Contents

Introduction

The purpose of this document is to outline the features of the Educator Live Coding Tool and define the course of implementation for it.

Outline of Implementation

While this document will outline and define further goals and sections, it is important to note the defined expectations of the project. As outlined in the Pro Forma Course Content, the design and implementation of the Educator Live Coding Tool can be summed up as a collection of four key milestones that are outlined in course outline.

(1) *An embedded editor (similar to Jupyter Hub) that allows users to write code and run it in the browser.*
(2) *Allows the assignment of different roles within the system, allowing different users to have slightly different usable features (i.e. Instructors have more administrative tools than students).*
(3) *Allows instructors to send code from their editor into all of their students' editors, or into a particular student's editor.*
(4) *Additional features based on time remaining, [including -- but not limited to -- actions] during live coding sessions to aid with instruction.*

For the purposes of this document, we will only highlight on the first three milestones because the fourth milestone cannot be determined until actual development and estimated development times begin.

Implementation Plan

In general, the following section defines any implementation plan for the Educator Live Coding Tool. It should be highlighted that the plan is done in advance before any sort of implementation has begin.

Deadlines

Furthermore, I would like to look further into the actual weighting of the course, and such, the project. Since many future sections will be based on the weighting, it is important to clearly label the descriptions, weighting, and deadlines of each proportion of the course. Each of which is outlined in the following table.

Description	Weight (%)	Date Due (DD-Mon-YYYY)
Feature list and implementation plan (including justification of development env)	10	17-Jan-2025
First prototype implementation with core features	10	31-Jan-2025
Implementation of standard features – i.e. completion of step (3) above	30	14-Mar-2025

Final System implementation (based on feedback from previous implementation)	40	04-Apr-2025
Project documentation	10	18-Apr-2025

Given a full course load, on top of both being a T.A. and working outside of both activities, I will be taking deadlines heavily to provide realistic expectations for each of the following implementation plans and definitions.

Core Features

As noted in Table 1, the first protype is the implementation of the core features. This implies that the finalized system is also built upon these core features. Such, defining key core features, which will act as a generalization of many features, is vital.

Defining Core Features

Given the current documentation provided for the project, we can group all potential features that will be needed as one of the following three core features.

1. Environment: Environment elaborates more on the overall container that the program will be supported in. This includes the structure of the program, both front end and back-end capabilities, live editing support, etc. Specifically support for multiple files and potentially the support of folders is also classified within the environment.
2. An embedded editor: Embedded editor can be interpreted similarly to Visual Studio or any other popular IDE. This should support
3. Role definition: Role definition should support limiting features or providing additional features depending on a role defined by the application. A general term to define role definition could consider anything that provided accessibility between the program and other users.

Any implementation plan from here on out will be grouped based on each of the core features. Discussion for additional core features can be planned in the future, given any remaining time to complete the project.

Prototype Implementation Plan

The first implementation of the program is a prototype; working proof to show the idea and concept of the Educator tool. With the deadline of the prototype currently scheduled to the 31st of January, it is key that for each core feature, there is a definitive and reasonable prototype plan.

Environment Prototype

Since the environment is primarily a web application on the front end, the environment is primarily going to be based on standard web application tools and processes (such as HTML, CSS, and JavaScript). However, for reasons that will be described later, I plan to use React as the main library

that will support the front end. This is because many of the other libraries are supported and accessible through React, making the development environment more convenient.

In terms of the style of the front end, as of the prototype, it should be primarily functional. Given the two-week deadline, there is just not enough time to fully make both a presentable and usable prototype. More discussion about the presentation and style will be developed after the prototype.

In terms of the back end, the primary focus should be on how a file might be saved to a particular user and how to send instances between multiple users. Given the two-week deadline, supporting a full user login system will take too much time. Such, in this case, we will consider a user as a temporary instance of the program -- as will be discussed later: a teacher role and a student role. Meaning that the back end will not be considered as much in the prototype implementation.

However, there is a plan to create a live editor experience between the two editors. Treat the current editor instance as a black box for now, it will be defined later, yet how do we create a live editor between two or more devices? For a full duplex experience, we will need to implement WebSockets. WebSockets requires the support of both a client side and a server side, so for the prototype, we will consider the teacher instance as the server side. Using Node.js is a popular tool to support this, especially for real time application, in addition to being scalable for the future. Node.js is also nonblocking and asynchronous, which will be vital when expanding the application.

Furthermore, a common way to implement WebSockets using React via useWebSocket and Socket.IO. Both are popular realtime messaging libraries, which have both connection diagnostics and capabilities to support the program. The majority of the use will be through useWebSocket through the Server Side implementation, with Socket.IO handling the inconsistencies on the client side.

Embedded Editor Prototype

While there are many code editor libraries and tools that work for a general-purpose embedded editor, there is a powerful code editor which can be extended to support many of the features needed for both the editor and role definition core features.

Monaco Editor: Monaco Editor is most known as the code editor that Microsoft's VS Code uses. The primary reason for Monaco that it structures its editors into separate instances. These instances allow the use for multiple files at a time and usability heuristics, but more importantly a way to update other users' instances to allow for multiple users to see and update a file at the same time. Furthermore, it also automatically supports visual aspects such as syntax highlighting and themes. As well as IntelliSense, a popular code editor tool that provides convenience to new users. All features which are nice to have, especially throughout the system implementation plan. All features listed here are supported for multiple programming languages and can be implemented through React. In addition, Monaco is able to support multiple files to support multiple file instances for a particular user.

However, the current issue with Monaco is that Monaco, while able to execute code with the help of editorWorker and getWorker (in addition to a few other language specific libraries), currently does not support live feedback between multiple instances of code, yet that is defined above in the environment prototype. While Monaco does not specifically run code, there are many API's that can handle this. For example, Google's Execution API for Apps Script. This is language depended, but also why we are using React and Node.js, to make it seamless with the server side execution.

Role and Accessibility Prototype

Given the two-week deadline, in terms of roles, there should be a minimum to ensure that the prototype can be developed. Currently, the proposal includes two primary roles which meets the purpose of the program. Given with how we have implemented the other sections, no additional libraries and tools are needed for now, and we will focus more on a feature list that each role supports.

Teacher Role: For the current prototype, I would like the teach role to highlight the following features

- Be able to view and edit other student files and code editor
- "Push" current code to other student files

Student Role: For the current prototype, the student role should be limited in terms of what it is able to do. For simplicity, it should not be able to the current features that the teacher roles is able to achieve.

System Implementation Plan

Notably, the prototype implementation plan can be done almost primarily through the frontend of the application. As noted above in the environment prototype, we have excluded the implementation of a proper server-side implementation for the sake of time. The system implementation plan will include the back end in the implementation. However, in terms of providing more support, I would like to rediscuss the plan for the finalized implementation plan at a future date, that of which is after the prototype implementation. Specifically for the backend, in which I am learning more about, and will be more confident about in the given future. This also allows for time to learn the necessary requirements for the back end to finish the project.

Furthermore, more visual aspects, like a more complicated terminal, would be nice to research and implement in the future.

References

https://www.freecodecamp.org/news/why-use-react-for-web-development/

https://microsoft.github.io/monaco-editor/

https://code.visualstudio.com/docs/editor/editingevolved

https://code.visualstudio.com/docs/editor/intellisense

https://github.com/Barahlush/monaco-lsp-guide

https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/

https://fastapi.tiangolo.com/advanced/websockets/#install-websockets

https://medium.com/@shreshthbansal2505/implementing-websockets-in-fastapi-9e7d8a236e8d

https://ably.com/blog/websockets-react-tutorial

https://www.npmjs.com/package/react-use-websocket

https://www.w3schools.com/nodejs/nodejs_intro.asp

https://www.freecodecamp.org/news/what-exactly-is-node-js-and-why-should-you-use-it-8043a3624e3c/

https://developers.googleblog.com/en/run-apps-script-code-from-anywhere-using-the-execution-api/#:~:text=The%20Execution%20API%20allows%20developers%20to%20execute%20scripts,and%20the%20Execution%20API%20will%20run%20your%20script.