Group 17
Advisor - Dr. Goce Trajcevski

Michael Davis - Database Management and Meeting Facilitator
Lily Krohn - Front End Developer and Meeting Scribe
Chris Lopez - Backend Developer and Record Manager
Kyle Marek - Front End Developer and Testing Engineer
Zachary Mohling - Full Stack Engineer and Project Manager
Griffin Stout - Front End Developer and Web Manager

Team Email: sddec20-17@iastate.edu
Team Website: https://sddec20-17.sd.ece.iastate.edu

Executive Summary

Development Standards & Practices Used

- Agile Development
- Continuous Integration / Continuous Delivery (CI / CD)
- Test Driven Development (TDD)
- Source Control (Github)
- Task Management Software (Github Issues)
- Backend Documentation (API Docs/Method Stubs/Swagger)
- Organized file structure
- High level design
- Web design standards

Summary of Requirements

- Application is accessible via Web Browser
- Multiple users are able to collaborate on the same file simultaneously
- Application supports the compilation of node based web applications
- Application supports an online text editor
    - Syntax Highlighting
    - Error Detection
    - Auto-completion
- Integrated version control via Git
- Server is able to store user files
- Backend supports direct interaction via websockets
- Automatic file saving
- Support file version control
- Support live edits for files
- User verification for login
- Utilization of Digital Ocean's infrastructure
- Utilization of open source softwares and libraries

Applicable Courses from Iowa State University Curriculum

- COM S 309 - Long term project management, team project management
- COM S 327 - Advance programming, large project development
- S E 329 - Project management

- S E 185/ EE 285 - Introduction into programming
- COM S 311 - Algorithms and advanced data structures
- COM S 363 - Database management systems
- S E 319 - Concurrent programming

New Skills/Knowledge acquired that was not taught in courses

- React.js
- Typescript
- Monaco Text Editor
- Kubernetes
- Digital Ocean
- Docker
- PostgreSQL
- Websockets

Table of Contents

1. Introduction

This section will cover our overall problem statement, functional and non-functional requirements, assumptions and anticipated deliverables.

1.1 Acknowledgement

Special thanks to Dr. Goce Trajcevski for his help with project planning and development.

1.2 Problem and Project Statement

In a world that is becoming more dependent on cloud storage and cloud computing, there is more of a focus on making software development less dependent on the personal computer. However, almost all software is developed on a local machine by one collaborator at a time. Developing on a machine requires additional steps to share files and download files from the internet. On top of these additional steps, there is maintenance to ensure that the project will be able to run on other computers. All of these tedious steps and maintenance just to simply run the project can be very time consuming and reduce the efficiency of software development.

The solution we have created to solve these issues is WIDE, a web integrated development environment. WIDE is a web application, meaning it will be accessible through a browser by going to our URL. Through WIDE, users are able to save files, develop software, and run node based web application projects all on the cloud. Users will no longer need to download files or upload files from their computer to the cloud. Nor will they have to be concerned with the project running on their local machine, because all code can be executed on the remote server. To top it all off, WIDE will enable users to collaborate on files together with real time updates. WIDE is the path to a more efficient, and less dependent future for software development.

1.3 Operational Environment

The operating environment of WIDE is a device's web browser, and it is served from various public cloud services. The targeted frontend environment is web browsers of consumer desktop and laptop operating systems.

1.4 Requirements

Functional Requirements

- Collaborative editing of source files

- Built-in Git UI for version control
- Authentication of users for joining and logging into projects
- Autosaving and source history
- Create/Import project
- Execution output and interaction
- Compile and serve artifacts

Non-Functional Requirements

- Low text collaboration latency
- High scalability
- Quick bug and feature deployments
- High availability

Engineering Constraints

- Time - this has been our main constraint
- Learning Curve - our team had to learn many new technologies quickly
- Budget - hard to find cheap but efficient cloud hosting service

1.5 Intended Users and Uses

Our intended user is any frontend JS developer or team which would prefer to keep project files on a consistent development environment. The intended use is for when individuals or teams who want to work on the same code at one time without having to install software or have multiple people work together on the same computer.

1.6 Assumptions and Limitations

Assumptions

- Basic security is be implemented (such as database protection)
- Our application will only include text in English

Limitations

- Javascript, HTML, and CSS will be the only available languages at first due to its compatibility with the Node.js runtime environment
- Advanced security implementations are lower priority compared to our main features
- Our compute infrastructure is limited to a fixed, monthly budget

1.7 Expected End Product and Deliverables

- Service-oriented Backend System
    - A microservice-based system which supports the functional requirements
    - Each service will be containerized and orchestrated to simplify deployment and provide resilience
- IDE Web Application / Editor
    - Our front end features a web based-text editor where users can simultaneously edit, debug, and execute code

2. Specifications and Analysis

This section will focus on our project approach and development process.

2.1 Approach

The approach method of this semester was improving our basic Monaco text editor and implementing the functional requirements of collaborative editing of source files, a built-in Git UI, the authentication of users when joining and logging into projects, autosaving and source history, and other requirements. We have also developed keeping in mind the non-functional requirements of fast response time, allowing for support of adequate number of users, and having sufficient storage space available.

2.2 Design Analysis

As the semester has gone on, we have made changes to our design implementation. We then updated the conceptual diagrams that organize the layout of all of the components that make up our project. Through meetings with our faculty mentor, we have identified the main features to focus on implementing in our project and have developed several use cases and scenarios to support those features. These use cases were also helpful in the development of our testing.

2.3 Development Process

We followed an Agile Development Process featuring 2 week sprints. This helped us to continuously develop and produce code and easily refinecode refinement. We also followed the Test Driven Development process to prioritize the testing throughout the implementation of our project.

2.4 System Diagram

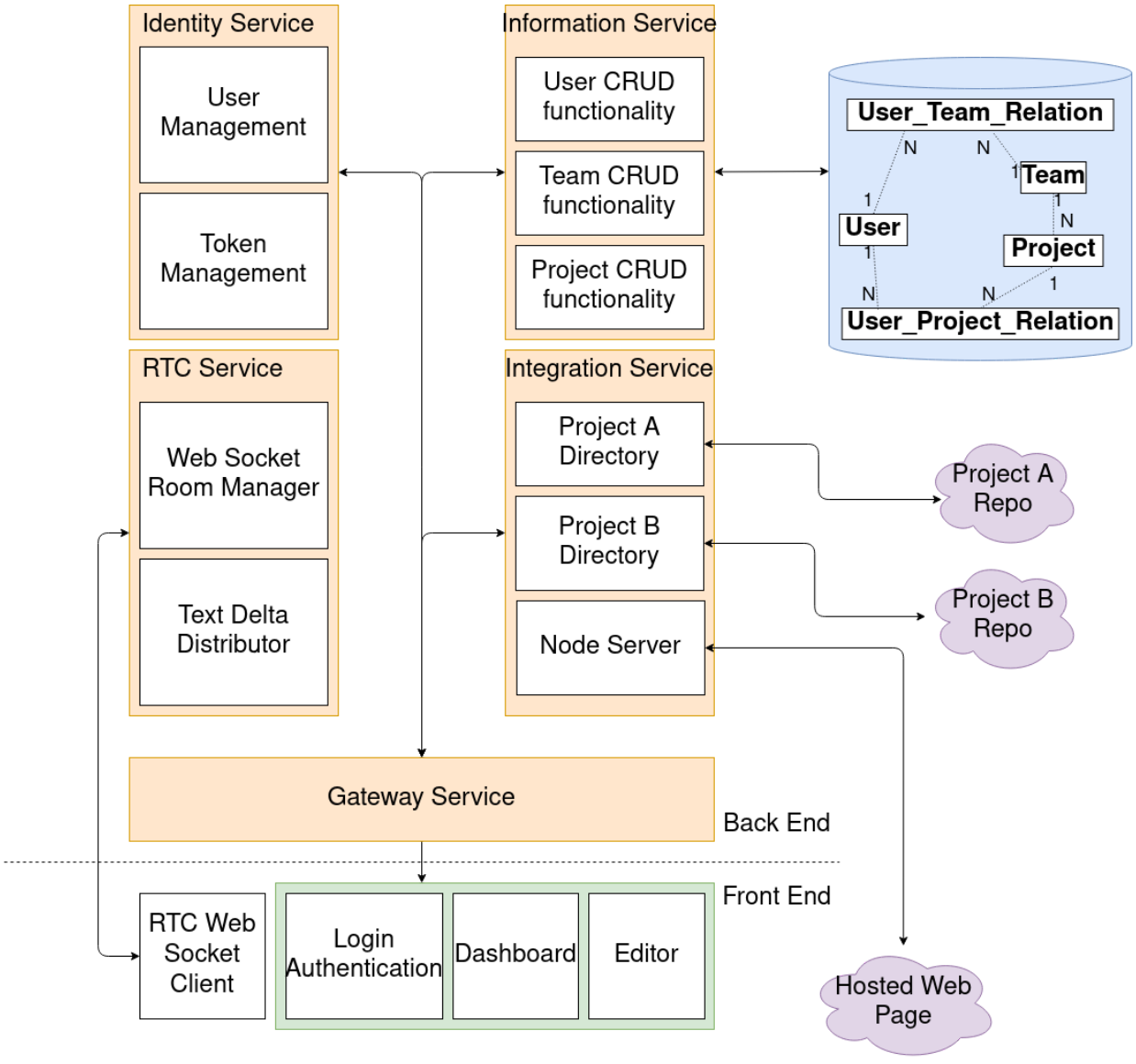

Figure 1 *- Architectural diagram showing backend and frontend services and modules.*

This diagram displays how the various elements and services of our system work together to support our project.

3. Statement of Work

This section will explain the task breakdown, language specifics, and expected results of WIDE.

3.1 Previous Work and Literature

There are a few applications that have similarities to our proposed project (ex: CoderPad), however they are lacking several things. CoderPad allows for simultaneous editing, however it lacks integrated version control. There are other well functioning IDEs, such as Visual Studio code, however there is no simultaneous editing [1]. Our project combines the features of simultaneous editing, auto saving, version control integration, and the functionality of a typical IDE to create a productive environment and allow for more efficient remote or in-person teamwork focused on web applications.

3.2 Technology Considerations

We selected an ecosystem of technologies that have been proven in the industry to support efficient, reliable software systems. The following is the list of the technologies we used, and their features which influenced our decision.

Infrastructure

Kubernetes

- Autonomous container orchestration
- Easy deployment
- Fault tolerance

Docker

- Provides a reproducible production environment
- Efficient alternative to virtual machines

Languages and Frameworks

Golang

- Standard library with built-in concurrency primitives
- Simple memory management
- Statically compiled into a single, versionable binary

TypeScript

- Strong typing system avoids increasing runtime error opportunities
- Included composite and utility types

React

- Fast and scalable
- Clean abstraction and reusable components

Data

PostgreSQL

- Performant and scalable relational database
- Large community and corporate support and maintenance

Protocols

WebSockets

- Long-lived sessions
- Unprompted server-to-client communication

3.3 Task Decomposition

The project was broken down into the following major tasks:

- Basic backend to frontend communication
- Account management and basic concurrent editing capabilities
- Concurrent editing implemented with more functionality
- Version control integration defined, concurrent editing with version control
- App is tested and all features are fully functioning

3.4 Possible Risks and Risk Management

Our main risks included time constraints and technology decisions. As we got more into development, we were dealing with user information. In the future we will need to implement more security precautions, but due to time constraints we did not make this the focus of our project.

3.5 Project Milestones and Evaluation Criteria

Some of our proposed project milestones include:

- A GUI the user can interact with
- A monaco-based text editor that allows simultaneous editing, syntax highlighting, error detection, “undo” abilities, as well as autosave [2]
- Comprehensive database system to handle both our internal and external file structures as well as users
- A way to share projects among users as well as register accounts and login
- Ability to import/create project as well as compile and serve it
- Integrate Git for version control

3.6 Project Tracking Procedures

We will be using Github's built in Scrum infrastructure to track our progress during this project.

4. Testing and Implementation

This section will focus on what we will use to test our project as well as how we will test our project.

4.1 Interface Specifications

Given that our project is software based besides the machines, there is no need for hardware testing. The interfaces this project will create are as follows: client-to-server interfacing will involve an API over websocket connection developed by the backend team. A microservice architecture will be implemented that will be interfacing via custom HTTP REST API as well. These interfaces will be both manually tested via software like Postman, and automatically tested through the tools listed below.

4.2 Software Tools

Frontend Testing Tools

React-testing-library

- Easily maintainable and scalable tests
- Tests resemble the way that the software is going to be used

Jest

- Works well in conjunction with react-testing-library
- Snapshot testing and implementation testing

Backend Testing Tools

Postman

- HTTP request testing
- Quick, easy API testing

Built-in Go testing

Manual Testing

4.3 Functional Testing

Testing a system which has adopted a microservice architecture introduces unique challenges and barriers. In order to test the elements of our frontend system, we utilized the built-in React

Testing Library in combination with the Jest testing framework in order to test the effectiveness and performance of our Typescript functions. Using Jest in combination with the RTL also allowed us to mock up data to test the http requests within our functions. Postman was also used to mock API calls and analyze their output. We also manually tested the functionality of the website to ensure correct usage.

Unit Testing

Our backend system utilized the Test-driven Development process to ensure total coverage of our critical modules. These fine grained tests provided certain guarantees about our business logic by observing a module's state, behavior, interactions, and collaborations. Using Postman we were able to continuously monitor the state of our services as well as the desired output for certain test cases.

Integration Testing

Integration testing verifies the communication path and interactions between the components to detect interface defects. [3] When applied to microservices, a tracing tool is paramount to observe the communication path between microservices and their assumptions each has about its peers. Our system uses Linkerd, a service mesh framework, to trace requests which enables service dependency analysis, root cause analysis, distributed transaction monitoring, and more.

4.4 Non-Functional Testing

There will be various levels of testing for each non-functional component of our project.

- Performance: For testing the performance of our application on a high level, we tested certain aspects of our project after their respective dates / deadlines that are specified in the timeline. For example, for testing the login for our application, we needed to evaluate the loading time and the time it took for the confirmation email to send to the user.
- Security: At this time, our team has decided to not prioritize in-depth security for our application. If we were to continue on this project we may choose to begin implementing and then testing the security of small parts of our project (ideally database data to start) by performing simulated hacks.
- Usability: In order to test the usability of our app, we gathered our team members to test the project's user experience. This allowed us to get feedback on how real users would interact with the project, the format of pages, error messages, how the collaborating features are working, and what we should change in order to create a more user-friendly application.

- Compatibility: For compatibility purposes, we tested how various repos respond with our product and if they are able to load into our editor quickly and fully. We also tested how our project looks on various different browsers.

4.5 Results

The results of our project can be seen in the below screenshots of our website. The first screenshot shows the login page where you can either sign in to or register for WIDE. The second shows the dashboard page where you can view teams and projects, and the third shows the editor when you open a project. Please visit our senior design website to see a video of it in action.

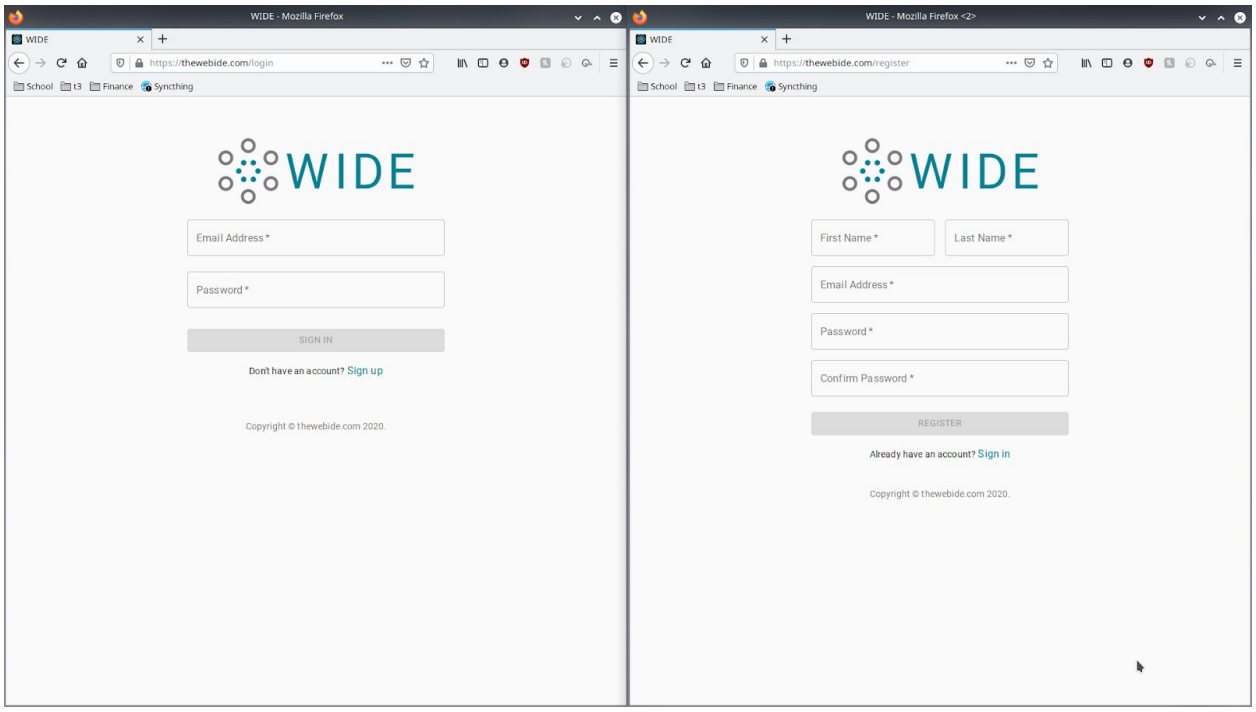

Figure 2 *- Login and Register pages*

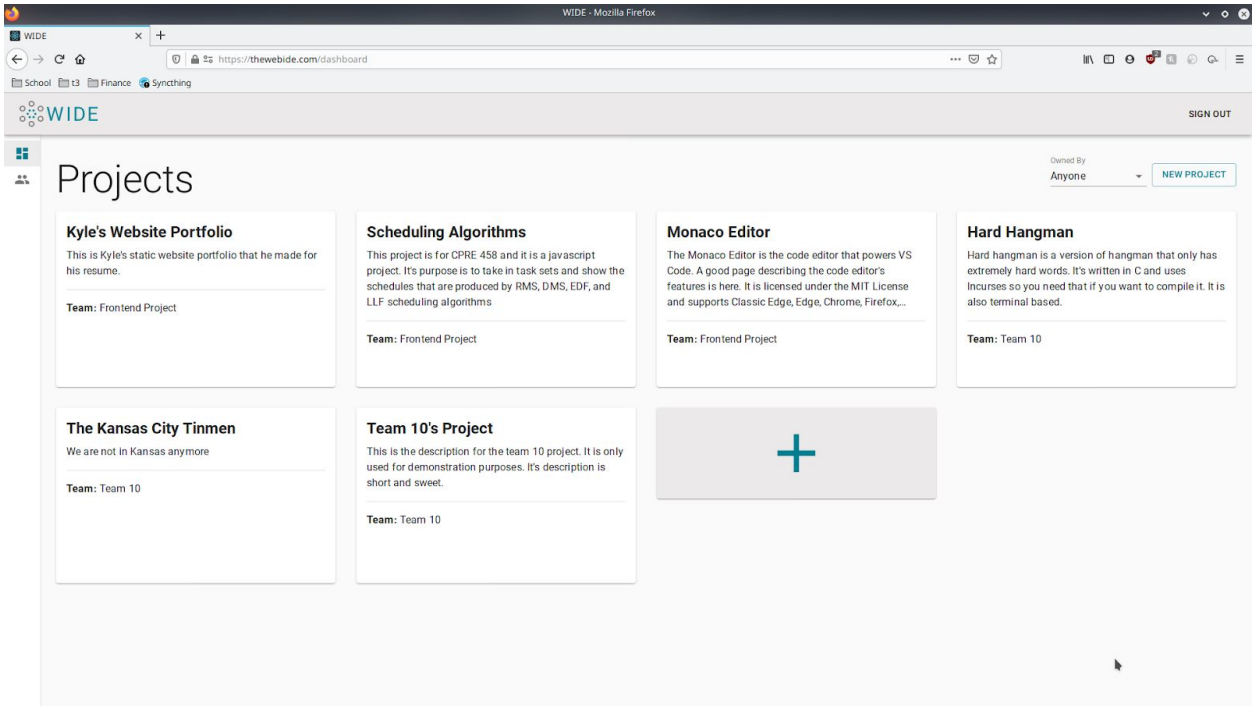

Figure 3 *- Dashboard page. After log in your projects and teams can be viewed here.*

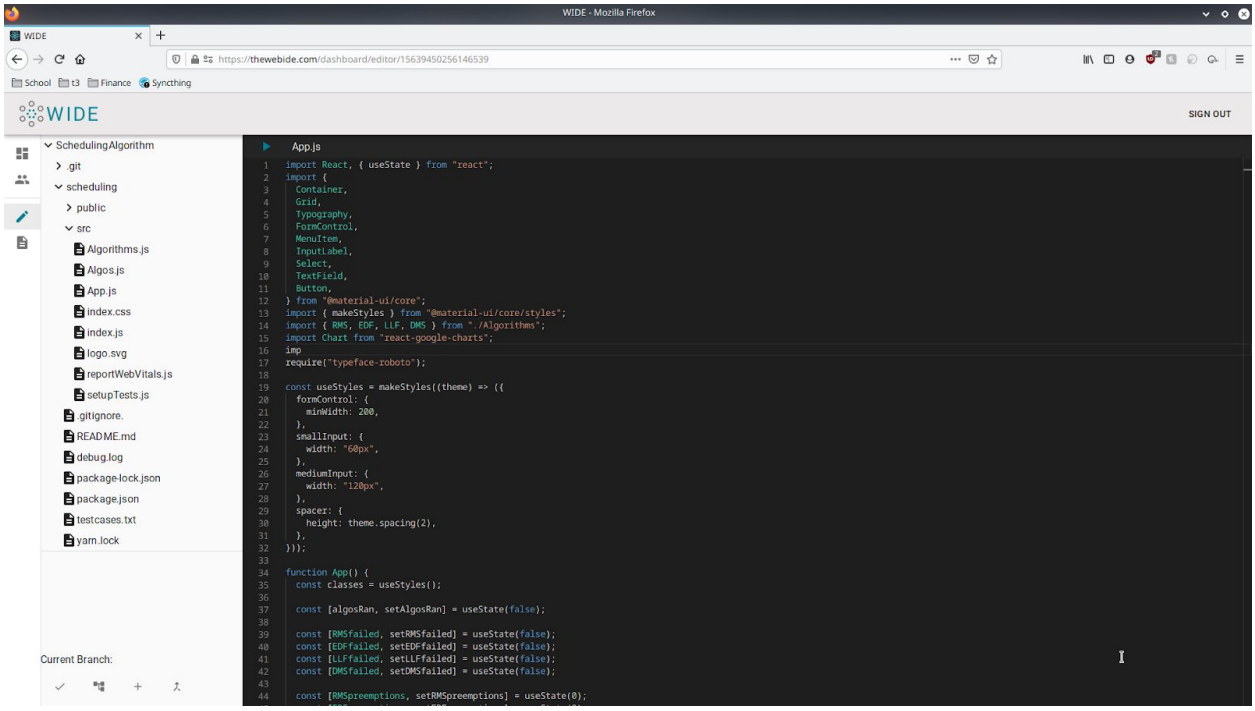

Figure 4 *- Editor page. File tree can be seen on the left. Clicking on a file will open a new session and populate the editor on the right.*

5. Closing Material

This section focuses on concluding remarks and the supporting documents used throughout this report.

5.1 Conclusion

Throughout the second semester of this course, we have achieved our main goals of implementing a web-integrated development environment. Although we had to alter certain features along with the timeline we created during the first semester of this class, we were still able to follow an efficient development process and complete our main application features.

5.2 References

[1] WebsiteSetup.org. 2020. *10 Best IDE Software For Web Development (2020) | Websitesetup.Org*. [online] Available at: <https://websitesetup.org/best-ide-software/> [Accessed 11 April 2020].

[2] Microsoft.github.io. 2020. *Monaco Editor*. [online] Available at: <https://microsoft.github.io/monaco-editor/> [Accessed 11 April 2020].

[3] Dhaduk, H., 2020. *Microservices Testing Strategies, Types & Tools: A Complete Guide*. [online] Insights on Latest Software Technologies - Simform Blog. Available at: <https://www.simform.com/microservice-testing-strategies/> [Accessed 11 April 2020].

5.3 Resources

[1] Material-ui.com. 2020. *Material-UI: A Popular React UI Framework*. [online] Available at: <https://material-ui.com/> [Accessed 11 April 2020].

[2] Theia-ide.org. 2020. *Theia - Cloud And Desktop IDE Platform*. [online] Available at: <https://theia-ide.org/> [Accessed 11 April 2020].

[3] Visual Studio. 2020. *Visual Studio Live Share | Visual Studio - Visual Studio*. [online] Available at: <https://visualstudio.microsoft.com/services/live-share/> [Accessed 11 April 2020].

[4] Reactjs.org. 2020. *Tutorial: Intro To React – React*. [online] Available at: <https://reactjs.org/tutorial/tutorial.html> [Accessed 11 April 2020].

[5] Redux.js.org. 2020. *Redux - A Predictable State Container For Javascript Apps. | Redux*. [online] Available at: <https://redux.js.org/> [Accessed 11 April 2020].

[6] ReactRouterWebsite. 2020. *React Router: Declarative Routing For React*. [online] Available at: <https://reacttraining.com/react-router/> [Accessed 11 April 2020].

[7] Docker. 2020. *Docker Documentation*. [online] Available at: <https://docs.docker.com/> [Accessed 11 April 2020].

[8] Kubernetes.io. 2020. *Production-Grade Container Orchestration*. [online] Available at: <https://kubernetes.io/> [Accessed 11 April 2020].

[9] Postgresql.org. 2020. *Postgresql: The World's Most Advanced Open Source Database*. [online] Available at: <https://www.postgresql.org/> [Accessed 11 April 2020].

[10] Access.crunchydata.com. 2020. *Crunchy Data Postgresql Operator Documentation*. [online] Available at: <https://access.crunchydata.com/documentation/postgres-operator/latest/> [Accessed 11 April 2020].

[11] Golang.org. 2020. *The Go Programming Language*. [online] Available at: <https://golang.org/> [Accessed 11 April 2020].

[12] Libgit2.org. 2020. *Libgit2*. [online] Available at: <https://libgit2.org/> [Accessed 11 April 2020].

[13] Hackernoon.com. 2020. [online] Available at: <https://hackernoon.com/building-conclave-a-decentralized-real-time-collaborative-text-editor-a6ab438fe79f> [Accessed 11 April 2020].

[14] Conclave-team.github.io. 2020. *Conclave*. [online] Available at: <https://conclave-team.github.io/conclave-site/#conflict-free-replicated-data-type-crdt> [Accessed 11 April 2020].

[15] Jaeger. 2020. [online] Available at: <https://www.jaegertracing.io> [Accessed 11 April 2020].

Appendix I - Initial Design

Originally, we considered implementing this project so that it supported C and compiling and executing those artifacts via a remote terminal. However, we discovered quickly that a remote terminal would cause huge security issues and it would be difficult to interface with terminal-based C applications, and impossible to interface with GUI based applications.