Collaborative Code Editor (CCE)

Mohamed Sharif, Justice Gauldin, Vincent Nguyen, Asrar Syed
Department of Computer Science, Georgia State University
Course: CSC 4311 - Cloud Computing
Professor: Dr. Lipeng Wan
Due Date: April 28, 2025

1. Introduction

For our project, we wanted to create a cloud-based code editor. We aimed to build a full-stack web application where users could log in, create documents, edit documents, compile their code, and collaborate in real-time. We wanted it to be simple and functional for solo programmers and groups of programmers working together.

The project combines frontend UX/UI, backend development skills, cloud database management, and real-time collaboration tools. Throughout the project, we made improvements based on experimental challenges, testing, and continuous refinement of our ideas.

At the beginning of the project, we focused on identifying what features would matter most to a typical user. We realized that just being able to edit and save code wasn't enough. Creating an online experience where collaboration felt natural mattered to us. That’s why we built real-time document sharing, manual version control, and an invite system. Every decision, from our backend to the simple dashboard layout, was made to keep the experience straightforward and user-friendly.

Another important factor we considered early was the balance between functionality and complexity. While it's tempting to add every possible feature into an app like this, we chose to prioritize reliability. We wanted a platform that worked well across devices and didn't overwhelm users with unnecessary options. This mindset kept the project achievable within our timeline, while still leaving room for future enhancements.

Overall, this project wasn't just about building a code editor. It was about connecting different skills we had learned over time. It pushed us to think about security, real-time data handling, frontend design, and backend database management as one connected system. CCE became a way for us to not only apply what we knew but also to learn things we wouldn't have picked up from our classes alone.

2. Motivation / Background

Our motivation came from the idea of building something that merged important concepts we learned across different areas, like cloud services, front-end development, real-time data updates, and basic code compilation into a single application.

We began by setting up a React application using Create-React-App and installed necessary libraries, including the Monaco Editor for code editing and Firebase for backend services (Authentication and Firestore). Firebase Authentication manages secure user logins, while Firestore handles document storage and real-time updates.

As development continued, we realized collaboration wasn't just about multiple users seeing edits. It involved properly structuring documents, managing permissions, and securely inviting collaborators to the owner's document. We made key backend design decisions early, like storing documents flatly under "/documents/{docId}" and using a simple collaboration system based on registered user emails.

We also focused on the compiler feature using Judge0 API and added manual version control for document backups. These additions helped us create a more complete and usable platform.

Looking back, a major part of our motivation was to create a tool that felt realistic. We wanted to create something people could actually use, not just a classroom demo. Some real-world workflows could include programmers working remotely, students collaborating on coding projects, or developers needing a lightweight editor without a complex setup. Building CCE gave us a chance to design a project with real-world use cases in mind, which made it more rewarding and gave it a clear purpose beyond just getting a good grade.

3. Design / Implementation

Our architecture was built around a separation between Firebase backend services and React frontend components. Here, you can view all of the important JavaScript files in order from our "src" folder, our tech stack, and our core features from the web application:

JavaScript Files

1. index.js: Entry point linking routes together.
2. LoginPage.js: Handles user authentication via Firebase.
3. AboutUs.js: Displays a description of our project.
4. DocsDashboard.js: User dashboard showing saved documents.
5. App.js: Code editor page with the Monaco Editor, compile pane, and top toolbar.
6. CollabModal.js: Collaboration invite logic.
7. sendEmail.js: Integration with EmailJS for sending invites.
8. firebaseConfig.js: Holds Firebase project configuration.

<u>Tech Stack</u>

– Frontend: React, React Router DOM, Monaco Editor, EmailJS
– Backend Services: Firebase Authentication, Firestore Database
– Compilation API: Judge0 API
– Supporting Libraries: Axios, lodash.throttle, Testing Libraries (Jest, Testing-Library React)
– Deployment: Firebase Hosting

<u>Core Features</u>

– **Authentication and User Profiles**: Users authenticate via Firebase Authentication. Each user has a corresponding "/users/{uid}" document storing their email and user identification for collaboration.

– **Document Management**: Documents are stored under "/documents/{docId}," containing fields like owner, an array of collaborators, code (the text they're writing), language, lastEdited, and title. Users can create, edit, save, and delete documents.

– **Versioning**: Each document supports version history. Versions are stored under `/documents/{docId}/versions/`, where users can manually save or delete versions.

– **Compiler Integration**: A cloud-hosted compiler service called Judge0 allows users to compile code in C, Java, and Python. Results appear in the compiler pane. Currently, we're using its free plan, which is 50 requests a day.

– **Collaboration System**: The "collab" feature was carefully built to distinguish between "Share" and "Invite."

Share: Users can send a static snapshot of their code to any valid email address via EmailJS. The recipient will receive your shared code in a simple format without any collaboration. The point of this was so that you could send someone what you wrote via email without having to share your document.

Invite: Users can invite registered Firebase users by their email. If the email matches a user, the system sends a collaboration invite link via EmailJS. Clicking this link allows both users to be on the same editor document, allowing real-time collaboration. Both users can see changes (after manual saves) and retain access to the document in their dashboards. If the email entered does not exist, an error message appears saying "User not found." More importantly, if the entered email is valid, but is not registered in our backend system, an error message will display saying “Sorry! That email does not have an account with us :(“

– **Security Rules**: Our Firestore rules ensure that any signed-in user can create, edit, and delete their documents, while collaborators can only edit documents but not transfer ownership. Doing this gives limited accessibility for collaborators, which is what we intended.

4. Evaluation

Achievements

We successfully built a cloud-based code editor that supports document editing, secure authentication, code compilation, and a versioning system. Users can collaborate easily without needing external accounts besides a valid Gmail login. All major features, including the invite system, were tested across multiple browsers and devices.

We paid special attention to testing security rules, ensuring users could only access documents they owned or were invited to. We also tested invite edge cases (already invited users, non-existent emails) to confirm that error handling worked properly.

Testing

Throughout testing, we verified that real-time updates between collaborators functioned correctly when users saved changes. We confirmed that document access was secure, ensuring no random authenticated user could access documents they weren't authorized to view. Document creation, deletion, and the invite system were all tested. Compilation through the Judge0 API was tested across all three languages, which were Java, C, and Python. Their outputs returned correctly inside our compiler pane. However, user-input programs can’t currently function because an input box isn’t implemented yet.

5. Discussion

Reflection

Throughout the project, we encountered multiple challenges, especially around real-time collaboration and security. Originally, documents were nested inside each user's collection, but we changed to a flat document structure to make sharing easier. This change simplified query logic and security rule writing.

Real-time updates were another important learning experience. Firestore naturally supports live data syncing, but in our case, changes made by one user were not instantly visible to another in real-time editing mode. Instead, users and collaborators must manually save their progress, after which other collaborators can view the changes. While changes do occur in real-time in the background, they aren't actively rendered until a save action takes place. Careful planning and separating fields (such as code, title, and language) helped reduce merge conflicts, but real-time editing would require more time to implement.

For now, we've been discussing how to better improve this project, splitting it into significant and marginal changes:

Significant Changes

– **Autosave**: Implementing autosave would reduce the need for manual saves and improve collaboration speed.

– **Presence Indicators**: Showing the username of the collaborators when they're actively making changes.

– **Clean Up User Input**: Validating and sanitizing user input (document titles, usernames) would prevent formatting issues.

– **Folder System for OOP Languages**: Adding a folder system would be especially important for object-oriented programming, where projects often consist of multiple files. This would allow more realistic project management inside the app.

– **Create Mobile App Version**: Extending CCE into a mobile application could allow users to code and collaborate on the go.

Marginal Changes

– **Cancel Compiler Button**: Allow users to cancel a compilation request if needed.

– **Manually Resize/Move/Hide Compiler Pane**: Let users drag, collapse, and control the compile output area more freely.

– **Trash Tab in Settings**: Give users an interface to recover recently deleted documents or versions.

– **Custom Usernames**: Allow users to customize or edit their display name. This could potentially be useful when sending invites.

– **Limited Guest Access**: Let users quickly create or edit documents without full registration.

– **Add More Languages**: Expand support beyond Java, C, and Python to include C++, Go, Rust, etc.

– **Mobile Web Fixes**: Improve mobile responsiveness and touch interface support.

Conclusion

Despite these limitations, the project provides a strong foundation for a full collaborative programming tool. The system works reliably and meets the goals we set, which are providing a lightweight, online code editor that integrates secure authentication, real-time updates, compiler functionality, and document versioning.

Building CCE combined cloud services, frontend programming, backend database design, and security concepts into a single application. We successfully built a system that allows users to code, compile, save, invite, and collaborate all in the browser. Our work on CCE reflects both the technical skills we learned during the semester and the real-world problem-solving needed to launch a functional cloud-based app.