Using Containers to Isolate Remote Code Execution for an Online Development Environment

Joseph Fazzino
24026478

Supervisor: Dr. Hong Wei

29th April 2019

Contents

0.1 Abstract

250 - 300 words

outline aims, methods, implementation, achievements, and conclusions

0.2 Acknowledgements

I'd like to acknowledge Dr. Hong Wei for being my project supervisor. Dan Justin and Dr. Martine Magnan for their continued support through the early stages of my career. Suhail Parmar for contributing to my caffeine levels and providing help with Docker and UNIX. Dan Davis and Max Denning for helping me brainstorm the idea for the project and enduring me talking about frontend for the past year. And finally, my parents, Paul and Joanna Fazzino for being unwaveringly supportive and great role models.

0.3 Glossary of Terms and Abbreviations

The following is a list of abbreviations that are commonly used in this document:

API - Application Platform Interface

CRA - Create React App

RTC - Real-Time Communication

REPL - Read-Evaluate-Print-Loop

REST - Representational State Transfer

HCI - Human Computer Interaction

UX - User Experience

DX - Developer Experience

UI - User Interface

JS - JavaScript

OS - Operating System

P2P - Peer to Peer

PaaS - Platform as a Service

VM - Virtual Machine

VPN - Virtual Private Network

LXC - Linux Containers

SSR - Server Side Rendering

IDE - Integrated Developer Environment

Chapter 1

Introduction

There is a theory which states that if ever anyone discovers exactly what the Universe is for and why it is here, it will instantly disappear and be replaced by something even more bizarre and inexplicable. There is another theory which states that this has already happened.

Figure 1.1: The Universe

Chapter 2

Problem Articulation & Technical Specification

2.1 Context

As computers have become more pervasive, coding has become a skill that has graduated past being something that only people who work in laboratories need to concern themselves with, to a skill that has become highly desirable commercially and is starting to be taught in the regular curriculum to children studying at a primary level education[1]. This new demand for beginner friendly coding tools lends itself nicely to the promise of an online based environment where people can get started with basic coding concepts without having to trawl through documentation and technical detail about how to get running with one of the popular languages/tools available. This has led to an explosion of popularity for web applications such as `codecademy.com` which offer pre-made, executable exercises for a number of languages. A similar platform `repl.it` offers a more open and free-form experience and attempts to recreate the environment a developer may have on their machine through the web browser along with online compilation.

2.2 Problem Statement

A common pattern with the current platforms that exist is that they provide a strict sandbox within the confines of a predetermined configuration that the user selects, for example, in `codecademy` and `repl.it` you're stuck in the environment you pick when you start desired tool. An argument can be made that this makes a new developers life easier as they don't have to consider the more nuanced parts of the file system or learn any sort of terminal commands. However, it seems as though there would be value in a system that can provide both the ease of use that current existing solutions offer and also the freedom to explore a full environment with an array of tools preconfigured that encourage exploration without compromising the security and integrity of the underlying system.

2.3 Technical Specification

Based on the problem statement the potential scope for the project is very broad, there are companies and teams of developers that have the sole goal of making sure their online environments are providing users with as smooth an experience as they would expect if they had installed the tools locally.

This project will focus on the essential functionality required to behave as an online development environment while supporting a good variety of languages and offering a space which encourages exploration into different coding concepts.

With the above in mind the enumerated objectives of this project are:

1. Create a platform where users can write/execute code
2. Give every user their own personal environment
3. Eliminate the need for locally installed tooling
4. Provide a system that encourages exploration into the world of development

2.3.1 Writing and Executing Code

As an essential requirement for the development experience, the ability to edit and execute code is crucial to satisfy the overarching objective of creating an online environment. The execution of code presents a significant technical challenge however as the only code execution that can be done remotely is on a web browser which must be able to execute HTML, CSS and JavaScript. Mobile applications developed for iOS and Android are not capable of executing code.

Functional Requirements

- Code will be able to be typed using the platform
- Code will be able to be saved
- Code will be able to be read from the platform
- Code will be able to be executed

Non-Functional Requirements

- A good variety of languages will be supported
- The basic features of a code editor will be available (i.e. syntax highlighting)
- Code that is executing will not stall the platform

2.3.2 Personal Environments

The need for the space that the user occupies to feel personal is a vital element to local development environment and therefore must be well implemented for an online equivalent.

Functional Requirements

- A personal environment will be allocated to every user

Non-Functional Requirements

- The personal environments will be isolated from the rest of the system
- The personal environments will be isolated from each other
- The personal environments will perform well and be responsive to user input
- If a personal environment fails then it will be restarted

2.3.3 Local Tooling Replacement

Tooling has been through some big changes both in web browsers and locally. Web browsers have got to the point where they are so powerful that some of the most popular desktop software is being powered by them[2]. It is important to provide tools that will help those new to development, while also offering experience in tools that are of a high quality.

Functional Requirements

- High quality tools will be available to the user
- Industry standard tools will be available to the user
- The system will eliminate the need for local tooling

Non-Functional Requirements

- Popular tools will be researched and considered before being added to the system
- Tools will be standardised across the system
- Tools will be customisable to the users needs
- Tools will behave in a responsive manner

2.3.4 Encourage Exploration into Development

Lowering the barrier to entry through the requirements stated above will inherently make it easier to explore development but more steps can be taken in order to engage users with the system such as allowing them to create short coding exercises that can be shared with friends or on social media.

Functional Requirements

- Implement exercises for users to do
- Allow creation of exercises by users

Non-Functional Requirements

- Allow any exercise to be shared
- Assign difficulty level to exercises
- Provide an open area for the user to explore their personal environment

2.4 Stakeholders

This project has a number of relevant stakeholders with various degrees of interest in the outcomes. All of them will be considered during the construction of the system.

The Developer - Joseph Fazzino

The developer of the system is responsible for making 100% of the the technical decisions and is responsible for delivering a fully functioning system adhering to the technical specification found in Section 2.3 of this report.

Project Supervisor - Dr. Hong Wei

The supervisor of this project is overseeing the development and design process that is being undertaken.

They provide guidance when it comes to essential functionality and ways that technical requirements can be implemented.

User - Beginner Level Developer

Those new to development will not have experience with the terminology and syntax that exists in programming and wider computer science. They may have an understanding of basic coding concepts taught to them during formal education.

The beginner user should be able to use the system in order to become more familiar with generic programming concepts. The exercises available through the system will likely be the area they spend the most time.

User - Intermediate Level Developer

A user more familiar with the general work flow of a developer will be able to understand certain levels of nuance of how a system might be implemented and consider how they may solve certain problems.

This kind of user would benefit more from the ability to have a playground to explore the system in so they can understand the functionality that it provides and maybe try to explore the extent to which it works.

User - Experienced Level Developer

This user will have successfully developed systems with a high level of complexity and will most likely have specialised knowledge in a certain domain/environment.

This type of developer will be difficult to convince the benefits of an online working environment when they undoubtedly have a solution that works well for them locally.

2.5 Constraints

Some constraints on the development of the project exist.

- Permanent deployment - as the system is likely to be complex, deploying it will be costly and time consuming. Test deployment will be done to experiment with configuration settings in the system but a permanent live deployment will not be.
- Computer resource availability - the system will be constrained performance wise by the resources available during development meaning that any stress tests are not representative of a deployed system
- Significant testing base - as the system will not be deployed it will be difficult to adequately test the system in the manner which it would be used by end user. A different method of testing will have to be explored.

2.6 Assumptions

A number of assumptions must be made to reasonably meet the technical requirements.

- The users will have a reliable internet connection
- The users will have the necessary software/hardware configuration in order to access the system (e.g. a modern web browser)

Chapter 3

Literature Review

This chapter examines various literature around relevant subjects to the project objectives stated in Section 2.3. It looks at the various methods of **Real-Time Communication** that exist in order to create an environment where feedback is fast and frequent *(Section ??)*. It examines some **existing systems** that are providing some of the features listed out and critically examines the positives and negatives of some of the technical choices that are apparent in these systems *(Section 3.2)*. It concludes by looking at some of the modern advances in **Virtualisation** technology along with how the advent of **Containers** have changed the landscape of PaaS services and virtual environments in general *(Section 3.3)*.

3.1 Real-Time Communication

Real-Time Communication is an important research topic for this project as in order to create an environment for users that feels as close to a local experience as possible, the requirement for fast feedback is essential.

An experiment was done in 2012 discussing the performance of different RTC methods by Professors at the University of New Brunswick[3]. This experiment compared the different standard HTTP methods of implementing Real-Time Communication compared to the new (at the time) technology of WebSockets which are designed to create a fully duplexed bidirectional data-flow.

HTTP polling is an attempt to solve the real-time issue by repeatedly making a request to a web server at a pre determined time interval to check to see if there are any messages waiting to be read. **HTTP long-polling** is another solution that sticks to the HTTP protocol but reduces the number of wasteful requests by having the server intelligently not respond to the request if there is no information available and hang until a timeout or information becomes available. Both of these solutions are inadaquate for a responsive system however because the HTTP protocol is still built on top of a system not designed for real-time, fully duplexed communication channel. HTTP relies on a standard 'Request-Response' model which is only half duplex so polling was only a solution that worked for systems that were reliably sending data at a steady rate such as sensors that are being queried for an API.

A modern solution to this is the **WebSocket** protocol proposed in RFC 6455 [4] which aims to reduce latency by a factor of 3 compared to HTTP in the real-time communication aspect. It is a fully duplexed, bidirectional communication channel that provides an efficient method of communicating between several different clients using a persistent connection between the client and the server. A client may connect to a websocket endpoint on the server, send messages to it, and the server may broadcast messages back to just that client or to every client connected. Due to this behaviour it is very popular

for creating text based chat communication systems.

WebSockets work by utilising a persistent TCP connection where messages can be sent back and forth without there having to be a new connection made every time. This behaviour is possible in HTTP since HTTP 1.1 however, WebSockets do not adhere to the standard, 'request, response' cycle that a HTTP request utilises. Any client connected to the socket is capable of broadcasting a message at any time. HTTP persistent connections also still suffer from latency due to the effort the protocol makes to control congestion [5]. take the concept further by making it simple to embed data with each request through the form of a string in the form of a JSON schema. This makes it ideal for the transfer of small chunks of text where only text is the required form of the response. WebSockets are not appropriate for downloading resources or assets such as images.

Another new approach of RTC on the web has been developed by Google in collaboration with other browser vendors called **WebRTC**. This technology is focused on streaming audio and video between different clients on the web. This new technology is aiming to be the replacement for the browser plugins that were necessary in order to use P2P video/voice chat software such as *Skype, Facebook Messenger, Google Hangouts, etcetra* [6]. WebRTC is more appropriate for applications that need a streaming based connection as it's latency is even lower than WebSockets due to it utilising the UDP protocol which has much less overhead compared to the TCP based connection of WebSockets [7]. WebRTC would not be appropriate for the use case of WebSockets as when transferring informational data between clients, such as a chat application, it is important to make sure that the data is being received in the correct order whereas UDP is less concerned so long as enough packets get transferred to create a stable audio/video connection.

3.2 Online Developer Environments

A number of existing solutions providing online development environments exist and have been analysed for the purpose of this review.

3.2.1 Repl.it

Repl.it is very similar to the idea proposed in the Problem Statement (Section 2.2) and a lot of the requirements lined out in Section 2.3. It offers a huge array of Repl templates available for users to get started with many languages/frameworks very quickly. It also uses the Monaco Editor provided by Microsoft in order to provide a first class text editor experience.

Repl.it takes advantage of containeris in order to gives users a full developer experience when visiting the system [8]. The system also uses it's own container orchestration software in order to scale the instances available to users up and down depending on demand.

Every code result that is available to be viewed/run is viewable through a special .repl.run subdomain. This includes long running processes like web servers which are able to be hosted from these subdomains and be always accessible. This means you could

create several repls which all connect to each other like a full system.

Technically the system is very impressive, something that the system doesn't recreate quite as smoothly as a local environment would is a small amount of latency between a key being pressed and the corresponding value appearing in the REPL itself.

The system also seems to remove all previously typed entries of the REPL on every press of the *Run* button. This suggests that it is giving you a new REPL instance on every execution which isn't how a local environment works.

From a HCI point of view the website feels very smooth to use and is not frustrating to use other than the latency noted when typing directly into the running container via the REPL.

Repl.it is clearly very focused on the objective of replacing local development environments and does a good job of fulfilling that need.

3.2.2 Codecademy

Codecademy is a educational focused online environment designed to teach users how to code. Ranging in topics from beginning web development to a course to the IBM Watson API. It is a more directed experience than Repl.it as users are performing tasks for exercises but they are typing code into a similar environment, the code is executed and the result is displayed to the user.

Codecademy does allow access directly to the REPL but if code is entered into the editor which allows for user input such as the `input()` function in Python. Then it interprets the input correctly.

The Codecademy web application is clearly a very complicated system and it shows by how unresponsive it feels when navigating from page to page. The page does a full refresh even though there are elements which do not change on the screen page to page. This leads to a frustrating wait looking a blank screen between page loads.

It is clear that Codecademy is a focused environment to encourage new developers to get into development by offering an easy to start environment and heavily directed experience. It is not concerned with the idea of replacing local development environments so much as making sure that it's not something beginners should need to think of when wanted to get to know a new tool.

3.2.3 Glitch

Glitch is a web application that is focused on trying to cultivate a social coding community that encourages developers to help each other out and build mini applications with JavaScript and Node.js. It provides an online coding environment that uses containers to isolate the users runtime.

Glitch is clearly focused heavily on the social aspect as on the homepage they have a section dedicated to users asking for help so more experienced coders can help them achieve their goals with the applications they want to build. It also showcases user made projects on the homepage which can be *Remixed* which is similar to forking a repository on GitHub for other users to modify.

In terms of design, the website has a very colourful friendly interface. A feature which is particularly notable is in each project editor there is an option to view *Container Stats* where the CPU usage in %, Memory usage in bytes and additional relevant information can be found. There is also guidelines on the technical restrictions to projects that are run in Glitch.

3.3 Virtual Machines and Containers

In order to provide as close to local experience as possible to the users of the system this project aims to create. A virtual environment for executing code and saving files is vital. Virtualisation technology is changing significantly due to the different Container solutions which attempt to promote a more disposable and quick type of virtual environment compared to their hypervisor powered counterparts.

3.3.1 Virtual Machines

Virtualisation is a technique in computing that, most commonly, is seen by users in the **Virtual Machine** *(VM)* software. Virtual Machines are heavily utilised to provide virtual desktop environments on top of a users already existing desktop. The advantages of which are a sandbox environment for potentially harmful operations, such as when penetration testers are trying to fingerprint a virus. The option of trying a different OS without needing to dedicate a partition of disk space to it or deal with a dual booting set up is another user facing benefit of virtual machines.

In the enterprise world, Virtual Machines are being used to host their customers applications in a full PaaS solution so customers no longer have to worry about hosting their own web servers or other online services.

The general way of interacting with fully virtualised environments is through a hypervisor which is a tool that is responsible for provisioning and monitoring Virtual Machines [9]. The hypervisor allocates resources such as memory and CPU cores from the host machine that the VM is allowed to consume. When the VM is shut down these resources are freed and can be used by the host system once again. The hypervisor also allows the VM to use a different base operating system than the one that is on the host machine as it provides a whole *Guest OS*.

3.3.2 Containers

Containers are a much lighter virtualisation method than Virtual Machines despite the functionality being similar. They achieve this as they are much closer to the systems 'bare metal' as any commands that are executed through a container are running on the host's hardware. This means that there is no need for a hypervisor as containers have direct access to the resources. Usage limits can be set in the configuration of container *images* which will be discussed further during this subsection.

As Containers traditionally don't utilise a hypervisor the biggest difference between them is that the engine that powers the container provisioning software such as the *Docker Engine* isn't able to virtualise an environment based on a different OS. This is more by design however as it is what gives containers their 'light weight' quality as they aren't having to simulate a whole kernel. Not having a full kernel to set up however means that containers can start up significantly faster than a VM.

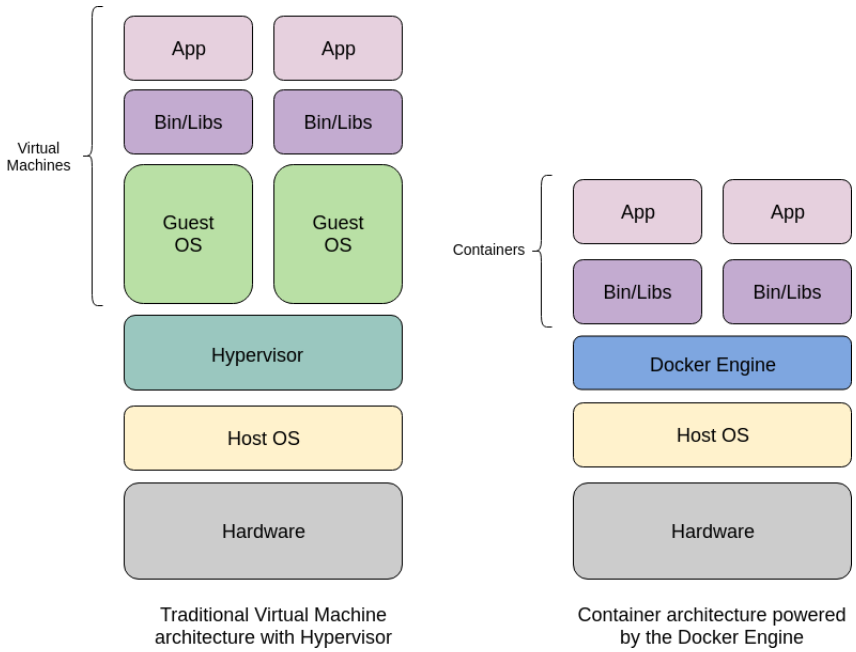

Figure 3.1: Architecture of Virtual Machines vs. Containers

Due to their performance benefits containers have become popular options for PaaS software. A paper was written comparing the benefits of a fully virtualised environment against a container based solution [10]. It concluded that containers have an inherent advantage over VMs due to the performance benefits and the quick start up time of them. It also mentions that few PaaS vendors are using containers for their systems so far as they are too new of a technology. It is worth nothing that the report was published in 2014 however and since then uptake will have increased.

Chapter 4

The Solution Approach

4.1 Solutions for Environment Virtualisation

In order to provide users with the most 'local' experience as possible on a remote platform it is key to analyse various technologies and techniques that are currently in widespread use in the industry. As discussed during Chapter 3 there is an consensus in the industry that containers are the better solution for PaaS type software of which this project would fall into the category of.

There are many different container solutions available currently, many have similar roots such as a backbone of using LXC but they build on top of those foundations in varying ways. Some of these ways are documented in the paper which was discussed in Section 3.3 [10]. This paper performed a comparison of the various different container technologies and stacked them against each other on key implementation features such as performance and security.

The decision to not research Virtual Machines as a potential solution for the system is due to the project not requiring the utility of a hypervisor and therefore does not require the ability to run a different Operating System on top of Linux. As almost all developer tools are freely available through most Linux distributions. This means that there are few to no downsides of using a container based solution versus a fully blown Virtual Machine set up and many benefits.

4.1.1 Container Providers

Linux Containers

As briefly mentioned above, **Linux Containers** or **LXC** are the foundation of many container solutions. This is due to the fact that they offer a light kernel implementation which provides every container with some key features.

- A unique Process ID for each container
- Isolates all resources for the container by using cgroups and namespaces
- Provides each container with it's own private IP address
- Isolates all files on the container from the Host by using chroot

In terms of downsides the LXC implementation is heavily tied to the Linux OS which means that it is not possible to run it on a different OS such as Windows. There are also some security concerns for LXC as all the containers share the one Host kernel.

OpenVZ Containers

OpenVZ makes use of a modified Linux kernel with it's own set of extensions. OpenVZ is able to manage physical and virtual servers with *dynamic real-time partitioning*. It similarly offers better performance than a traditional hypervisor based system and utilising the cgroups and namespaces features of Linux to provide it's virtual environments.

On top of the advantages of LXC it also provides the following benefits.

- **Container Lifecycle** remote management can be done of containers using an API to modify the status of a container in real-time.
- **Container State** is able to create checkpoints during the container's lifecycle so that it may be recovered from that point should anything go wrong.

A big limitation of OpenVZ is that it can't run on the standard Linux kernel so it is not a very viable solution for this project as the aim to to deploy the system and requiring a modified kernel will add complexity.

Docker

The Docker process is a daemon which can provide and manage Linux Containers as *images*. It uses LXC for the container implementation and then adds on top an image management system and a *Union File System*.

Using the daemon, Docker manages to provide similar functionality as the OpenVZ containers in relation to lifecycle and state. The state of a container at any time can be saved to a new image which can then be reloaded by the daemon to that same point

Unlike OpenVZ, Docker can be run with the standard Linux kernel and therefore is more suited for PaaS software. It also has a thriving ecosystem of pre-made images which offer a huge array of different starting points and tools.

4.1.2 Virtualisation Conclusion

For the system, Docker seems to be the clear choice as it provides good tooling with its daemon. A strong foundation on top of LXC, and it doesn't require additional modification before it can be used on a computer/server which is preferable to avoid for this project.

4.2 Chosen Solution for Real-Time Communication

Based off the research performed in Section 3.1 it would seem as though WebSockets fit the requirement of the project in order to ensure rapid communication between the client and their virtual development environment.

By using WebSockets it will be possible to have incredibly low latency bidirectional messages sent from the client to the server which can give the server instructions on what to do with the container that the client is allocated. The ability to send structured data chunks in string form makes it perfect for sending code and returning the output that is executed by the container.

WebSockets are available through the native Web APIs and so no 3rd party dependency is required to interact with them on the client side.

On the server side there are a few ways to implement a WebSocket endpoint but as Node.js and Express are already being used for the REST api it using a 3rd party dependency such as `express-ws` makes the most sense.

4.3 Requirements for Frontend

In order to create an experience that emulates a local installation of tooling and a text editor it is necessary to make sure that a tooling solution is chosen for this project that can meet these needs. The key requirements for the user facing side of this project are:

1. Text Editor with essential features that users expect in a standard developer environment
2. A terminal emulator that can display output of code to users and allow input where it is appropriate

Without these two features there is no way to adequately provide users with an environment that can be anywhere near the level of quality that they would expect from a local installation of tools.

4.3.1 Text Editor

A text editor is a vital part of a developers tool-chain and a few solutions exist on that can be rendered in the web browser. The reason a simple text input cant be used is that a text editor performs actions such as automatic indentation which, while could be recreated, is very specific to languages where for some indentation doesn't matter and for some it is how the language detects what code belongs to what block or function.

With this in mind it is helpful to establish a list of features that are a key requirement for any text editor.

- Automatic indentation
- Syntax colouring
- Control + F compatibility for Find
- Bracket matching
- Copy-Paste compatibility

With these features in mind it is now worth investigating the available resources to see which one satisfies the features best and if any bonus features cna be found.

Ace - *https://ace.c9.io/*

Ace is a code editor which is used by Amazon in order to provided their cloud based IDE 'Cloud9' it offers all the features that are listed above and notably includes support for themes, multiple cursors and bracket highlighting.

It's worth noting that Ace is first and foremost a code editor for online use and isn't available for users to install locally.

CodeMirror - *https://codemirror.net/*

CodeMirror is another code editor that is exclusive to the browser, it is the code editor that both Firefox, Chrome and Safari use inside their dev tools. It offers all the same features as Ace, however it has a more modern design which more accurately resembles a locally installed text editor.

CodeMirror claims to have experimental support for mobile browsers however it isn't an experience that is good enough to consider as a bonus feature for the editor.

Monaco - *https://microsoft.github.io/monaco-editor/index.html*

The Monaco text editor is made by Microsoft and used in their very popular code editor 'VSCode'. VSCode is one of the most popular text editors at the moment for a variety of different developer communities such as web developers and people starting out with a new language that don't want to have to deal with a fully blown IDE.

Feature-wise it offers all the features that are offered by CodeMirror and Ace but also comes built in with auto complete support for TypeScript, JavaScript, HTML and CSS. Through language servers as well, any language can gain support for auto complete for the standard language syntax. It also comes with a 'diff-editor' mode which can be used to gently introduce users to the idea of version control.

Monaco does not have as many themes available as the alternatives but the features that it does offer outweighs the value that alternate themes provides.

Decision on Text Editor

Based on the details above it makes a huge amount of sense to use the **Monaco Editor**, being provided by Microsoft is a significant benefit and the fact that it powers one of the most popular code editors that is being used in the industry means that it will provide as close an experience to a local coding environment as any of the other options.

Giving users experience with this tool will mean the transition to local development tools will be less abrasive as they already know the features that are available in these industry tools. The ability to provide auto complete for some languages is a huge benefit as well as new developers can be sure that the syntax they write is correct.

4.3.2 Xterm.js Terminal Emulator - https://xtermjs.org/

For online emulation of a terminal/command line the most popular solution is **Xterm.js** which has widespread adoption across many developer tools that require emulation of a terminal. It is used with VSCode in order to give users access to their local shell. It is highly performance focused in order to provide low to no latency between keystrokes and when they render on the screen. It also provides an array of addons that mean the terminal can be connected to a WebSocket stream so the online terminal can be connected to a real terminal on a machine. This is a key feature of a local development environment and therefore Xterm.js is ideal for this project.

4.4 Solution for Building the Interface

With the solutions established above it is now appropriate to evaluate the options when it comes to how to build the whole user facing side of the system.

As the developer has the most experience building websites using the `React.js` library (https://reactjs.org/) to help with creating reactive, data driven, single page web applications, that is the overarching technology that will be used to build the solution.

The need for a library such as React comes from the difficulty involved with maintaining data synchronisation between what is displayed on the screen and variables that exist in the JavaScript code. React also offers a high amount of code reuse with its component architecture.

Within the React ecosystem however, there is a number of options about how to manage certain essential features such as page routing and how to style components in a way that fits into the React methodology.

4.4.1 React Framework Options

In order to get started with React the recommended way is to use a package called `create-react-app` however a limitation of this method is that it is not configurable to

the extent that is required by the Monaco code editor if syntax colouring is considered a key feature, which it is.

It is neccessary then to evaluate some other options for getting started with a React application that allows for the configuration options that means syntax colouring works.

The popular options in the community are: ejecting a CRA project, Next.js or Razzle.

Eject Create-React-App

Ejecting a CRA provides the developer with full access to all the configuration options that are previously abstracted away. It also installs a lot of dependencies that need to be maintained correctly and the configurable options are overwhelming when all that's required is a few lines added to a config file. This solution is undesirable.

Next.js - https://nextjs.org/

Next.js advertises itself as a React 'framework' as it provides many additional features out of the box versus traditional CRA projects. It offers functionality for:

- Routing via the File System
- Code Splitting
- Server Side Rendering
- High Level Configuration

With these features, fewer external dependencies are required and the configuration which is needed to get syntax colourisation working is available.

Razzle - https://github.com/jaredpalmer/razzle

Razzle attempts to find a middle ground between the opinionated decisions made by Next.js and the overwhelming amount of configuration that is required after ejecting an app created by CRA. It is also agnostic to the technology that you use it with so it can be used with several other different frontend libraries such as *Vue.js*.

Due to the less opinionated nature of the project the only real benefit provided by it is the server side rendering and configuration options that is also provided by Next.js.

Despite similar configuration extensibility as Next.js trying to enable syntax colouring for the Monaco editor didn't work.

4.4.2 Conclusion on Frontend Framework

As syntax colouring has been described as a key feature there isn't much of a choice beyond choosing to use either Next.js or the Ejected CRA. As Next.js has a number of other benefits however, this makes it the most attractive and powerful option for building the frontend.

4.5 Overall declaration of solution chosen

Chapter 5

Implementation

Chapter 6

Testing: Verification and Validation

Chapter 7

Discussion: Contribution and Reflection

Chapter 8

Social, Legal, Health and Safety and Ethical Issues

Chapter 9

Conclusion and Future Improvements

Bibliography

[1] S. Chalmers, "Why schools in england are teaching 5 year olds how to code," Oct. 2014. [Online]. Available: https://www.bloomberg.com/news/2014-10-15/why-schools-in-england-are-teaching-5-year-olds-how-to-code.html (visited on 04/01/2019).

[2] T. Claburn, "Carlo has a head for apps and a body (tag) for rendering: Google takes on electron with js desktop app toolset," Nov. 2018. [Online]. Available: https://www.theregister.co.uk/2018/11/02/carlo_chromium/ (visited on 04/02/2019).

[3] V. Pimentel and B. G. Nickerson, "Communication and displaying real-time data with websocket," May 2012. [Online]. Available: https://ieeexplore.ieee.org/document/6197172 (visited on 04/03/2019).

[4] I. Fette and A. Melnikov, "The websocket protocol," Dec. 2011. [Online]. Available: https://tools.ietf.org/html/rfc6455 (visited on 04/03/2019).

[5] V. Choudhary, "Http vs websocket (or http 2.0), which one is right for you.," [Online]. Available: https://developerinsider.co/http-vs-websocket-or-http-2-0-which-one-for-you/ (visited on 04/06/2019).

[6] S. Dutton, "Getting started with webrtc," Jul. 2012. [Online]. Available: https://www.html5rocks.com/en/tutorials/webrtc/basics/ (visited on 04/06/2019).

[7] *Tcp vs. udp*. [Online]. Available: https://www.diffen.com/difference/TCP_vs_UDP (visited on 04/06/2019).

[8] F. Lardinois, "Repl.it lets you program in your browser," 2018. [Online]. Available: https://techcrunch.com/2018/03/15/repl-it-lets-you-program-in-your-browser/ (visited on 04/03/2019).

[9] M. G. Sumastre, "Virtualization 101: What is a hypervisor?," Feb. 2013. [Online]. Available: https://www.pluralsight.com/blog/it-ops/what-is-hypervisor (visited on 04/04/2019).

[10] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs containerization to support paas," Mar. 2014. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6903537 (visited on 04/04/2019).