficiently. This project focuses on building a web-based code editor that supports real-time editing and previewing of HTML, CSS, and JavaScript code. The editor provides a user-friendly interface with features like dark mode, auto-run, fullscreen mode, and the ability to open the preview in a new tab. The project demonstrates the use of modern web technologies like application. The primary goal is to provide developers with a lightweight, browser-based editor for quick prototyping, testing, and learning. The editor emphasizes a clean, modern UI and real-time rendering of code output. where the usern

---

SOFTWARE DEVELOPMENT LIFE CYCLE(SDLC)

The project followed the SDLC framework to ensure systematic development:

1. **Planning**
    - **Objective**: Develop a web-based code editor with real-time preview, dark mode, and cross-platform compatibility.
    - **Scope**: Support HTML, CSS, and JavaScript; integrate UI features like auto-run and fullscreen mode.
    - **Feasibility**: Leverage React for component-based architecture and localStorage for persistence.
2. **Defining Requirements**
    - **Software Requirements**:
        - **OS**: Windows, macOS.
        - **Languages**: React, JavaScript, HTML/CSS.
        - **Tools**: Visual Studio Code.
        - **Libraries**: React, lucide-react (icons).
    - **Hardware Requirements**: Modern web browser with JavaScript support.
3. **Designing**
    - **UI Structure**:
        - Navbar (logo, theme toggle, auto-run, fullscreen).
        - Editor panels (HTML, CSS, JS tabs).
        - Preview panel (iframe for live output).
    - **State Management**: React hooks (useState, useEffect) for code, theme, and preferences.
    - **Code Execution**: runCode function to combine code into an iframe.
    - **Persistence**: localStorage to save code and user settings.
4. **Building (Implementation)**
    - Developed using React functional components and hooks.
    - Integrated lucide-react for icons.

- Implemented features:
  - Dark mode via CSS variables.
  - Auto-run with setTimeout delay.
  - Fullscreen mode using the Fullscreen API.
  - Code clearing and preview in new tab.

5. **Testing**
   - Unit Testing: Validated state updates, localStorage persistence.
   - Edge Cases: Tested invalid code inputs, empty fields, and theme toggling.
   - Performance: Ensured real-time updates with minimal lag.
6. **Deployment**
   - Deployed as a standalone web application.
   - **Future Deployment Plans**:
     - Mobile app development.
     - Integration with cloud-based collaboration tools.
7. **Conclusion**

   Following SDLC ensured a structured approach, resulting in a robust, user-friendly editor with scalability for future enhancements.

PROCEDURE AND METHODS USED:

1. **Application Workflow:**
   - **Input: User writes code in HTML, CSS, or JS editors.**
   - **State Management:** React hooks track code changes and UI preferences**.**
   - **Code Execution:**
     - Combined code injected into an iframe via srcdoc.
     - Auto-run triggers updates with a 1-second delay.
   - **Persistence**: localStorage saves code and settings on exit.
2. **UI Features:**
   - **Dark Mode:** Toggles CSS variables for theme switching**.**
   - **Fullscreen Mode:** Utilizes the Fullscreen API for immersive editing**.**
   - **Preview:** Real-time output in iframe or new tab.

ALGORITHM:

1. **Initialize Application**:
    - Load code and preferences from localStorage.
    - Set initial states for editors, theme, and auto-run.
2. **Render UI**:
    - Display navbar, editor panels, and preview.
    - Update active tab based on user selection.
3. **Handle Code Changes**:
    - On user input, update corresponding state (HTML/CSS/JS).
    - Trigger runCode if auto-run is enabled.
4. **Execute Code**:
    - Combine HTML, CSS, and JS into a single string.
    - Inject into iframe using srcdoc.
5. **Toggle Features**:
    - Update CSS variables for dark mode.
    - Toggle fullscreen via Fullscreen API.

FUTURE SCOPE:

1. **Syntax Highlighting:** Integrate libraries like react-syntax-highlighter**.**
2. **Additional Languages**: Support TypeScript, Python, or Markdown**.**
3. **Export/Share:** Enable code download or sharing via URL.
4. **Collaboration:** Implement real-time collaboration using WebSockets**.**

FUTURE ENHANCEMENTS:

1. **Mobile App:** Develop a cross-platform mobile version**.**
2. **AI Integration:** Add code suggestions using OpenAI Codex**.**
3. **Code Formatting:** Integrate Prettier for auto-formatting**.**
4. **Error Checking:** Implement linting for JavaScript**.**

ADVANTAGES:

1. **Real-Time** Preview: Instant feedback for rapid prototyping**.**
2. **Cross-Platform:** Runs on any modern browser**.**
3. **Lightweight**: No backend dependencies**.**
4. **User-Friendly**: Intuitive UI with essential features**.**

REFERENCES:

1. **React Documentation: https://reactjs.org/**
2. **lucide-react Icons: https://lucide.dev/**
3. **Fullscreen API: MDN Web Docs**
4. **localStorage: MDN Web Docs**
