



Developing a Web App to Improve the Usability of Terraform

Bailey Search

UP937683

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Bachelor of Science
of the
University of Portsmouth.

School of Computing
Engineering Project

2023

Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Date: 2023

Abstract

Cloud computing and DevOps remains a challenging area for beginners to break into, and like the rest of computing, is continuously changing and improving. There are countless ways to achieve the same thing when it comes to provisioning and managing cloud computing resources. Over time, many tools have been developed to help address these issues and streamline the process, but they end up adding additional complexity to an already complicated landscape and another thing for people to understand and use.

Terraform is one such tool that has risen to great popularity and support by cloud computing providers. It is an infrastructure as code (IaC) tool that allows users to manage cloud resources through declarative code files.

This project documents the development of TerraFormer, a prototype GUI Terraform file builder which improves the experience and usability of Terraform. The report includes gathering requirements, designing, developing, testing, and evaluating the artefact against the objectives and requirements.

Acknowledgements

I would like to thank my friends and family for their support throughout this project.

I would also like to thank my project supervisor, Dr Kirsten Smith, for her constant help, advice and guidance.

I would also like to thank all of the study and survey volunteers and participants.

Consent to Share

I consent for this project to be archived by the University Library and potentially used as an example project for future students.

Contents

List of Tables	9
List of Figures	10
List of Code Examples	12
1 Introduction	14
1.1 Background	14
1.2 Aims & Objectives	16
1.3 Project Deliverables	16
1.4 Legal, Ethical, Professional and Social Issues	16
1.5 Project Risks	16
2 Methodology	17
2.1 Introduction	17
2.2 Waterfall	17
2.3 Agile	17
2.4 Kanban	18
2.5 Extreme Programming	18
2.6 Conclusion	19
3 Literature Review	20
3.1 Research Strategy	20
3.2 Introduction	20
3.3 Software Usability	20
3.4 Evaluating Software Usability	21
3.5 Cloud Computing Usability	22
3.6 Conclusion	24
4 Software Requirements	25
4.1 Introduction	25

4.1.1	Purpose	25
4.2	Overall Description	25
4.2.1	User Needs	25
4.2.2	Assumptions and Dependencies	25
4.3	Existing and Similar Software	26
4.3.1	Stack Lifecycle Development	26
4.3.2	ChatGPT	26
4.3.3	Gaia & Terraboard	27
4.4	Study	28
4.4.1	Co-Discovery Study	28
4.5	Questionnaire	28
4.5.1	Q1 - Have you had any experience with Cloud Computing Providers such as AWS, Azure or GCP?	29
4.5.2	Q2 - How confident are you about working with the cloud?	29
4.5.3	Q3 - Have you had any experience with Terraform or any other Infrastructure as Code tool?	30
4.5.4	Q4 - For the above screenshot: What is the name of the Microsoft SQL Server resource when provisioned on Azure?	30
4.5.5	Q5 - For the above screenshot: If you were to change the location of the resource group, would it change the location of the database?	31
4.5.6	Q6 - For the above screenshot: If you wanted to use the database server's name as the name for the database, what would you use?	31
4.5.7	Q7 - How easy do you think it is to understand the configuration of each resource and how they connect together?	32
4.5.8	Q8 - Looking at the above example, what issues could you imagine if you had to work on this file?	32
4.5.9	Q9 - How useful do you think a visual node-based editor would be for managing Terraform files?	33
4.5.10	Q10 - What 5 features would be the most important to support?	33
4.5.11	Q11 - Are there any issues you could see with such an editor?	34
4.5.12	Questionnaire Results Conclusion	35
4.6	System Features and Requirements	36
4.6.1	Functional Requirements	36
4.6.2	Non-Functional Requirements	37
5	Design	39
5.1	Introduction	39
5.2	Client	39
5.2.1	Figma Mock Ups	40

5.3	API Server	41
5.4	C4 Diagrams	42
5.5	DevOps & Deployment	43
5.5.1	Client	44
5.5.2	API Server	44
6	Project Management	45
6.1	Introduction	45
6.2	Kanban Boards	45
6.3	Gantt Chart	46
6.4	Weekly Supervisor Project Meetings	47
7	Development	48
7.1	Introduction	48
7.2	Versions	48
7.2.1	v0.1 - Initial two column view	50
7.2.2	v0.2 - Change to a Node-based UI	51
7.2.3	v0.3 - Block Nodes & Styling Improvements	53
7.2.4	v1.0 - Prototype Works	54
7.2.5	v1.1 - Testing & HCL Displayer	56
7.3	Tests	57
7.3.1	Client end-to-end Testing	57
7.3.2	Client Component Testing	59
7.3.3	Backend API Testing	59
7.4	Continuous Integration	59
7.5	Continuous Deployment	60
7.6	Conclusion	60
8	Evaluation	62
8.1	Introduction	62
8.2	Methodology	62
8.3	Project Management & Time Management	62
8.4	Evaluation against Project Objectives	63
8.5	Evaluation against Non-Functional Requirements	63
8.5.1	F1 - Present a filterable list of available resources for the user	63
8.5.2	F2 - Allow the user to create an instance of a resource which can then be edited	64
8.5.3	F3 - Provide information about available configurations for a specific resource	64

8.5.4	F4 - Generate the code used to provision the selected resource with the selected configurations by the user	65
8.5.5	F5 - Allow the user to reference other attributes from a resource	66
8.5.6	F6 - Allow the user to add optional configuration options	67
8.5.7	F7 - Allow the user to reference object type variables	67
8.5.8	F8 - Inform the user of any mistakes or invalid configurations	67
8.5.9	F9 - Provide pricing information and estimate to help inform the user on expected costs	68
8.5.10	F10 - Provide links to other relevant documentation, such as from Terraform and the cloud provider's official documentation, as a shortcut in case the user needs additional information	68
8.5.11	F11 - Allow the user to save the current configuration so it can be edited at a later time	68
8.5.12	F12 - Provide a quick start option to build common resources quickly	69
8.6	Evaluation against Functional Requirements	69
8.6.1	NF1 - Provide a modern visual web application for users to interact with	69
8.6.2	NF2 - Generates valid Terraform HCL code	70
8.6.3	NF3 - A reliable, well-built app with a high level of quality. This means that the app should act predictably and work well regardless of how the user uses it	71
8.6.4	NF4 - Provide a greater level of abstraction for the user to make it easier to work using the cloud without having to spend a lot of time learning about it	71
8.6.5	NF5 - The web app must scale appropriately to cover standard desktop sizes	72
8.6.6	NF6 - The web app should comply with WCAG 2.1 AA accessibility standard	73
8.6.7	NF7 - Communicate securely with cloud provider APIs	74
8.6.8	NF8 - Support generating '.tf.json' files and '.tf' files	74
8.7	Future Work	74
8.8	Reflection & Conclusion	75
References		77
A Ethics Certificate		81
B Project Initiation Document		84

C Initial Study Co-Discovery Plan & Tasks	95
C.1 What is Terraform?	95
C.2 Commands	95
C.3 Tasks	96
C.3.1 Task 1	96
C.3.2 Task 2	96
C.3.3 Task 3	96
C.3.4 Task 4	97
C.3.5 Task 5	97
C.3.6 Task 6	97
C.4 Questions for each Participant:	97
D ChatGPT Terraform Test	98

List of Tables

4.1	Functional Requirements Table	37
4.2	Non-functional Requirements Table	38

List of Figures

4.1	Q1 - Have you had any experience with Cloud Computing Providers such as AWS, Azure or GCP?	29
4.2	Q2 - How confident are you about working with the cloud?	29
4.3	Q3 - Have you had any experience with Terraform or any other Infrastructure as Code tool?	30
4.4	Q4 - For the above screenshot: What is the name of the Microsoft SQL Server resource when provisioned on Azure?	30
4.5	Q5 - For the above screenshot: If you were to change the location of the resource group, would it change the location of the database?	31
4.6	Q6 - For the above screenshot: If you wanted to use the database server's name as the name for the database, what would you use?	31
4.7	Q7 - How easy do you think it is to understand the configuration of each resource and how they connect together?	32
4.8	Q8 - Looking at the above example, what issues could you imagine if you had to work on this file?	32
4.9	Q9 - How useful do you think a visual node-based editor would be for managing Terraform files?	33
4.10	Q10 - What 5 features would be the most important to support?	33
4.11	Q11 - Are there any issues you could see with such an editor?	34
5.1	Initial design using a dynamic form	40
5.2	Revised design using dynamic nodes to represent resources.	41
5.3	C4 Level 1: System Context Diagram for TerraFormer	42
5.4	C4 Level 2: Container Diagram for TerraFormer	43
6.1	GitHub Kanban Board	46
6.2	Project Gantt Chart	46
7.1	GitHub releases of TerraFormer	49
7.2	v0.1 - Initial two column form view	50
7.3	v0.2 - Change to a Node-based UI	51

7.4	v0.3 - Block Nodes	53
7.5	v1.0 - Prototype Works	54
7.6	Shared private NPM library	56
7.7	Generated Terraform File is now shown to the user	57
7.8	Cypress Axe Testing Results	58
7.9	Example CI GitHub Actions	59
7.10	Example Cloud Environment in Terraformer	61
8.1	The left sidebar of the website with a filterable list of resources	63
8.2	3 Unique Instances of the Resource Group type Resource	64
8.3	Provide information about available configurations for a specific resource	65
8.4	User resource configuration	65
8.5	Generated HCL from the user resource configuration	66
8.6	A Container Registry using the referenced attributes "name" and "location" from a resource group.	66
8.7	A MS Database resource with optional tag selected and listing another optional key the user could add	67
8.8	A MS Database using a Tag type object variable	67
8.9	Two resources with valid and non-valid values.	68
8.10	gtmetrix.com result page for TerraFormer	69
8.11	Terraform CLI validating the generated file	70
8.12	Shorter and wider viewport	72
8.13	Taller and thinner viewport	73
D.1	ChatGPT defaults to using outdated resources	99
D.2	Guiding ChatGPT to use more up-to-date resources	100
D.3	ChatGPT generates invalid resources and crashes	101

List of Code Examples

7.1	The resource object interface	52
7.2	The IResourceKey object interface	52
7.3	An example of block values within an Azure Kubernetes Cluster Terraform Resource	53
7.4	Before my Regex fixes	55
7.5	After my Regex fixes	55

Chapter 1

Introduction

1.1 Background

Terraform is a popular Infrastructure as Code (IaC) tool which allows developers to manage cloud resources through code. Over recent years, this has become an extremely common method to manage increasingly complicated cloud environments by developers, administrators, architectures and cloud engineers (Sabharwal et al., 2021).

Terraform uses a declarative language called HashiCorp Configuration Language (HCL) to define the end state of the cloud in a special file. HCL is a domain-specific language designed to easily be human and machine-readable (Riti & Flynn, 2021). Terraform also tracks the state of the cloud against the defined state in the file, in order to efficiently perform updates.

This has several benefits over traditional cloud management, such as through CLI or web consoles (Sabharwal et al., 2021). For example:

1. The files can be checked into source control. This allows for both easy sharing and easy viewing of history.
2. Building and destroying cloud infrastructure is quick and basically automated. This means that you do not have to remember how to create similar resources and since the tools track the state of the resources, they can easily delete all of the resources they are managing.
 - (a) This also provides a feasible way of only provisioning infrastructure when needed. For example, deploying infrastructure at the start of the day and then deleting it at the end. This reduces overall costs as you aren't paying for resources when you are not needing to use them.
3. Low risk of human errors compared to manually provisioning resources each time or when compared to writing cloud provider CLI commands in a bash shell in order

to perform an action.

4. IaC is usually declarative, not imperative. So you describe the end state of the cloud and its resources and it smartly works out how to get the current state of the cloud to match the end state. This means that resources are managed efficiently and quickly.
5. Building the resources from a file and deleting all managed resources is very quick and allows you to provision resources only when needed, which helps to reduce development costs.

Although it is one of the most popular options for efficiently managing cloud resources, it is not without fault. For example:

1. Using Terraform, or similar, is yet another thing that must be learnt. On top of the chosen cloud provider, the required resources for the project and how you use them, programming languages, frameworks and libraries and eventually how to wire them all up together.
2. Terraform Specific:
 - (a) Difficult to find out the required key-value pairs to create a required resource. You have to turn to the official documentation almost every time you need to make a change.
 - i. Key-value pairs are hard to remember.
 - ii. Can be difficult to tell which values are optional or not.
 - iii. Optional keys are not always listed in the official docs and can only be found mentioned on sites such as StackOverflow.
 - (b) Values are sometimes inconsistent with the structure of accepted values and capitalisation of words.
 - (c) Validity of key-value pairs depends on other configured options and requires you to look up and read the official resource documentation on the cloud service providers website to see which options are dependent on other options. For example:
 - i. Not all cloud resources or features are available in every location.
 - ii. Lower-cost versions of a resource often lack options and features.
 - (d) Most cloud providers use rolling deployments which mean that resources and the management APIs gradually change over time. So what works today might not work tomorrow. Although Terraform's providers have versions, these can

eventually break when the management APIs on the cloud provider change enough.

- (e) Cloud providers all handle equivalent resources differently so even if you are a very experienced with the AWS Terraform provider, you may struggle when it comes to using another provider such as IBM or Oracle.

These issues impact the usability of Terraform and creates a steeper learning curve for beginners.

1.2 Aims & Objectives

This project aims to develop a modern graphical web application to provide an easier and quicker way to develop Terraform files, which will improve the usability of many cloud service providers. This application will also provide an easier way for beginners to get started using the cloud by providing support and guidance on creating resources on different cloud service providers such as AWS or Azure.

It will do this by allowing the user to visually edit and configure resources and then automatically generate the supporting Terraform files, which, when executed, will provision the specified resources on the user's cloud account.

1.3 Project Deliverables

The project will have the following deliverables:

1. Literature review for evaluating the current level of usability within the cloud computing domain.
2. The TerraFormer web application.
3. This project report.

1.4 Legal, Ethical, Professional and Social Issues

During this project, I only plan to gather user data for study or questionnaire participation. While doing this, I must comply with the General Data Protection Regulation and the Data Protection Act 2018 ("Data Protection Act 2018", 2018).

The application will not store any personally identifiable information or user-entered data.

There are no other legal, ethical, professional or social issues.

1.5 Project Risks

The log of project risks can be found in Appendix B.

Chapter 2

Methodology

2.1 Introduction

This chapter will detail some project management methodologies I considered using for this project and what I ultimately decided to use and my reasoning behind it.

2.2 Waterfall

The waterfall methodology originated in the 1960s as the first software development methodology. It evolved from traditional hardware engineering project management, where engineers could easily predict project time and how pieces of the system would interact with each other (Cusumano & Smith, 1995). They wrongly assumed software development would be the same.

This methodology follows a simple process: analysis of requirements, design of the system, implementation, and testing.

The largest problem with the Waterfall methodology is that changing the requirements during development was difficult. Customer requirements can change for a variety of reasons and may not be fully known at the beginning of a large project.

The Waterfall methodology is still suitable for some small software projects where the requirements are set in stone at the beginning (Saeed et al., 2019).

2.3 Agile

Agile methodology is designed to offer a lightweight and flexible project structure to allow for quicker software development (Abrahamsson et al., 2017) and is the natural successor to the Waterfall methodology. Essentially agile scales the Waterfall methodology into regular small iterations with regular releases. This allows requirements and the design of the system to change over time and allows for better time estimations using previous feature timelines within the same project.

The agile manifesto details several core ideas that would improve software development. This was born out of the shared frustration of modern corporate software development by 17 developers, where planning and documentation are prioritised over functional software and customer satisfaction (Fowler & Highsmith, 2001).

Agile also encourages stakeholder involvement by regularly connecting with them and demonstrating the system. This creates a regular interval where feedback is given, requirements can change, and stakeholders can confirm the software is what is required (Edeki, 2015).

2.4 Kanban

Kanban methodology is a lightweight methodology within the larger Agile methodology.

Kanban uses a board, usually divided into 4 or more columns, to categorise project items. These items are small tasks that together create the software system. Unlike many other project methodologies, this visualises the workflow and helps limit what is being worked on (Kirovska & Koceski, 2015).

Kanban has also been linked to improved customer satisfaction, software quality, and communication between developers and stakeholders (Ahmad et al., 2013). Many projects use a hybrid of Kanban and Agile as a methodology, although this can complicate management.

A standard Kanban board will look similar to the following:

- TODO - Tasks yet to have work started.
- In Progress - Tasks currently being worked on by a developer. Tasks will usually have a Git branch or Pull Request linked to it for easy progress monitoring.
- Test - Tasks that have been completed but require another person, typically a QA engineer, to review them.
- Done - Completed tasks.

2.5 Extreme Programming

Extreme programming was the natural reaction to the Waterfall and the Iterative methodologies (Beck, 1999). Extreme programming focuses on very short cycles to ensure that any issues with time management can be corrected before causing large delays to the whole project.

Extreme programming also promotes using other practices, such as tests and continuous integration. Testing has two rules: developers write their own tests and use test-driven

development (Beck, 1999).

Extreme programming does have a few shortcomings. Documentation is usually lacklustre and outdated, and when working in a team, it requires frequent stand-ups and meetings (Saeed et al., 2019).

2.6 Conclusion

Ultimately, I decided to use a combination of Kanban and Extreme Programming to best satisfy my requirements with this project.

From Kanban, I want to use a Kanban board to track project progress and limit the work-in-progress tasks, to better focus on the most important tasks.

From Extreme Programming, I plan on using unit testing to test the main features of my software and use continuous integration to build and run the software automatically. I also want to release in small cycles, such as for every completed Kanban board item.

Chapter 3

Literature Review

3.1 Research Strategy

My research strategy has primarily consisted of using Google Scholar to find papers, articles and reports on the usability of software, cloud computing and developer preference.

3.2 Introduction

ISO defines usability as the "degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" (ISO/IEC 25010, 2011). A key aspect of that definition is the "specified context of use" as what makes a system usable differs based on the intended use (Ferré et al., 2001). Some systems might only be used by a user once and others might be used daily for years. Usability is defined by 5 basic attributes; learnability, efficiency, user retention over time, error rate and satisfaction (Ferré et al., 2001).

Usability isn't the only required attribute for software to be good, it also has to be acceptable (Holzinger, 2005). Acceptable refers to the software's suitability for a particular job and if it has the required functionality (Nielsen, 2012). Acceptable but not usable software has the necessary functionality for a job but is difficult to use and usable but not acceptable software is easy to use but lacks the required functionality for its job. Together, usability and acceptability ultimately determine a piece of software's success.

This literature review aims to identify methods and designs which can improve the usability of complex cloud environments for beginner users.

3.3 Software Usability

Poor usability is the biggest cause of software failure (Ahmed & Eduard, 2004). This does not mean that all software being used has good usability as there are often other conditions such as price, availability, and previous training which play into this (Ferré

et al., 2001). Humans also adapt quickly to computers in Human Computer Interaction (HCI) to overcome some usability issues typically with GUI applications (Mariage et al., 2006). Although, this will often lead to a reduction in efficiency, capabilities and user satisfaction.

By focusing on the software's usability from the beginning of development, it will reduce costs on the required rework (Anderson et al., 2001). Following usability recommendations often requires deeper work than just changing the surface level UI, such as providing a cancel options for a task, successfully reverting the state of the system to as it was and then notifying the user of the successful cancellation (Carvajal et al., 2013). Studies have shown that approximately every dollar spent on usability returns \$30 (Saxena & Dubey, 2013). Involving the end user from the start of the design process is an approach called "User-centered design" and helps to capture how the user interacts with the system and helps to reduce the user rejection of the software (Moreno et al., 2013).

For websites, usability is a "necessary condition for survival" (Nielsen, 2012) as users will easily leave the site and use another when encountering difficulty as this action has essentially no cost. Due to the low cost for access (Mariage et al., 2006), the acceptable usability threshold for a website is extremely high such as most users abandoning sites taking longer than 3 seconds to load (Kirkpatrick, 2016).

For ecommerce sites, the usability of their website has a direct affect on their bottom line. The first law for ecommerce sites is that if users cannot find a product, they cannot buy it (Nielsen, 2012). Other studies have shown that good usability redesigns can improve all positive website metrics such as visitor count, sales and user performance (Nielsen, 2003). Since the ROI from usability scales with the number of users using the system, this can be a very effective and relatively cheap way of improving business.

Although, usability is a well known important aspect of software it is often considered as more of an after thought (Ferré et al., 2001). In the open source community, which develops a huge amount of software, there are very few contributors with usability experience (Nichols & Twidale, 2002). This often leads to issues with usability and the end software being difficult to use. This could be because commercial software has the luxury of being able to hire additional people with usability experience or because open source software usually targets users with a higher level of technical skill than average.

3.4 Evaluating Software Usability

There are many different methods to successfully evaluate software usability. What choice to use depends on what data you want to get.

Lin et al. (1997) proposes a usability index to improve how we compare the usability of different software. Currently the concept of "usability" is very subjective and relies

heavily on the "evaluator's expertise".

Evaluation techniques for usability can be split into two groups; inspection methods and test methods. Inspection methods require usability experts to evaluate and review the usability across a system whereas test methods measure the "system performance against pre-defined criteria" (Avouris, 2001).

Inspection methods include Heuristic Evaluations, Cognitive Walkthroughs and Standards Inspections.

A popular test method for evaluating usability is the Thinking Aloud method. This is because it allows the end user to continuously say what they are thinking about a system, even down to really small specific UI items whereas other methods may require more of a high level summary of a system (Ferré et al., 2001). A variant of the Thinking Aloud method is the co-discovery method, which requires two, or more, participants to complete the task together. This often provides a more natural experience for the end users since people are more accustomed to talking to others and allows them to build on each others points (Holzinger, 2005).

3.5 Cloud Computing Usability

Almost all leading cloud computing providers support both graphical and Command Line Interface (CLI) (Sampath et al., 2021). The graphical user interface usually takes form as a modern SPA website and the Command Line Interface (CLI) as a piece of software that is manually installed onto the developers machine.

Although experienced cloud users are the most profitable for providers, cloud beginners make up nearly a third of total users (Silva et al., 2018). Cloud beginners are also more likely to choose a platform for its ease of use rather than additional features and functionalities (Kavya et al., 2021). This is important to remember as many providers prioritise developing new features and support for new technology instead of refining the existing UI/UX to improve the usability for beginners.

Experienced developers mostly prefer using the command line interface for performing tasks on the cloud as it often supports the best user performance and heavy personalisation (Murillo & Sánchez, 2014). This is unsurprising as many of the other tools they likely depend on are also terminal-based, such as version control and package management. Coleman et al. (2022) also reported that the chosen interface to use to complete a task isn't dependent on the experience of the user but instead on the task. For example, although more experienced users prefer using the CLI for CRUD operations on the cloud, monitoring is one aspect that experienced users generally prefer to use the web consoles for (Coleman et al., 2022). This could be because web consoles provide a range of graphs to quickly visualise data in a useful format. Coleman et al. (2022) also shows that many

cloud beginners are willing to use their cloud providers' CLI tool if it's the quickest way to complete a task. This follows the same pattern as to how developers use Git version control with most starting off by using the CLI and others starting off using the GUI and gradually move to using the CLI and many using a combination of the two depending on what task they are trying to complete (Olofsson & Hultstrand, 2015).

Interestingly, experienced users tend to rely on extra guidance or graphical interfaces when using a new tool or software and decrease their usage of the GUI as they become more experienced (Parameshwar & Yudelman, 2022).

Stanton et al. (2015) proposes a standardised usability framework to be shared by all cloud service providers. This would require all cloud service providers to follow the same models and deployment structures as each other which would reduce time spent onboarding an application to the cloud or time spent moving from one provider to another and would reduce training requirements and technical help. This would also give greater freedom for companies and users of these platforms to "shop around" to reduce costs and use the best-suited tools for a job without the fear of becoming locked into a vendor just because it's the easiest or only option. Although Terraform sometimes requires additional cloud service provider configuration, it is one of the closest options for quickly switching between providers and linking resources. It also follows the same pattern for deploying resources on different cloud service providers, which makes it better than some similar cloud provider-specific alternatives such as Azure Bicep.

Buyya et al. (2019) notes that one of the poorest areas of the usability within cloud computing is cost prediction and management. This means that it is often difficult to get an accurate estimation of cloud computing costs as values can fluctuate on a daily basis and it is often difficult to find out exactly what you will get charged for as storage, data transfer, hardware specs and monitoring/analytic software is all calculated separately and options for one may change the others. For example, more detailed monitoring or analytic software will use more storage space but how much is difficult to say. This could potentially indicate that a key aspect of cloud computing usability is clear and easy to understand pricing.

Another concern is that users are often "overloaded with resource and service types available to run their applications" (Buyya et al., 2019). This is especially true for the primary CSPs which all host hundreds of different resource types and this could also partially explain the success of some simpler cloud service providers such as Heroku and DigitalOcean. These smaller providers focus on basic resources such as databases, storage and platform as a service application hosting but don't try to compete with the likes of Microsoft and Google on the machine learning/artificial intelligence fronts. Potentially

this could indicate that for my software, I should focus on the most commonly used resources for each provider instead of trying to support all possible resources for a provider as these will be the most commonly used and will have a greater chance of providing help for the end user.

3.6 Conclusion

Overall, the research shows that the usability of many CSPs is at a high standard apart from a few specific aspects such as cost management & prediction, cross provider support (vendor lock-in) and an often overwhelming number of resource types & configuration options.

The research also indicates that my software could provide help to both beginner and experienced cloud developers as long as it provides a quicker and more convenient solution than the currently available options. This is because both groups prefer to use the command line apart from in cases where a graphical option is quicker.

Chapter 4

Software Requirements

4.1 Introduction

4.1.1 Purpose

The purpose of the software is to make it easier and quicker to develop complex cloud infrastructure. This will lower the barrier to entry for many to cloud computing.

The purpose of this document is to outline the features my software needs to have. This will act as a plan so that I can spend my time more effectively developing the key areas of my web application and reducing the time spent on the less important areas.

4.2 Overall Description

4.2.1 User Needs

The user of this software will probably be new to cloud computing yet understands that they have an issue it can help with.

The user could also be someone more experienced who just wants a quick way of building a configuration that they can modify later.

The very end user of this would be someone who is using software which is deployed or managed on the resources that this tool generates infrastructure as code for. As long as the IaaC being generated is usable for the software and all of its components, they should not be able to notice any difference. If the IaaC was invalid and used, this could result in parts or features of the software breaking or functioning differently.

4.2.2 Assumptions and Dependencies

Assumptions:

- User has a good enough understanding of what resources are required for their situation. This includes networking capabilities. User is able to make use of resources through their software on their own. This software will aid in the provisioning

and maintenance of cloud resources but will not provide help with connecting to or using the resources from within the user's software. There are already a lot of resources online about how to do this in a range of different languages.

Dependencies:

- This software will depend on Terraform and cloud service provider plugins.
- Front end - NPM Packages:
 - Typescript
 - React
 - Redux + Redux Toolkit
 - Tailwind CSS + Headless UI
 - Cypress, Axe, cypress-axe
- Backend - NPM Packages:
 - Typescript
 - Express
 - Chai, Mocha & Supertest

4.3 Existing and Similar Software

4.3.1 Stack Lifecycle Development

<https://github.com/D10S0VSkY-OSS/Stack-Lifecycle-Deployment>

Stack Lifecycle Development is an open-source project on GitHub and is the most similar piece of software to what I plan on developing. It is still fundamentally different as it is used to manage different environments by controlling environments and scheduling and has a rest API to automate Terraform management. It allows for the management of different environments by using Terraform under the hood and generates dynamic forms for Terraform resources and variables created by the user.

Although it has a focus on management and can only generate dynamic forms, it doesn't offer much of a difference from filling out key-value pairs in a Terraform file. It also provides no help for determining the correct value for a resource property and what the possible values include.

4.3.2 ChatGPT

ChatGPT can be used to generate many things, including code. In particular, ChatGPT seems quite good for generating "stand-alone" code, such as a function or two, to complete

a small code problem. I decided to test it to generate a couple of Terraform files defining a few simple Microsoft Azure resources, as seen in Appendix D.

There were a number of issues with ChatGPT generating Terraform files:

1. ChatGPT often uses deprecated resources that either don't work well and have as much support as preferred alternatives or will be removed in a future update. In figure D.1, ChatGPT used "azurerm_app_service_plan" and "azurerm_app_service" instead of the newer "azurerm_service_plan" and "azurerm_linux_web_app".
2. ChatGPT also seems to get the resource properties mixed up with other resources. For example, in figure D.2, 'sku' is not a valid block in an 'azurerm_service_plan' resource.
3. Relationships between resources may not always be correct. In figure D.2, "azurerm_app_service_plan" cannot have a "azurerm_linux_web_app" resource within it.
4. Ultimately crashes and refuses to accept any more prompts as seen in figure D.3.

Technologies such as Terraform are very strict when it comes to the validity of a configuration. Because of this, ChatGPT isn't a suitable tool for generating short snippets of code.

Since you must guide ChatGPT to make changes to ensure the content is valid, you still need to understand what resources and configurations you need. You may also need to suggest multiple changes for the Terraform configuration to be correct, so you spend more time doing this than using official documentation and help.

Ultimately, ChatGPT and its technology is still in its infancy and will undoubtedly improve in the coming years, especially as it uses more training data and the training data becomes more recent, which is part of the issue here due to the frequent updates of Terraform, and its providers.

4.3.3 Gaia & Terraboard

Both of these provide UIs to improve the user experience when it comes to specific aspects of Terraform, but neither makes it easier to create resource blocks.

Gaia is a self-hosted UI to import and run Terraform modules but doesn't help with developing the modules or resources in the modules.

Terraboard is used to visualise Terraform's state data.

4.4 Study

I needed to conduct a study to identify better my requirements for my artefact and which areas I would need to focus my limited time on. It would also help to see if there was actual demand for the application I planned on making.

4.4.1 Co-Discovery Study

I originally planned to conduct two co-discovery studies, each with two participants. These participants would include two people with limited cloud computing knowledge and no or nearly no experience with Terraform. They would then try to complete a number of small tasks such as provisioning, editing and deleting resources using Terraform and Microsoft Azure. I would then ask each participant several questions about the experience, what was easy and difficult about it, whether they would consider using it for one of their own projects and whether they could see a drag-and-drop web application as a tool that would make this process easier.

After I developed my artefact, the co-discovery experiment would be repeated, but this time using my web application instead. The participants would then carry out similar tasks and then I would again ask them questions on how they found the experience to be and use their feedback to evaluate the success of the project.

Unfortunately, neither of these studies could be carried out since I could not find two volunteers to complete this hour-long study. I used various methods to advertise the study and reach out to people, including using LinkedIn, Twitter, and university discord channels. I believe a core reason for study recruitment difficulties was because of the technical topic of the study

The plan of the initial co-discovery study can be found in Appendix C.

4.5 Questionnaire

My fallback option was a short questionnaire. The questionnaire asks the user about their experience with cloud computing and demonstrates a short Terraform script, and questions them about it. Testing usability through a survey is not ideal, but I tried to get around this by asking the user questions about how well they can understand what the script does and what they would do in order to modify its functionality.

The success rate of the survey was much better, but I still found recruiting participants difficult. Again, I used similar methods to reach as many students as possible such as social media, discord messages and a small email campaign to target people on the most relevant university courses (250 students).

The "cloud computing" topic of the survey might have discouraged some potential participants from completing it, as it isn't relevant or interesting to everyone. They might have

felt others would be better suited to answer the survey.

Although I only managed to receive nine responses in total, much lower than my goal of 30, the data the questions produced showed some clear patterns.

4.5.1 Q1 - Have you had any experience with Cloud Computing Providers such as AWS, Azure or GCP?

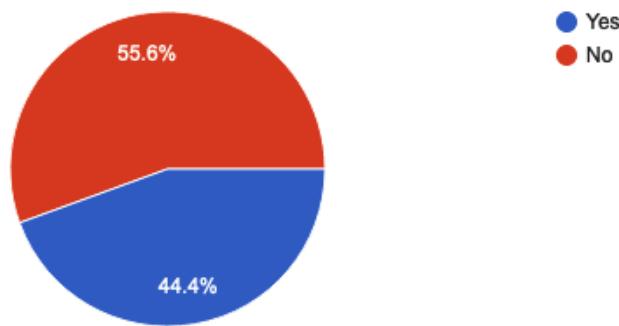


Figure 4.1: Q1 - Have you had any experience with Cloud Computing Providers such as AWS, Azure or GCP?

From the questionnaire results, most people haven't had any experience with cloud providers (55% - 45%). This was good for the survey as I was trying to get participants with little to no cloud experience.

4.5.2 Q2 - How confident are you about working with the cloud?

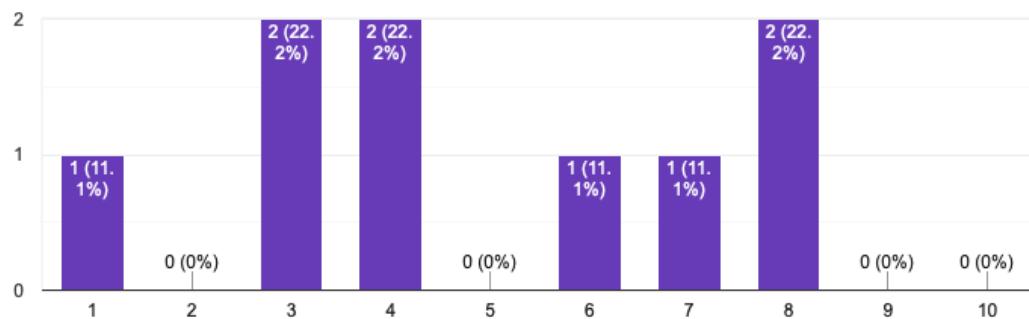


Figure 4.2: Q2 - How confident are you about working with the cloud?

As you would probably expect from the first question (figure 4.1), this meant that most people were not confident working with the cloud.

Although 2 people rated themselves 8 out of 10 and one person 7, which both seem fairly high.

4.5.3 Q3 - Have you had any experience with Terraform or any other Infrastructure as Code tool?

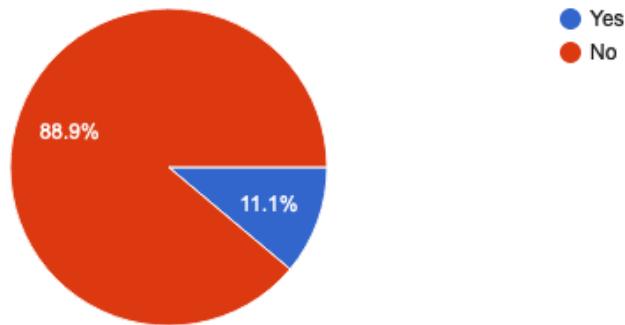


Figure 4.3: Q3 - Have you had any experience with Terraform or any other Infrastructure as Code tool?

Question 3 asked about the participant's previous experience with infrastructure-as-code (IaC) software. Only one participant had any prior experience, which was actually quite surprising. The same participant was the only one to get the next 3 questions correct.

4.5.4 Q4 - For the above screenshot: What is the name of the Microsoft SQL Server resource when provisioned on Azure?

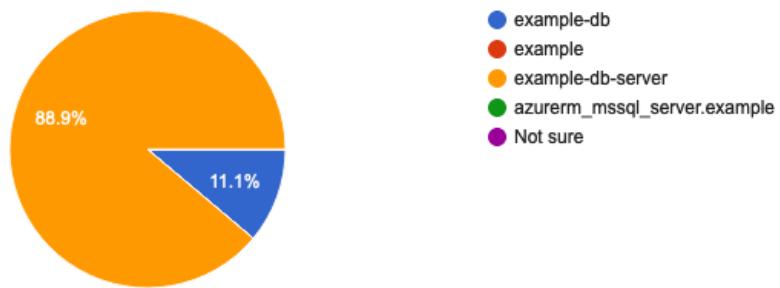


Figure 4.4: Q4 - For the above screenshot: What is the name of the Microsoft SQL Server resource when provisioned on Azure?

This question was to evaluate how easy it is for someone to read and understand a simple Terraform file.

Only one person got this question wrong.

4.5.5 Q5 - For the above screenshot: If you were to change the location of the resource group, would it change the location of the database?

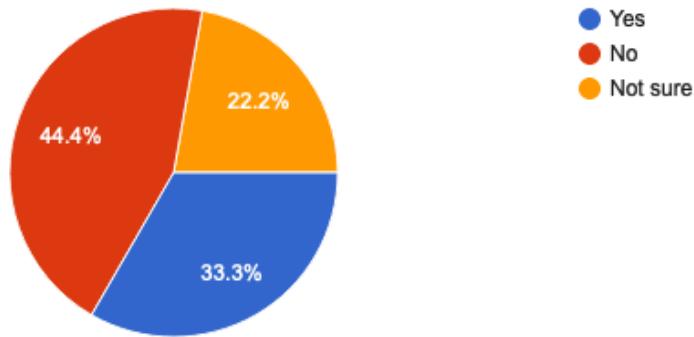


Figure 4.5: Q5 - For the above screenshot: If you were to change the location of the resource group, would it change the location of the database?

This question had a good split of different answers, and so indicates that the Terraform configuration is fairly difficult to understand.

Only 3 people correctly answered the question with the "Yes" option.

4.5.6 Q6 - For the above screenshot: If you wanted to use the database server's name as the name for the database, what would you use?

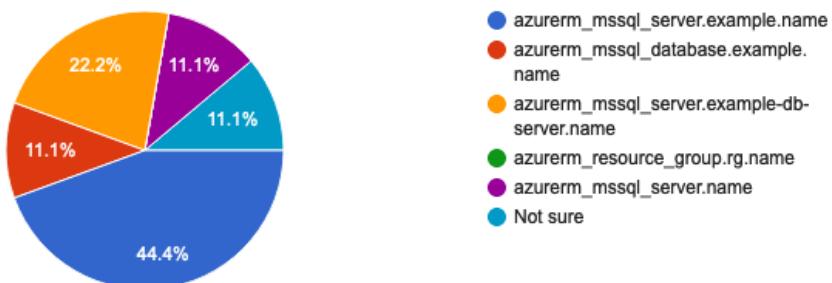


Figure 4.6: Q6 - For the above screenshot: If you wanted to use the database server's name as the name for the database, what would you use?

For the last exam-style question, 4 people answered correctly. There was again a range of other answers, including 1 "not sure".

4.5.7 Q7 - How easy do you think it is to understand the configuration of each resource and how they connect together?

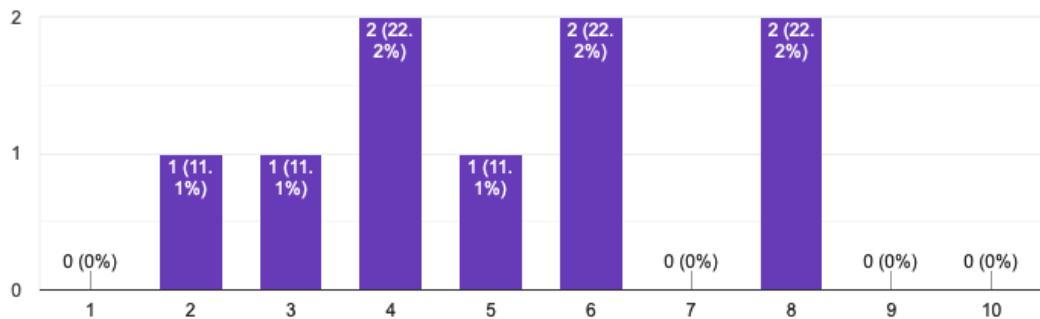


Figure 4.7: Q7 - How easy do you think it is to understand the configuration of each resource and how they connect together?

Now that each participant has had a chance to look at a Terraform file and think a bit about how it is structured and what it can do, this question was to ask them to evaluate how easy it was to understand the configuration and the relationships between the resources.

Two people answered confidently with an 8 out of 10, while most ranged between 2 and 6 out of 10.

4.5.8 Q8 - Looking at the above example, what issues could you imagine if you had to work on this file?

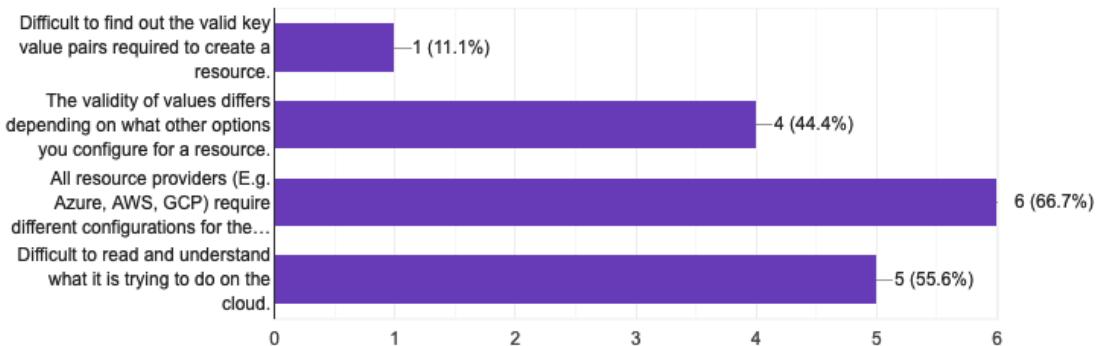


Figure 4.8: Q8 - Looking at the above example, what issues could you imagine if you had to work on this file?

This multi-select question was to try to gauge what issues the participants could predict when it came to working on more difficult and larger Terraform files.

The most popular issue identified by survey participants is that all cloud providers require a different Terraform config for the equivalent resource.

The second most popular answer was difficulty understanding the Terraform configuration and what it would do on Azure.

4.5.9 Q9 - How useful do you think a visual node-based editor would be for managing Terraform files?

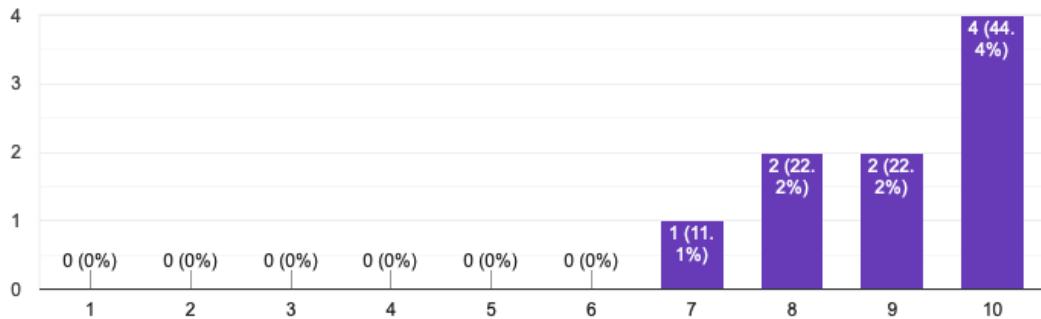


Figure 4.9: Q9 - How useful do you think a visual node-based editor would be for managing Terraform files?

This was one of the most important questions in this survey, as it would really help to validate my core idea. All of the responses support my idea, and 44% rated the usefulness of such an editor as 10/10.

4.5.10 Q10 - What 5 features would be the most important to support?

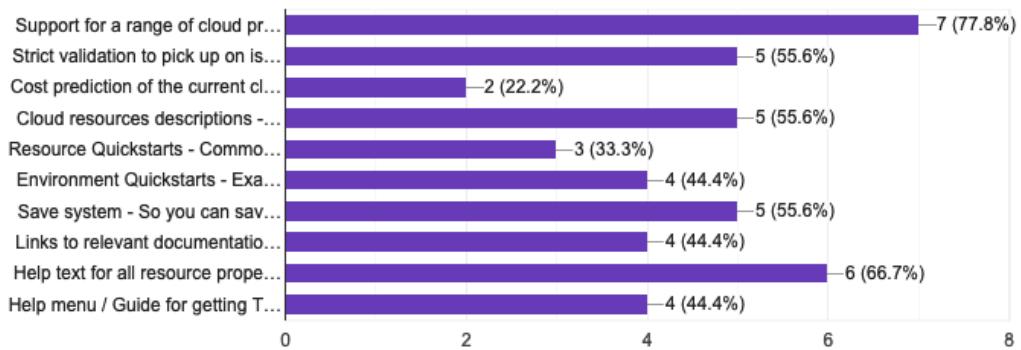


Figure 4.10: Q10 - What 5 features would be the most important to support?

This question was designed to help gauge the core features and functionality the application should have. The 5 most commonly picked options were:

1. Support for a range of cloud providers - This option was the most popular. It would be the most difficult to add as each cloud provider has a huge number of resources, with many of them supporting complex configurations.

2. Help text for all resource properties - This would give additional information to the user about what the resource option does and the differences between the possible values.
3. Strict validation to pick up on issues early - This would help to save time by identifying issues with the user's configuration before generating the HCL code and using the Terraform command line tool to plan or apply it. This feature would use the Error Prevention usability heuristic as it would prevent users from encountering errors with the Terraform CLI software by informing them earlier of issues (Nielsen, 2005).
4. Cloud resources descriptions - This would explain what each cloud resource is used for to make it easier for the user to decide what they need to use instead of having to manually search for each resource on the cloud service providers' documentation page.
5. Save system - This would allow the user to view and modify previous configurations without having to recreate them each time manually. This could either use a file save system or a database save system. A file save system would give the user a JSON file to save, which could then be uploaded to the web application to change what configuration is currently loaded. The database system would save the same information within a database that the user could load through the web application, although this would require authentication and additional work to add.

4.5.11 Q11 - Are there any issues you could see with such an editor?

Screen clutter
Overload of information on new users
The UI could potentially take up a lot of a computers resources when running the editor.
Lack of control over the underlying resources (sometimes you'd want to manually edit - either in the interface or as an extension). Would this automatically update along with Terraform?
I think scalability is a concern here. If you have multiple services I think it might become very complex to navigate. A search function would greatly help I think.
I am not an expert myself, so it looks like it doesn't have any issues to me.

Figure 4.11: Q11 - Are there any issues you could see with such an editor?

The participants identified the following possible issues, and I have identified possible ways to reduce them:

1. "Screen Clutter"
 - (a) Use collapsible boxes for users to expand or collapse.
 - (b) Provide an option to show or hide additional information some users may require at times.
2. "Overload of information on new users"
 - (a) Provide a basic getting started guide with Terraform and Cloud Providers.
 - (b) Provide a tutorial guiding the user on how to use the website.
3. "The UI could potentially take up a lot of a computers resources when running the editor"
 - (a) The editor will be built using a modern React + Redux website. The computer resources used while running the website won't be much higher than any other website.
4. "Lack of control over the underlying resources (sometimes you'd want to manually edit - either in the interface or as an extension). Would this automatically update along with Terraform?"
 - (a) Adding functionality to manually edit the Terraform files and update the nodes and their configurations accordingly.
5. "I think scalability is a concern here. If you have multiple services I think it might become very complex to navigate. A search function would greatly help I think."
 - (a) Adding some search functionality could improve how easy it would be to navigate larger configurations.
 - (b) A Command Palettes component could allow users to find and navigate to specific resources quickly. It could also be used to create new resources quickly.

4.5.12 Questionnaire Results Conclusion

The questionnaire significantly helped identify requirements for the software design and would work as a solution. It also helped prove that reading and understanding Terraform files is difficult for beginners, and a graphical Node UI would provide a better experience. The survey also helped identify what features and functionality the potential users of the application would find the most useful and important to implement.

It also allowed me to start thinking about some of the pitfalls that could occur with such a design and to think of additional ways to alleviate them. As identifying and solving these issues now will be much easier than during development.

4.6 System Features and Requirements

These are the requirements I have identified for the project. These requirements have been identified from the literature review, questionnaire and the observation of current cloud computing tools like Terraform and cloud providers.

There are a number of different prioritisation methods for ranking software requirements. For this project, I chose the MoSCoW method as it works well for simple projects and categorises requirements into non-negotiable categories (Hatton, 2008).

Each requirement will be put into one of the following categories:

- Must-have: An essential feature that the project must have to support basic use.
- Should-have: A feature that provides much-needed functionality.
- Could-have: A nice-to-have feature which would improve the software but isn't essential for functionality.

4.6.1 Functional Requirements

Functional requirements are "requirements which specifies what the system should do" ("Functional vs Non-Functional Requirements - Understand the Difference", 2012). These will include the features that the user is able to perform.

ID	Description	Priority
F1	Present a filterable list of available resources for the user.	Must-have
F2	Allow the user to create an instance of a resource which can then be edited.	Must-have
F3	Provide information about available configurations for a specific resource.	Must-have
F4	Generate the code used to provision the selected resource with the selected configurations by the user.	Must-have
F5	Allow the user to reference other attributes from a resource.	Should-have
F6	Allow the user to add optional configuration options.	Should-have
F7	Allow the user to reference object type variables.	Should-have
F8	Inform the user of any mistakes or invalid configurations.	Should-have
F9	Provide pricing information and estimate to help inform the user on expected costs.	Could-have
F10	Provide links to other relevant documentation, such as from Terraform and the cloud provider's official documentation, as a shortcut in case the user needs additional information.	Could-have
F11	Allow the user to save the current configuration so it can be edited at a later time.	Could-have
F12	Provide a quick start option to build common resources quickly.	Could-have

Table 4.1: Functional Requirements Table

4.6.2 Non-Functional Requirements

Non-functional requirements are "requirements that specifies how the system performs a certain function" ("Functional vs Non-Functional Requirements - Understand the Difference", 2012).

ID	Description	Priority
NF1	Provide a modern visual web application for users to interact with. (Under 2 seconds load time)	Must-have
NF2	Generates valid Terraform HCL code.	Must-have
NF3	A reliable, well-built app with a high level of quality. This means that the app should act predictably and work well regardless of how the user uses it.	Must-have
NF4	Provide a greater level of abstraction for the user to make it easier to work using the cloud without having to spend a lot of time learning about it.	Must-have
NF5	The web app must scale appropriately to cover standard desktop sizes.	Must-have
NF6	The web app should comply with WCAG 2.1 AA accessibility standard	Should-have
NF7	Communicate securely with cloud provider APIs to get information about valid options for resources, e.g. the available VM sizes at a specific data centre.	Could-have
NF8	Support generating ‘.tf.json’ files and ‘.tf’.	Could-have

Table 4.2: Non-functional Requirements Table

Chapter 5

Design

5.1 Introduction

This web application will have 2 main components. A React client and a backend API. The user will interact with the React client to select and edit cloud resources. This data will then be sent to the backend API to generate the supporting HCL code.

5.2 Client

For the client, I have chosen to develop a single-page application (SPA) with React and Redux. This is because it needs to be cross-platform, and the page will be very dependent on what the user does and will have to update accordingly quickly. For example, when the user adds a resource to their selection. For this reason, using a multi-page application (much like a normal website) didn't make sense due to the large amounts of data needing to be sent between each page load.

I also considered developing a traditional desktop application using a framework such as WPF or Avalonia but decided this would have a slower development speed and would provide a more annoying user experience due to the required download and installation to access and more difficult to update. Developing a website is quicker and allows for near-instantaneous access. It is also much quicker to update, which I thought was important due to the fast-changing cloud computing scene, which I would have to try to account for.

5.2.1 Figma Mock Ups

Initial Idea

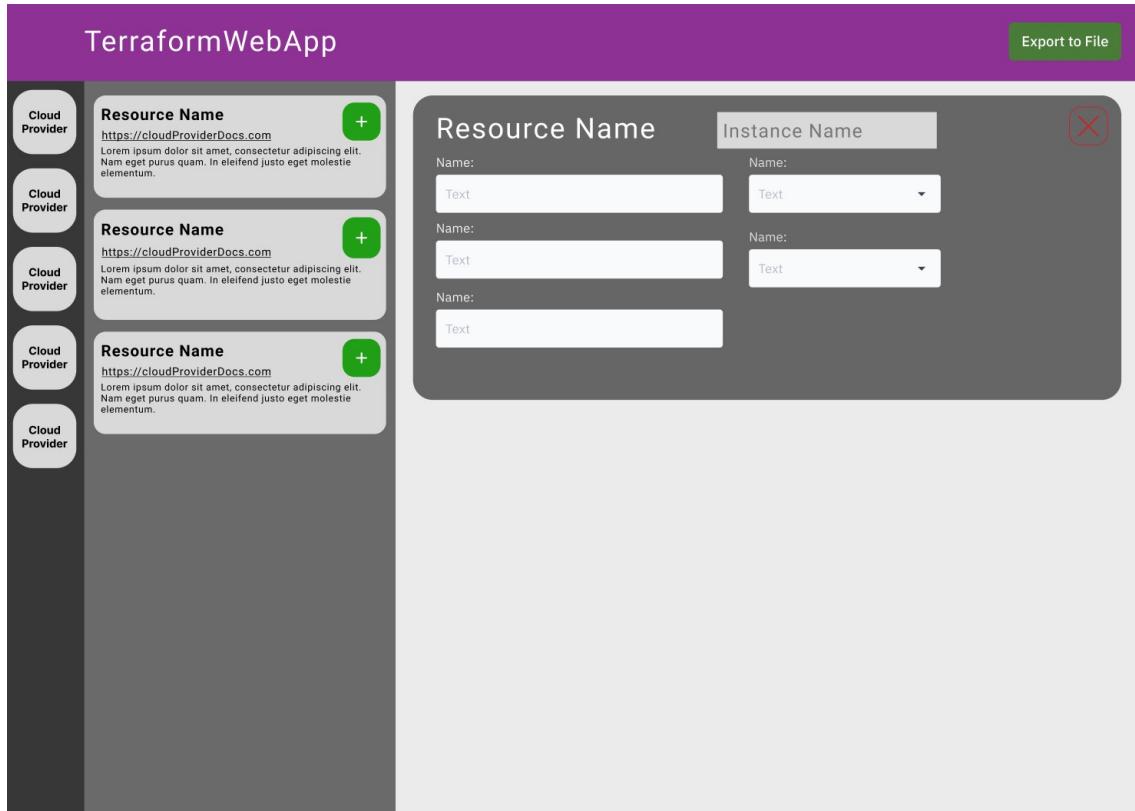


Figure 5.1: Initial design using a dynamic form

My initial idea for the web application was dynamically generated forms depending on the resources the user wants to create in the cloud.

Figure 5.1 shows a basic design with 3 columns.

1. Column 1 contains a list of buttons that filter the selected cloud provider for the listed resources in the second column.
2. Column 2 contains a list of resources for the selected cloud provider. Each resource element contains the resource name, URL to the official documentation, a small description and a green button to add an instance of the resource to the third column.
3. Column 3 contains a list of dynamically generated forms based on what resources the user has selected in the second column. It also includes an input for the instance name, the unique name for the resource within the Terraform file, and a deletion button to remove the form and the resource from the user's selection.

While feedback for this design and the initial version of the prototype (7.2.1) was good, it seemed that I was designing something too safe and too close to a Terraform file. Using it

didn't provide much of an advantage compared to developing a Terraform configuration within a normal 'main.tf' file.

Revised Idea

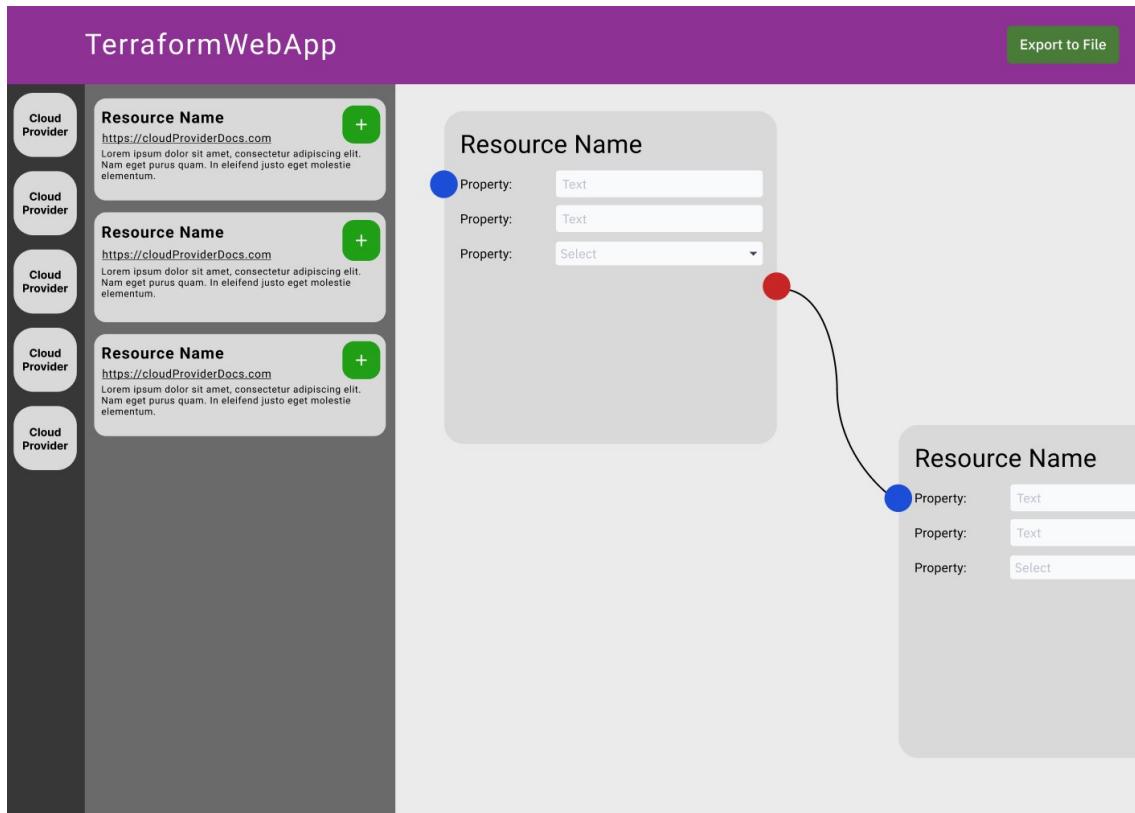


Figure 5.2: Revised design using dynamic nodes to represent resources.

Figure 5.2 shows my revised design using nodes to represent resources. I revised my design based on early feedback suggesting I try a more challenging approach to this problem instead of dynamically generating and filling in forms. Using nodes should offer a better perspective of seeing how resources connect to one another and allow for more dynamic and efficient configuration.

5.3 API Server

I have chosen to develop the API server with NodeJS and the ExpressJS web framework. This is because it is quick and easy to use and develop. It will also easily work with the JSON that will need to be sent to generate the Terraform code.

I considered other frameworks such as ASP.NET core WebAPIs as I have used them before but decided against it due to its slower development speed, and the performance gains over Node and ExpressJS wouldn't be noticeable due to the low request rates the prototype will be making.

I am also very confident with the client and API server libraries and frameworks, and so

it will reduce the amount of learning, I'll have to do during development.

5.4 C4 Diagrams

C4 Model is a diagramming technique for modelling the architecture of software systems (Richards & Ford, 2020). It helps development teams describe and communicate systems better and provide different levels of detail (Brown, 2022).

The name refers to the 4 C's that make up the diagramming system:

1. Context - Provides a high-level overview of the system and how it interacts with other systems and the world.
2. Container - Zooms into one system and shows the containers that make up the system.
3. Component - Zooms into one container and shows the various components that make up the container.
4. Code - Zooms into one specific container and shows a UML diagram of the component.

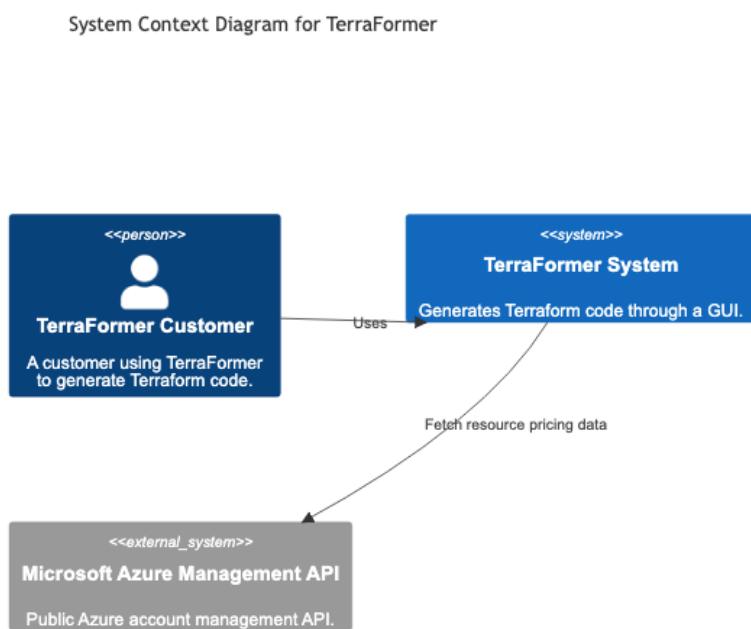


Figure 5.3: C4 Level 1: System Context Diagram for TerraFormer

Figure 5.3 shows the customer interacting with the software and the software interacting

with an external system, Microsoft Azure Management API.

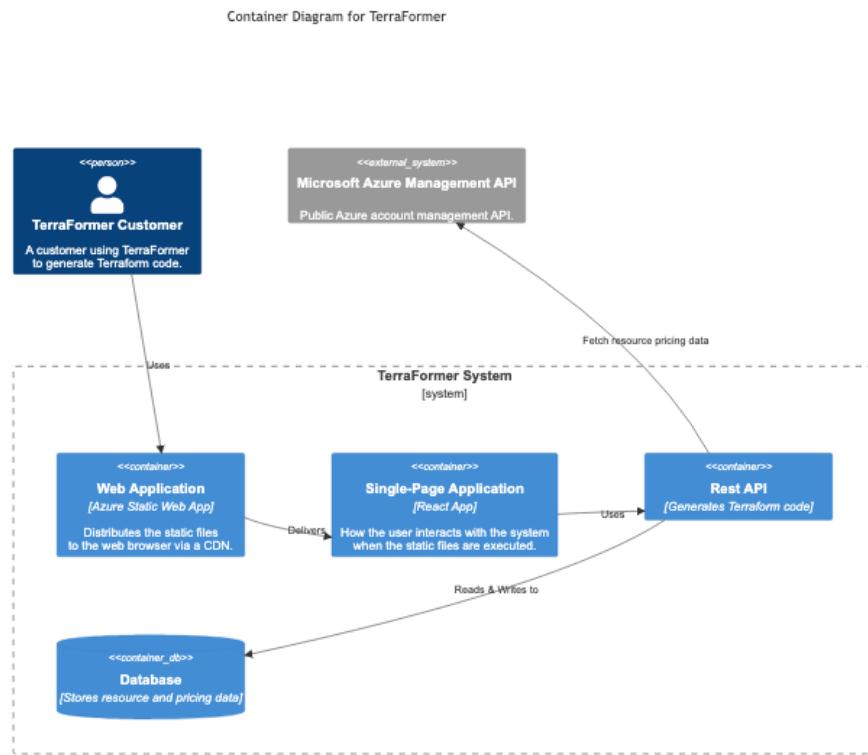


Figure 5.4: C4 Level 2: Container Diagram for TerraFormer

Figure 5.4 shows the project system in more detail. This view shows how the client is delivered to the web browser and interacts with the REST API backend server. It also shows that the backend server sends requests to Microsoft Azure Management API and reads/writes data to a database.

5.5 DevOps & Deployment

I am also planning on automating the entire deployment process using GitHub Actions. This will automate the building, testing and deployment of the software, which will help ensure that the final product will meet the requirements and can be run on the "production" cloud resources (Machiraju & Gaurav, 2018). I will also use continuous integration and continuous deployment (CI/CD) to automatically test and deploy my software.

Continuous integration means that after every commit to the GitHub repository, an Action will automatically build and test the application. The build and testing process is standardised, so the entire process is reproducible and will deliver the same result each time (van Merode, 2023). Testing the code in a production-like environment as soon as

a change is introduced allows us to identify bugs or testing issues quicker (Gene et al., 2021).

Continuous delivery means there will "always be a stable mainline of the code" suitable for production deployment (van Merode, 2023). Over time, as other branches are tested and merged into the stable branch, it will automatically deploy the latest version to the production server.

5.5.1 Client

I will use an Azure Static Web App to deploy the React client. This will have a close relationship with the GitHub repository I am using and will automatically deploy the latest version of the master branch to the cloud when changed through an automatic setup GitHub action.

5.5.2 API Server

The API server will be hosted on an Azure App Service instance, a platform as a service resource that makes it very quick to deploy a NodeJS application. This, again will be linked closely to my GitHub repository so that it is automatically deployed through GitHub actions when the primary branch has been updated.

Chapter 6

Project Management

6.1 Introduction

This chapter documents how I manage the project during development. This is different to the methodology chapter, which is more theory-based.

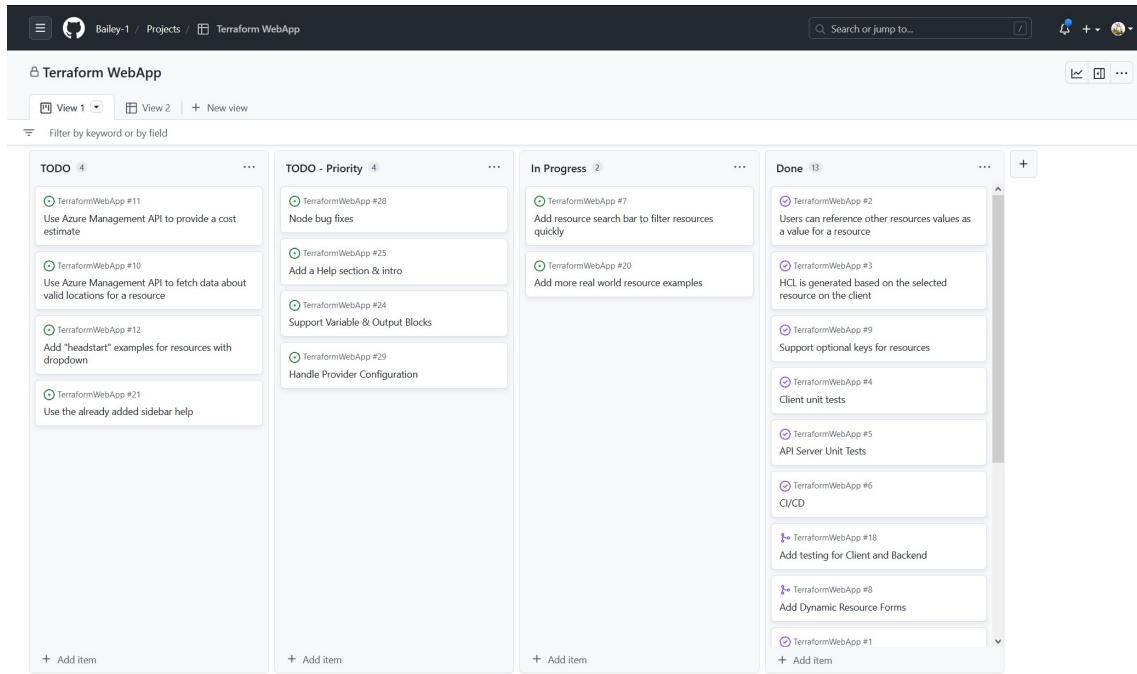
6.2 Kanban Boards

To manage the development of my project more easily, I used a Kanban board and divided the project into smaller, more easily achievable blocks. These smaller tasks are not the same as the project's functional or non-functional requirements and are more focused on the actual development of the application.

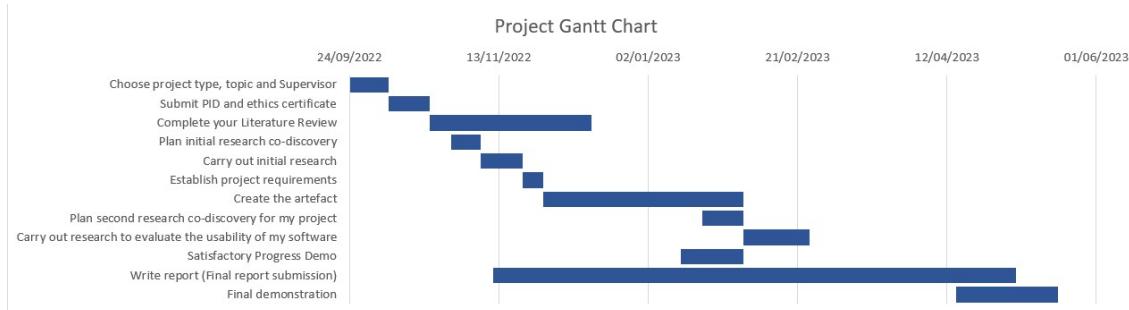
The Kanban board has 4 different columns:

- TODO: Tasks that I would like to complete, although not essential for the software to function.
- TODO - Priority: Tasks that I need to complete in order to have basic functionality working.
- In Progress: Tasks linked to a currently in a development pull request.
- Done: Completed tasks with all changes merged into the Master branch.

When working on a specific Kanban task, I will create another git branch for the feature to store commits and updates to the application. I will then open a Pull Request to the master git branch on GitHub. A pull request may contain changes to complete multiple tasks if they are closely related, for example adding testing support for the backend and adding continuous integration changes to run the tests automatically.

**Figure 6.1:** GitHub Kanban Board

6.3 Gantt Chart

**Figure 6.2:** Project Gantt Chart

The Gantt chart in figure 6.2 shows my project's intended plan and structure.

There were several delays with the original plan, which delayed further parts of my project.

Firstly, the literature review took longer than expected because of difficulties finding relevant sources and articles. To get back on track for this task, I spent additional time working on it and increased the scope of the review from just "cloud computing usability" to "software usability" and "cloud computing usability". This increased what I could discuss in the review and helped me identify better trends regarding developer software usability.

Since I wasn't able to carry out my initial co-discovery research, this caused a delay in

establishing my requirements and then starting the development of the software application. To minimise how much this would affect my project, I used my fallback option of a survey to ensure that I could at least get some useful data about my planned project and the direction I wanted to develop it in.

The "Plan second research co-discovery" and "Carry out research to evaluate the usability of my software" tasks couldn't be carried out because of study recruitment issues and so were skipped. This time was spent advertising and recruiting for my questionnaire instead.

6.4 Weekly Supervisor Project Meetings

Throughout this project, I regularly met with my project supervisor weekly to ensure that the project and this report remained on track. It also helped with project and report planning as it ensured I always knew what I was working on for the next week and helped with the pacing of work to ensure work was spread evenly over the year. Regular meetings also provided a point where I could ask for help and advice about dealing with unexpected project issues, such as problems with survey and study participant recruitment and what to try for the next week to circumvent these problems. This helped keep the project on track even with unforeseen issues.

Chapter 7

Development

7.1 Introduction

This chapter will explain how I developed the TerraFormer web application and the reasons behind some of my decisions. It will also explain the different iterations of my application.

7.2 Versions

To make the development of this software easier, I tracked significant milestones using git tags. This is so I can create a "release" for each tag that details changes in the version and makes it easier to see how this application evolves over time. It also makes it easier to see what functionality was added in what order, including functionality that is eventually removed or replaced.

Each release or version contains changes from one or more pull requests.

The screenshot shows the GitHub releases page for the repository "Bailey-1 / TerraformWebApp". It displays three releases:

- v1.1 - HCL Displayer & Testing** (Latest, 2 weeks ago):
 - Contributors: Bailey-1
 - Assets:
 - Source code (zip)
 - Source code (tar.gz)
- v1.0** (3 weeks ago):
 - Contributors: Bailey-1
 - Assets:
 - (empty)
- v0.3** (last month):
 - Contributors: Bailey-1
 - Assets:
 - (empty)

Figure 7.1: GitHub releases of TerraFormer

7.2.1 v0.1 - Initial two column view

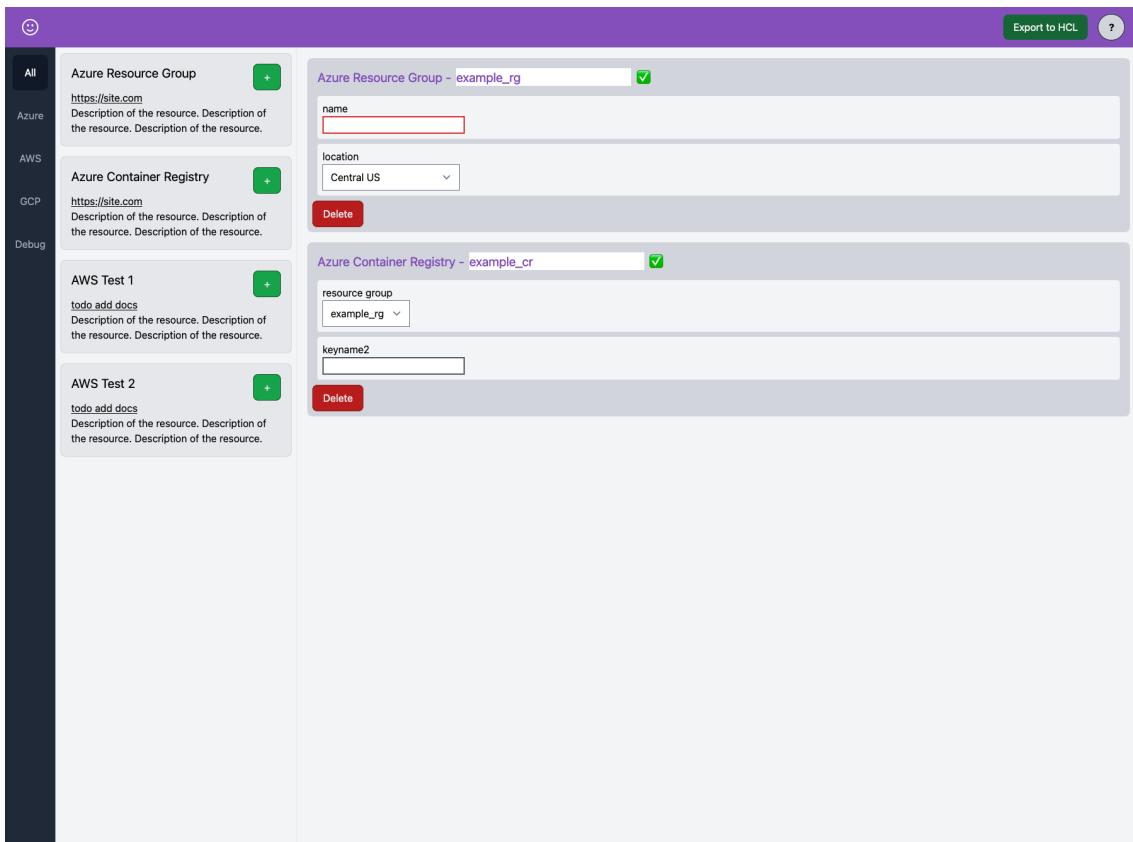


Figure 7.2: v0.1 - Initial two column form view

This was my initial idea for the application. It features two columns, left and right. The left column lists the available resources the user is able to add to their selection, and the right column shows the users' currently selected resources and their configuration. This allows the user to fill out each resource form to customise them easily.

One of the biggest issues I quickly faced during the development of the client was how to model the data for the different cloud resources correctly. I needed a way to create instances of complex resources which shared the same logic. For example, creating two specific virtual machine resources, each with their own unique properties but sharing the same logic for validation. Validation would consist of two different parts; form input and the overall form. It would also have to have some way of dynamically looking data up and work nicely with Redux for state management.

- Form input validation - is the validation for the user input on a specific field. For integers, this would include things such as a minimum and maximum number, and for text, it could check that only valid text characters were used.
- Overall form or the resource level validation - is the validation for the overall form of a single resource. This would check that the combinations of inputs are valid such

as ensuring that the user has only chosen premium features when using a premium tier or that the resource can be deployed to the selected region.

I also created dynamic links between resources so one resource could reference the property of another resource, such as a resource name. I also added validation to check that the resource instance name was unique, indicated by the green check mark or a red cross next to the field.

I chose this approach for the project as it was relatively simple and would be easy for me to develop, and so I couldn't foresee any big issues that could occur with it. Feedback from my project supervisor and moderator suggested that I should develop a more abstract interface that resembles less like an actual Terraform file.

7.2.2 v0.2 - Change to a Node-based UI

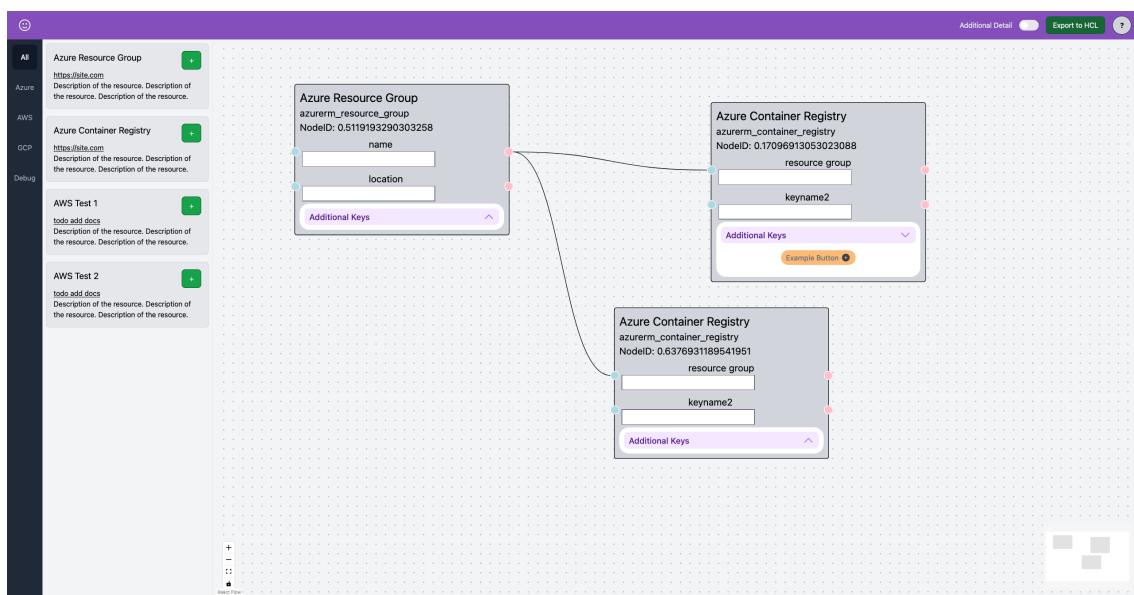


Figure 7.3: v0.2 - Change to a Node-based UI

In this version, I started to develop a node-based UI where each resource is a different node and can be linked to other resources to show how they all relate to one another. I decided to switch to this design as it would give a more abstract view of the resources and how they are configured in a Terraform configuration.

Since Terraform is very powerful and has a lot of different configuration options, one of the main issues I faced with this change is deciding how much to hide from the user and deciding which Terraform features to support. For example, resources have attributes which provide information on the resource instance. Some of these are known or set by the user and others are unknown until runtime such as IDs or IP addresses. Terraform allows you to reference these properties in other resources dynamically as variables so you

can link resources together easily. I plan on using the node line connections to represent these links to make it easier to see how resources work together.

Although, I haven't started working on generating valid HCL code from the configuration.

One aspect of this I found challenging to design was designing a dynamic system for generating each resource node without having to manually design each one and duplicate a large amount of code. This means that each resource UI node is given a large and detailed object which contains all the information required for the node to be rendered and used by the user. An example is shown in 7.1.

Listing 7.1: The resource object interface

```

1 export interface IResourceObject {
2     name: string;
3     display_name: string;
4     provider: string;
5     docs: {
6         terraform: string;
7         provider: string;
8     };
9     validation?: () => boolean;
10    keys: IResourceKeys[];
11    attributes: string[];
12    description: {
13        small: string;
14    };
15    tags: string[];
16 }
```

The ‘keys‘ object property is an array that defines each of the properties the resource has, such as a name, VM size and tags. The form inputs on the resource node updates automatically to reflect this current array. This key accepts a few different input types, such as resource, string, and select.

Example 7.2 shows one of these types.

Listing 7.2: The IResourceKey object interface

```

1 export interface IResourceKey {
2     name: string;
3     display_name: string;
4     type: 'string';
5     value: string;
6     validation?: (value: string) => boolean;
7     validation_message: string;
```

```

8     required: boolean;
9 }

```

7.2.3 v0.3 - Block Nodes & Styling Improvements

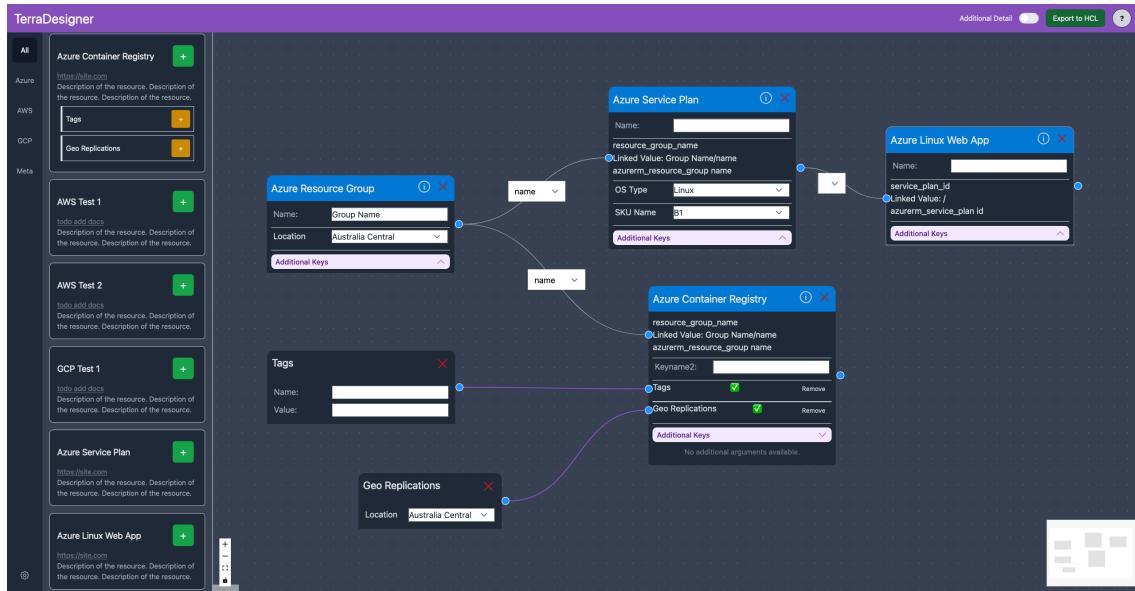


Figure 7.4: v0.3 - Block Nodes

In this version, I started to develop block nodes. These are different from the resource nodes as blocks represent an additional configuration object a resource might require, such as tags or a network interface.

In the code example 7.3, it shows 3 blocks inside of a resource block.

Listing 7.3: An example of block values within an Azure Kubernetes Cluster Terraform Resource

```

1 resource "azurerm_kubernetes_cluster" "example" {
2   name = "example-aks1"
3   location = azurerm_resource_group.example.location
4   resource_group_name = azurerm_resource_group.example.name
5   dns_prefix = "exampleaks1"
6
7   default_node_pool {
8     name = "default"
9     node_count = 1
10    vm_size = "Standard_D2_v2"
11  }
12
13  identity {
14    type = "SystemAssigned"
15  }

```

```
16  
17     tags = {  
18         Environment = "Production"  
19     }  
20 }
```

Designing this was challenging as the existing resource property reference system wouldn't work for it as a resource block is not a property but a nested object within the resource HCL block.

7.2.4 v1.0 - Prototype Works

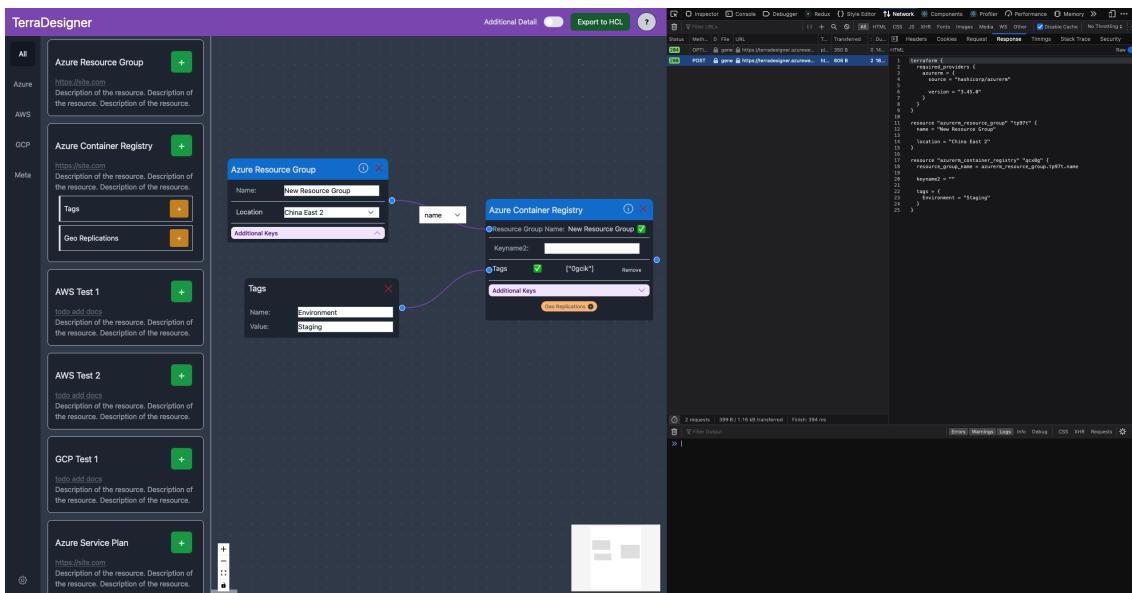


Figure 7.5: v1.0 - Prototype Works

I further developed the backend to generate HCL based on the request body data. The request requires details about the current nodes and the connections between them. It will then loop through each resource and map over the keys and values in order to generate a valid Terraform HCL file.

The resulting HCL can also handle dynamic resource attribute references and block node references such as tags.

One of the biggest issues with generating HCL is that JSON to HCL (or vice versa) isn't completely compatible. Terraform also uses a slightly modified version of HCL for Terraform keyword blocks, which means I had to use Regex to handle these issues, such as removing quotes from specific Terraform keywords and the equals symbol from mistakenly assigning values to Terraform keywords. The example below shows the Terraform block used to control which provider packages it uses and how it has been fixed to be valid.

Listing 7.4: Before my Regex fixes

```
1 "terraform" = {  
2   "required_providers" = {  
3     "azurerm" = {  
4       "source" = "hashicorp/azurerm"  
5  
6       "version" = "3.45.0"  
7     }  
8   }  
9 }
```

Listing 7.5: After my Regex fixes

```
1 terraform {  
2   required_providers {  
3     azurerm = {  
4       source = "hashicorp/azurerm"  
5  
6       version = "3.45.0"  
7     }  
8   }  
9 }
```

I also deployed the backend API on an Azure Web App instance which the already deployed client can use.

I tagged this version as 1.0 because it is the first version where the prototype technically works, although it still requires work to add additional functionality and improve existing features.

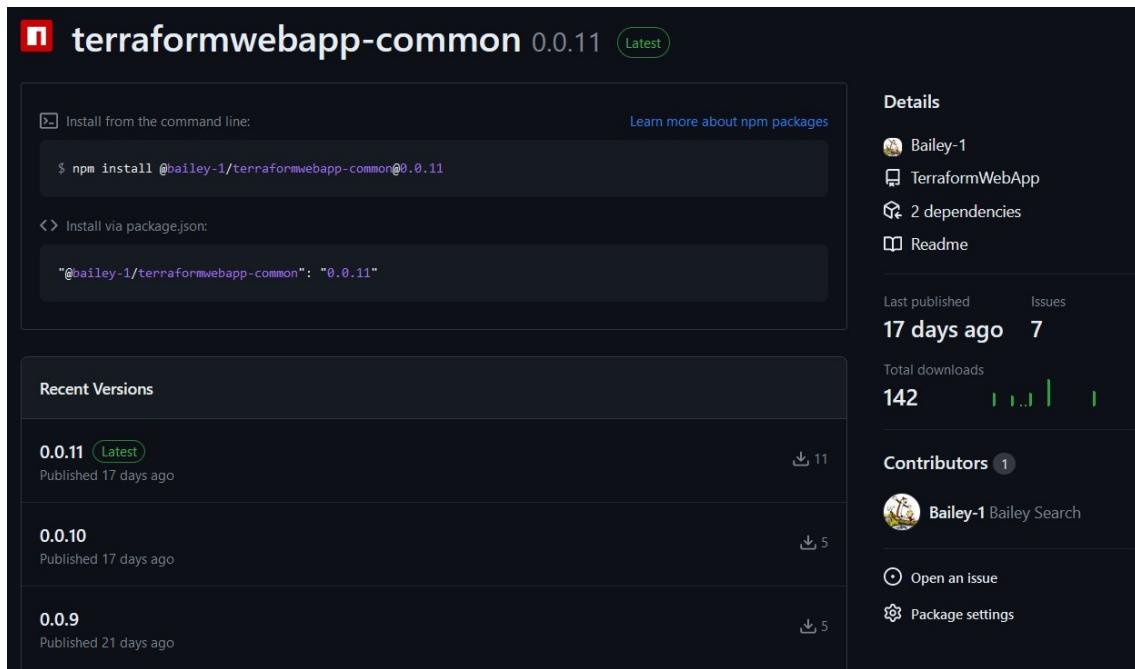


Figure 7.6: Shared private NPM library

To ensure that both the client and the backend remained compatible and any important changes are reflected in both, I developed a custom NPM package to store shared interfaces and types. This worked really well, as it prevented me from accidentally forgetting to update type usage in other locations. Figure 7.6 shows this shared NPM package.

7.2.5 v1.1 - Testing & HCL Displayer

In this version, I added tests to both the back and front end. I also added a popup modal to display the generated Terraform code to the user with an option to download or copy the file.

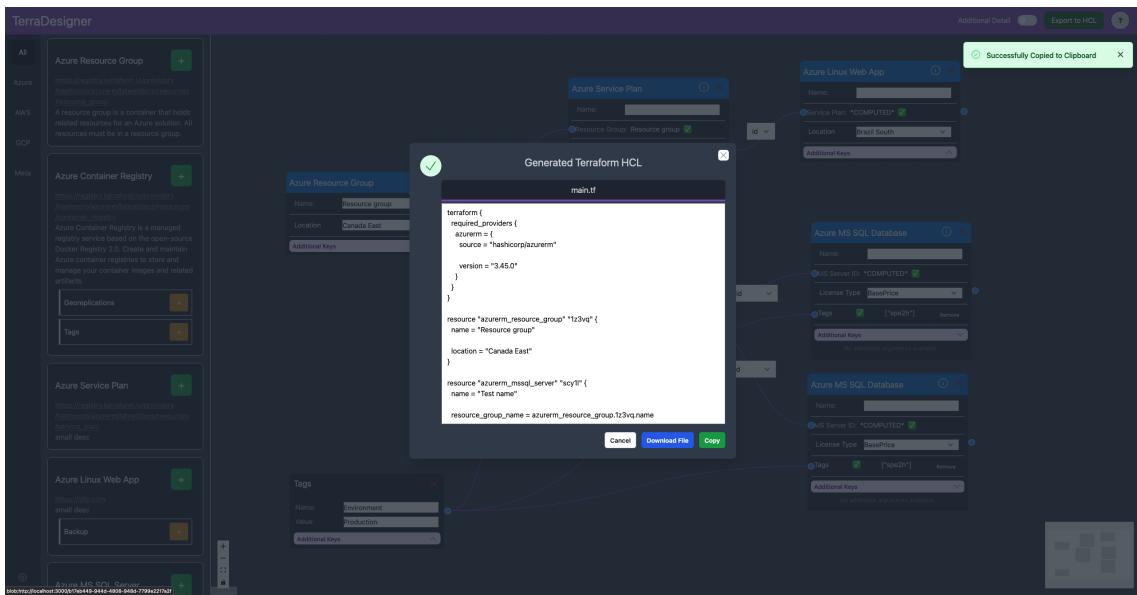


Figure 7.7: Generated Terraform File is now shown to the user

Image 7.7 shows a generated code example based on the user's configuration in the background. It also demonstrates the new notification system, which will give updates to the user of the system's state and notify them of anything important.

7.3 Tests

For both frontend and backend testing, I will use behaviour-driven development over test-driven development. This is primarily due to time management, as I am now sure what I will be able to develop within the time frame, so I don't want to waste time testing functionality I never implement.

7.3.1 Client end-to-end Testing

The end-to-end client testing uses the Cypress testing framework and software. This provides a very easy-to-use way to develop tests to test the whole application from the end user's perspective. This testing will test both the front end and the back end.

As well as standard testing, such as clicking a button and testing the correct elements should be displayed, I used automatic accessibility testing based on Axe. While it doesn't cover all WCAG 2.1 AA requirements, it does successfully check many of them, such as ensuring elements have readable text, element hierarchy is respected, and contrast ratios for text and backgrounds are valid to ensure excellent readability. An example can be seen in Figure 7.8.

3 accessibility violations were detected			
(index)	id	impact	description
0	'button-name'	'critical'	'Ensures buttons have discernible text.'
1	'color-contrast'	'serious'	'Ensures the contrast between foreground and background colors meets WCAG 2 AA contrast ratio thresholds'
2	'page-has-heading-one'	'moderate'	'Ensure that the page, or at least one of its frames contains a level-one heading'

1) Initial load			
6 accessibility violations were detected			
(index)	id	impact	description
0	'button-name'	'critical'	'Ensures buttons have discernible text'
1	'color-contrast'	'serious'	'Ensures the contrast between foreground and background colors meets WCAG 2 AA contrast ratio thresholds'
2	'heading-order'	'moderate'	'Ensures the order of headings is semantically correct'
3	'label'	'critical'	'Ensures every form element has a label'
4	'nested-interactive'	'serious'	'Ensures interactive controls are not nested as they are not always announced by screen readers or can cause focus problems for assistive technologies'
5	'select-name'	'critical'	'Ensures select element has an accessible name'

Figure 7.8: Cypress Axe Testing Results

7.3.2 Client Component Testing

The client component testing also uses the Cypress testing framework. This is different to the above end-to-end testing as it only tests a small component in isolation instead of the whole application. This means that tests are easier and quicker to write and can easily test difficult-to-reach component states to ensure full component coverage.

7.3.3 Backend API Testing

I set up unit tests for the backend to test the API using NPM packages such as Mocha, Chai and Supertest. This allows me to send test requests to the express REST API server and create assertions based on the data I get back in the response. This is useful for ensuring errors are handled correctly, and the correct status codes are used.

7.4 Continuous Integration

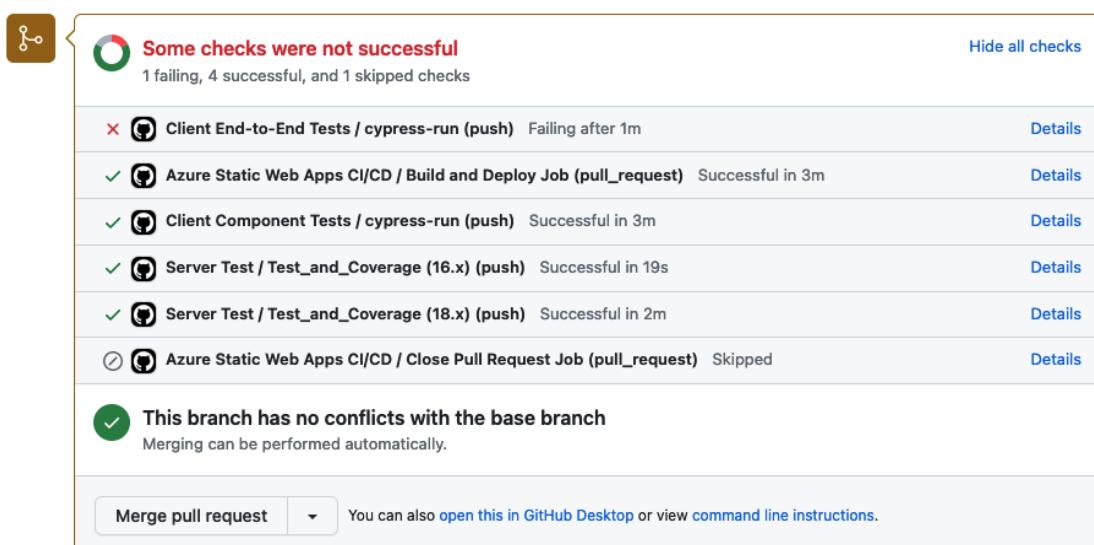


Figure 7.9: Example CI GitHub Actions

I also configured continuous integration for these tests so that any changes made to the client or server in a commit will run the tests automatically on GitHub Actions when the commit is pushed to GitHub. This helps avoid unknowingly breaking functionality when developing this application further. Green ticks indicate a successful action run, and the red cross shows a failure in figure 7.9.

This is particularly helpful for avoiding software regression issues where a feature is accidentally broken after previously working.

7.5 Continuous Deployment

Every push and merge to the Master branch updates the currently deployed software on Azure. This always makes it easier to have the most up-to-date software version running.

Changes to the client directory force an update of the client to the Azure Static Web App production slot.

The client is deployed to one of three additional staging slots for pull requests. This allows me to test it on the cloud on other devices more easily and helps identify any issues preventing it from being built or deployed that the tests do not cover. For example, the Azure configuration for the Static Web App.

7.6 Conclusion

Overall, I am extremely happy with how the development of TerraFormer progressed and what functionality I was able to build into it. The management of the project also went well as I always had a clear feature and next step that I could work on. This helped keep development quick and on track against the systems requirements and objectives.

There were many more features I was planning to develop but was unable to due to the limited development time.

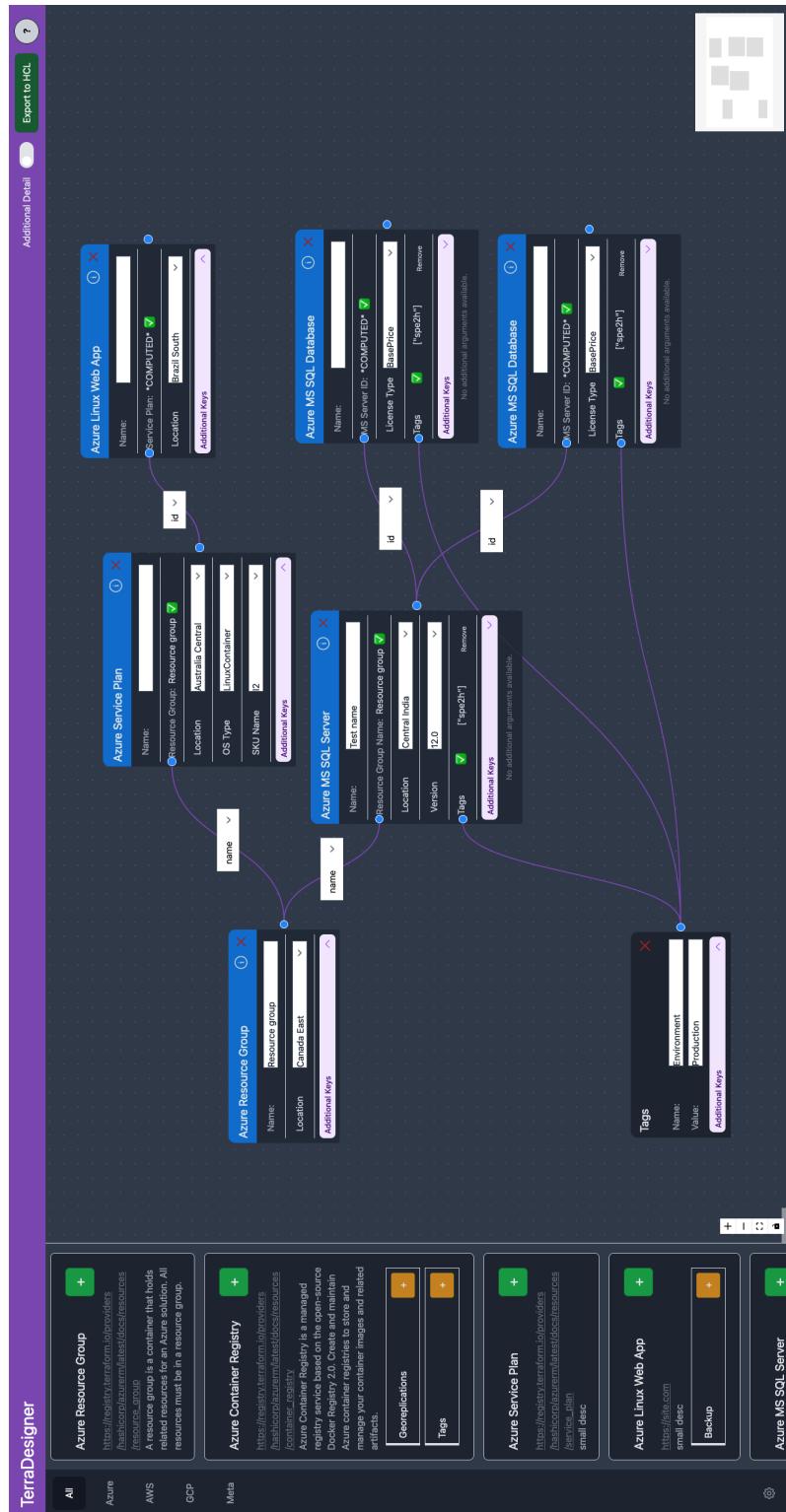


Figure 7.10: Example Cloud Environment in Terraformer

Chapter 8

Evaluation

8.1 Introduction

This chapter is for the evaluation of my software and the project itself.

The evaluation of the software artefact is against its aims, objectives and requirements.

8.2 Methodology

For this project, I used Kanban and Extreme Programming methodologies as these were the best suited for my project.

Both of these methodologies worked very well especially using the Kanban board to efficiently break down features into smaller tasks which I could then work on.

Unfortunately, I often found it difficult to work on a specific Kanban item as they are often closely coupled to other items, and I was the only one developing the software. This meant that most of my pull requests contained the changes required for 3-5 Kanban items instead of 1 or 2. This increased the size of what an iteration should be and reduced how quickly I could deploy new functionality. Larger changes increased the likelihood of accidentally adding a software regression bug to the codebase, which happened on occasion. It also increased the difficulty of adding unit tests to the project and ensuring all features have sufficient test coverage.

8.3 Project Management & Time Management

The project management worked very well, as I was able to develop most of the intended website over a few months.

Time management was good as well, as I always knew what I should be working on next and had a clear path of what functionality to implement and when.

Time management could have been improved several times, as I would spend more time

than necessary on a couple of areas. This was partially because I would get distracted and build more into a feature than required or because of an unexpected delay in the development of a feature. Better time management might have meant that I would have added or improved more features.

8.4 Evaluation against Project Objectives

This project was mainly successful against its objectives. This is because I was able to develop a modern graphical application to improve Terraform file development and improve the usability of cloud providers. This improves the experience for beginners, and the application successfully generates the supporting Terraform files.

The project wasn't successful against the other cloud service providers I had intended for it to support. This was because adding cloud resource support to the application was more difficult than anticipated, so I just focused on adding Microsoft Azure support. I could also not add as many resources as I wanted due to the time required.

8.5 Evaluation against Non-Functional Requirements

8.5.1 F1 - Present a filterable list of available resources for the user

Priority: Must-have

Met: Yes

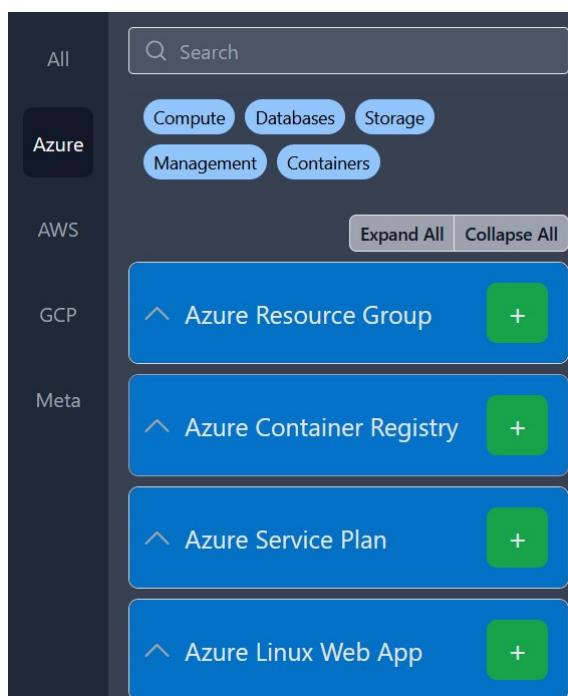


Figure 8.1: The left sidebar of the website with a filterable list of resources

The user is able to filter the list of resources in a range of useful ways.

1. Firstly, the user can filter the cloud provider of the resources. For example, only showing Azure or AWS items.
2. The user can also use the search bar to filter resources container a key word quickly
3. Lastly, the user is able to use the category filters to show all resources within a category quickly. For example, databases or compute-type resources.

8.5.2 F2 - Allow the user to create an instance of a resource which can then be edited

Priority: Must-have

Met: Yes



Figure 8.2: 3 Unique Instances of the Resource Group type Resource

The user is able to create unique instances of a resource and then edit the properties of the resource.

8.5.3 F3 - Provide information about available configurations for a specific resource

Priority: Must-have

Met: Yes

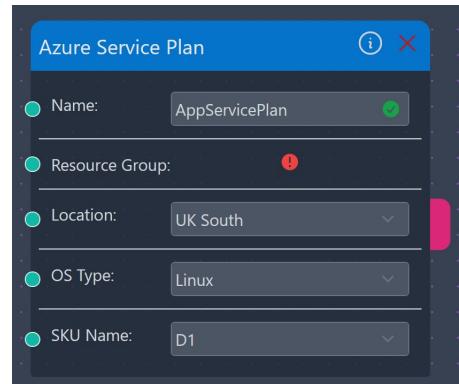


Figure 8.3: Provide information about available configurations for a specific resource

When suitable, resources list available options to make it easier for the user to identify the available configurations.

8.5.4 F4 - Generate the code used to provision the selected resource with the selected configurations by the user

Priority: Must-have

Met: Partially

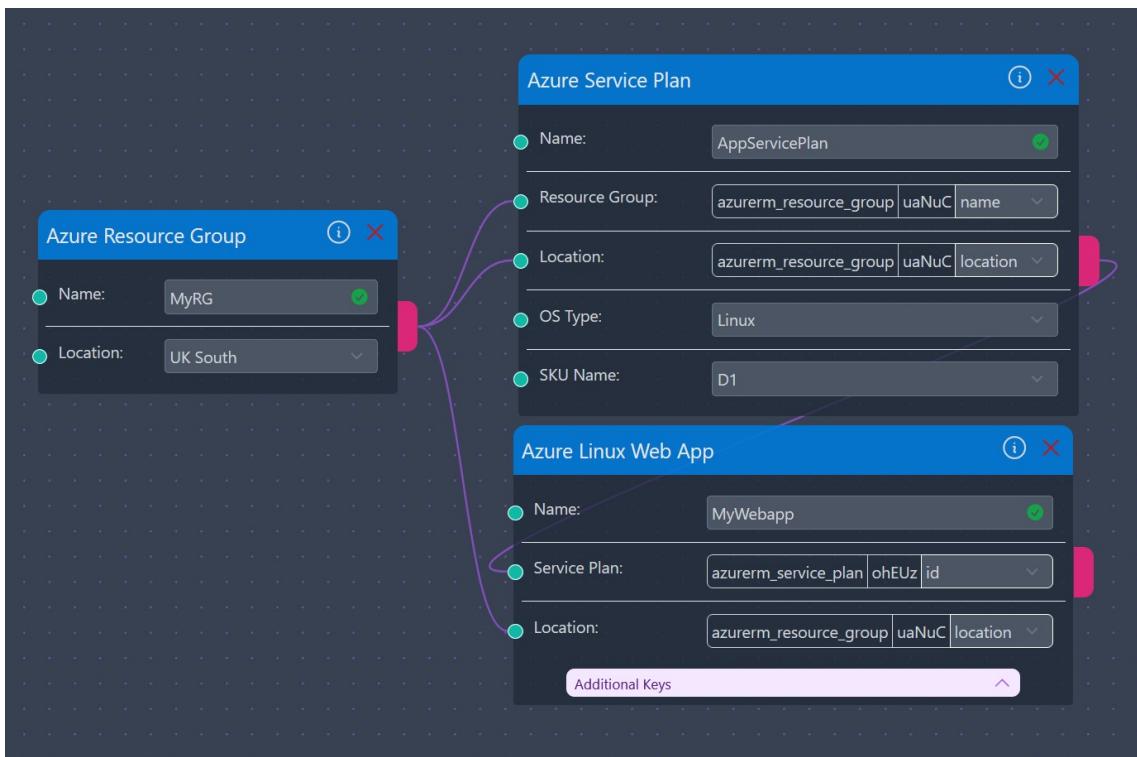


Figure 8.4: User resource configuration

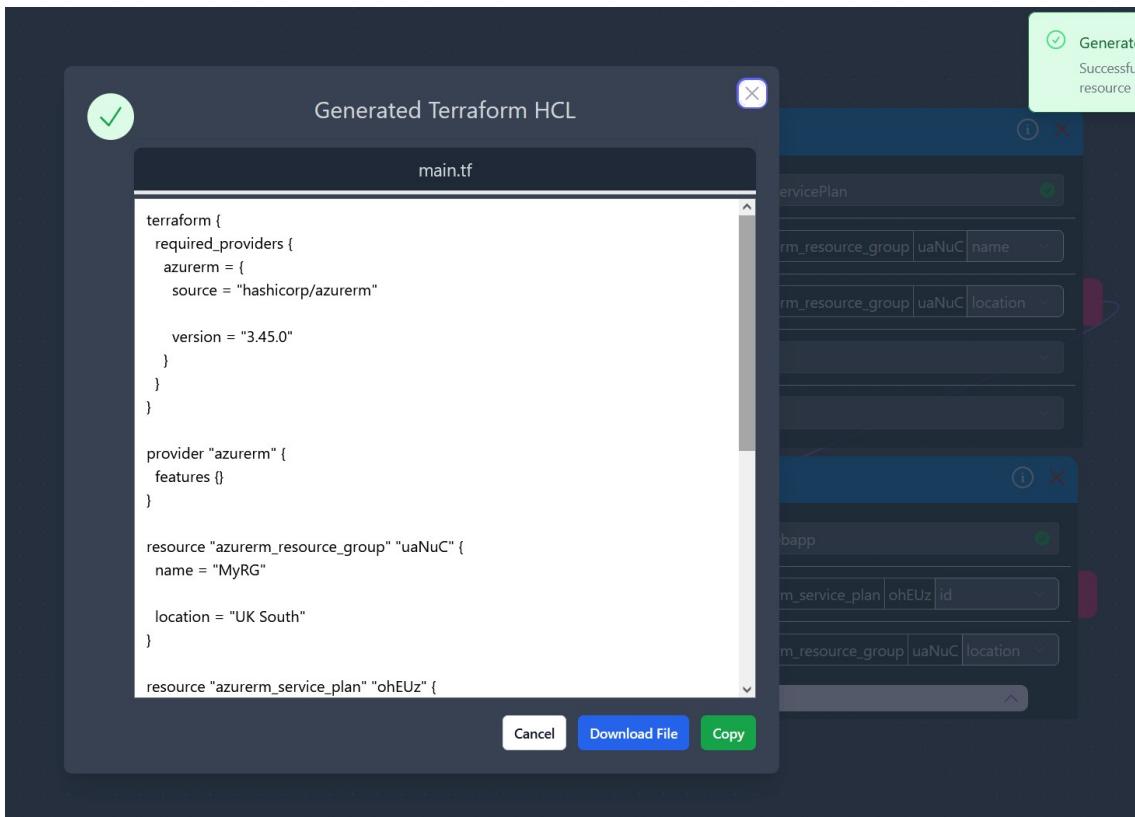


Figure 8.5: Generated HCL from the user resource configuration

The software successfully generates valid Terraform code for some Azure resources.

8.5.5 F5 - Allow the user to reference other attributes from a resource

Priority: Must-have

Met: Yes

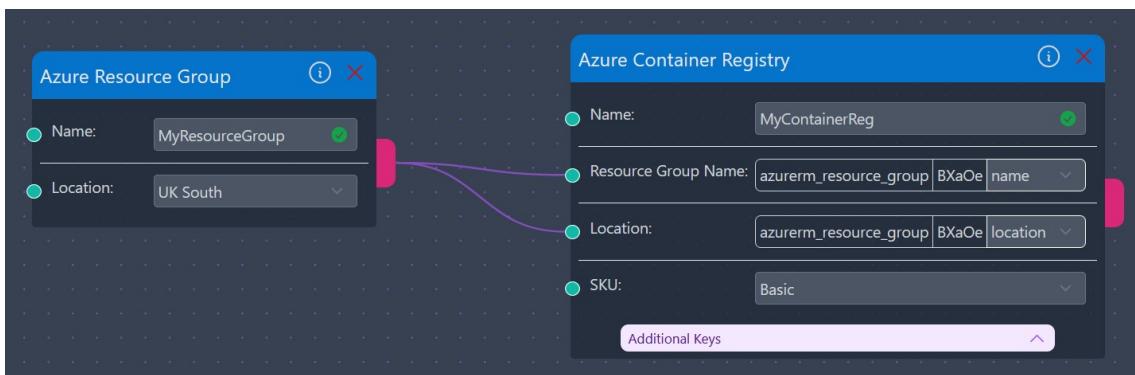


Figure 8.6: A Container Registry using the referenced attributes "name" and "location" from a resource group.

The software supports using references to other resource attributes as values.

8.5.6 F6 - Allow the user to add optional configuration options

Priority: Should-have

Met: Yes

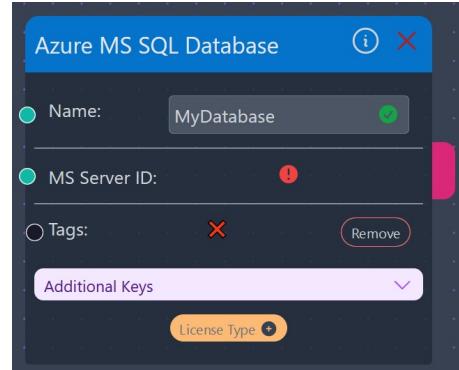


Figure 8.7: A MS Database resource with optional tag selected and listing another optional key the user could add

The software allows the user to add optional keys to a resource depending on the exact configuration the user requires.

8.5.7 F7 - Allow the user to reference object type variables

Priority: Should-have

Met: Yes

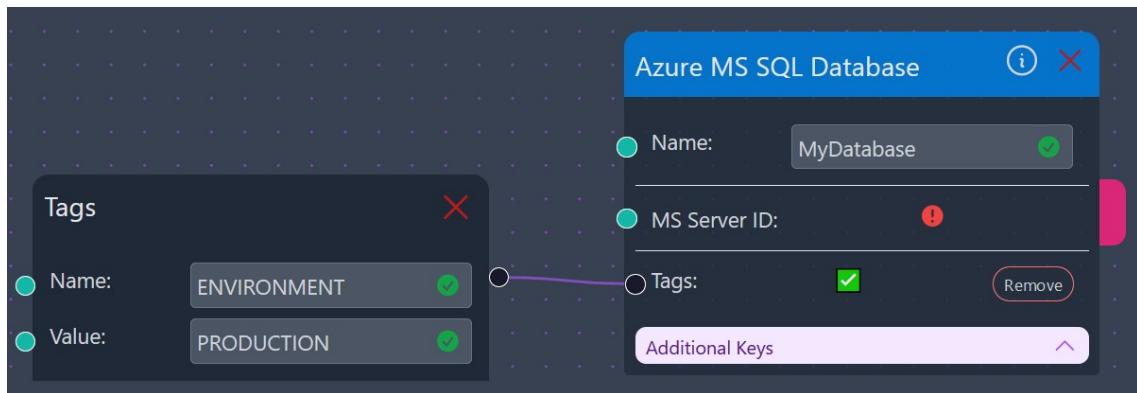


Figure 8.8: A MS Database using a Tag type object variable

The software supports using variable objects for more advanced configurations.

8.5.8 F8 - Inform the user of any mistakes or invalid configurations

Priority: Should-have

Met: Partially

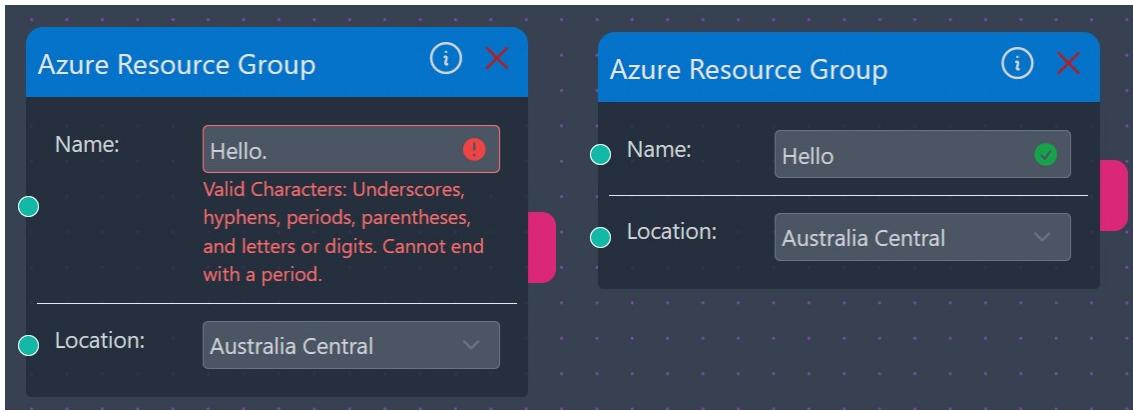


Figure 8.9: Two resources with valid and non-valid values.

The software only has support for single input validation, which should prevent most errors, but the user is still able to generate invalid Terraform configurations in some situations.

8.5.9 F9 - Provide pricing information and estimate to help inform the user on expected costs

Priority: Could-have

Met: No

This was outside of the scope of my prototype due to the large amount of work required to add it. This feature is probably large enough to become its own complete project.

8.5.10 F10 - Provide links to other relevant documentation, such as from Terraform and the cloud provider's official documentation, as a shortcut in case the user needs additional information

Priority: Could-have

Met: No

Development originally included links to other sites, but these were removed to improve the design and reduce screen clutter. These were originally in the resource list on the left side of the screen.

8.5.11 F11 - Allow the user to save the current configuration so it can be edited at a later time

Priority: Could-have

Met: No

This feature wasn't added as it was dependent on the database functionality, and trying to

add this to my project would have taken up too much time for what little improvement it would make.

8.5.12 F12 - Provide a quick start option to build common resources quickly

Priority: Could-have

Met: No

This wasn't included in the final prototype due to time limitations.

8.6 Evaluation against Functional Requirements

8.6.1 NF1 - Provide a modern visual web application for users to interact with

Priority: Must-have

Met: Yes

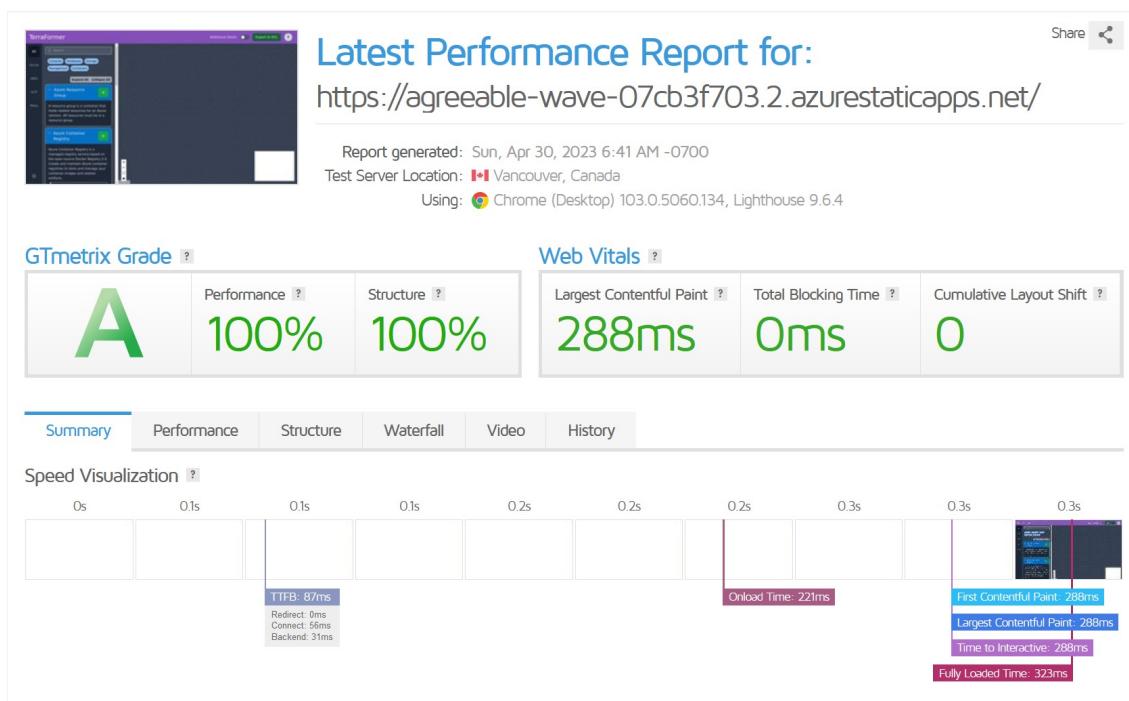


Figure 8.10: gtmetrix.com result page for TerraFormer

Figure 8.10 shows the test result page for my application using a popular website performance reporting tool, GTmetrix. This tool records the performance of the initial website load, such as the file sizes and time to load. The time for a website to load has a huge impact on user engagement, and long times are generally considered to be unacceptable. For example, the BBC found that for every additional second a webpage takes to load, 10 per cent of users leave (Clark, 2018).

The result from the testing shows that the page loads completely in 323ms, and the time to interact is 288ms.

The "Time to Interactive" measurement is the time required for the site to become reliably responsive by the user (Anonymous, n.d.).

The target for this requirement was to have a full load time of under 2 seconds, as this is what users typically expect (Baker, 2022).

8.6.2 NF2 - Generates valid Terraform HCL code

Priority: Must-have

Met: Yes

```
# azurerm_mssql_server.tZViR will be created
+ resource "azurerm_mssql_server" "tZViR" {
    + administrator_login                  = "username"
    + administrator_login_password          = (sensitive value)
    + connection_policy                   = "Default"
    + fully_qualified_domain_name         = (known after apply)
    + id                                = (known after apply)
    + location                           = "uksouth"
    + minimum_tls_version                = "1.2"
    + name                               = "azuresqlserver"
    + outbound_network_restriction_enabled = false
    + primary_user_assigned_identity_id   = (known after apply)
    + public_network_access_enabled       = true
    + resource_group_name                 = "MyRG"
    + restorable_dropped_database_ids     = (known after apply)
    + version                            = "12.0"
}

# azurerm_resource_group.Yqcwd will be created
+ resource "azurerm_resource_group" "Yqcwd" {
    + id      = (known after apply)
    + location = "uksouth"
    + name    = "MyRG"
}

Plan: 4 to add, 0 to change, 0 to destroy.
```

Figure 8.11: Terraform CLI validating the generated file

As figure 8.11 shows, the generated Terraform file is valid according to the Terraform CLI tool. This validation step is required by the Terraform system before changes can be executed on the cloud service provider. This configuration example included a resource group, container registry, MS SQL server and an MS SQL database.

8.6.3 NF3 - A reliable, well-built app with a high level of quality. This means that the app should act predictably and work well regardless of how the user uses it

Priority: Must-have

Met: Partially

While the React web application is as well built as I was able to, there are still some issues with it and some occasional bugs. I have tried to solve as many of these as possible but was limited by the development time. I used typescript to add static typing to JavaScript to prevent common issues with incorrect types and non-standard objects. This helped significantly to ensure objects implemented the correct interfaces and had consistent properties regardless of where the object was used. I also developed a shared private NPM package to store interfaces and types so that they can be used consistently in the front and backend, and both sides use any updates.

The application has also been tested in other web browsers, such as Chrome, Firefox and Safari, to ensure it works. This means that the web application works predictably regardless of what browser the user wants to use and ensures I haven't accidentally used any browser-specific functionality.

8.6.4 NF4 - Provide a greater level of abstraction for the user to make it easier to work using the cloud without having to spend a lot of time learning about it

Priority: Must-have

Met: Yes

This requirement was successfully implemented into the application as it allows users to develop a Terraform file without having to understand its underlying structure or manually work out the required properties for each cloud resource.

The user is only required to know what resources they want to use and the required configuration of each resource.

8.6.5 NF5 - The web app must scale appropriately to cover standard desktop sizes

Priority: Must-have

Met: Yes

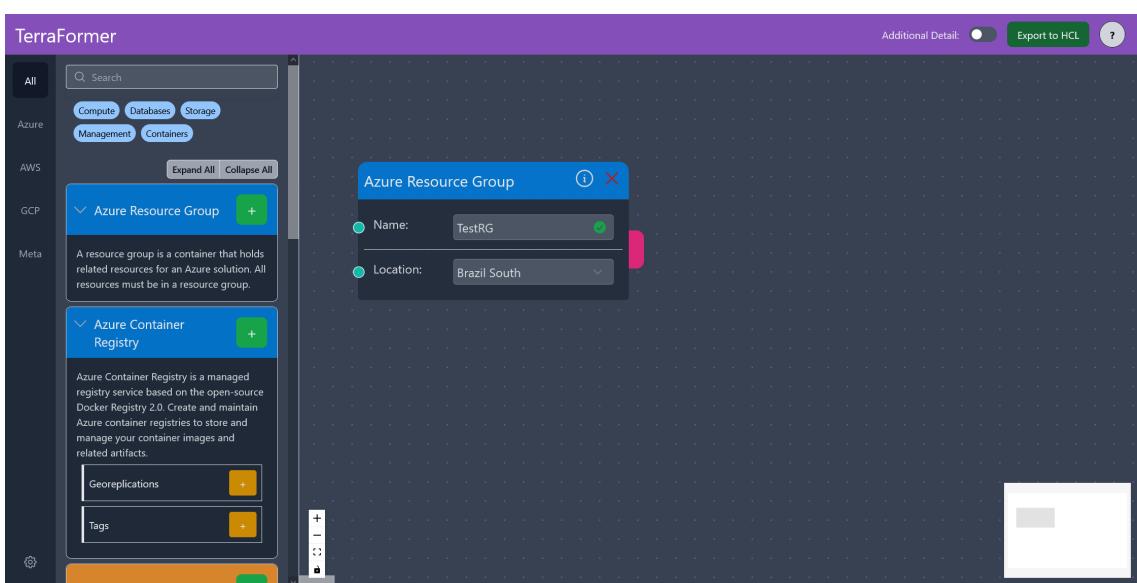


Figure 8.12: Shorter and wider viewport

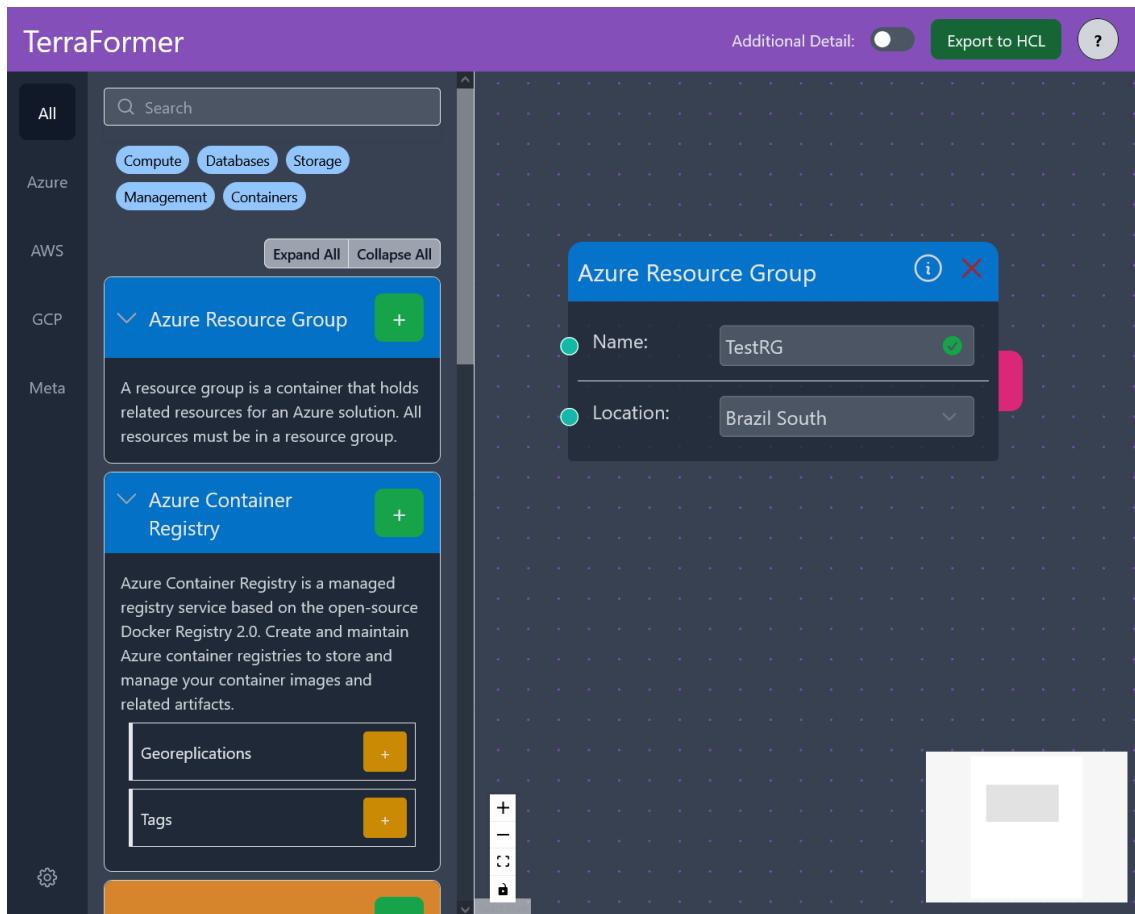


Figure 8.13: Taller and thinner viewport

Figures 8.12 & 8.13 both show that the web page smoothly resizes to fit the browser viewport dimensions. This is successful because it will allow the user to completely use the application regardless of what desktop browser or monitor they are using.

The website does not resize well for mobile devices, as it is highly unlikely that this sort of application would need to be accessed from such a device.

8.6.6 NF6 - The web app should comply with WCAG 2.1 AA accessibility standard

Priority: Should-have

Met: No

Testing the accessibility of the web application is a clear requirement. To do this, I am aiming for WCAG 2.1 AA compliance.

WCAG has three levels of support, from A to AAA. For this project, the target is AA level support, which is what most websites should aim for and would offer good accessibility support (“Web Content Accessibility Guidelines (WCAG) 2 Level AAA Conformance | Web Accessibility Initiative (WAI) | W3C”, n.d.).

- A: Essential
- AA: Ideal Support
- AAA: Specialised Support

WCAG 2.1 builds on from 2.0 with a greater focus on mobile devices and accessibility for people with low vision and cognitive and learning disabilities (“What’s New in WCAG 2.1 | Web Accessibility Initiative (WAI) | W3C”, n.d.).

While the application does have automatic accessibility testing support, which tests for many of the requirements, I was not able to fix all of the identified issues. This was mainly due to the limited time to complete this project and the standard’s very high threshold for what is acceptable. Although I failed to meet this requirement, I did fix most of the issues it found, and because of the testing, I am now aware of the other issues. Despite not reaching AA compliance, the accessibility of my web application has been improved and will now be easier for more people to use. If I had more time, I could address these issues, and due to the testing, I would know exactly what page elements to look at and what the issue with them is.

8.6.7 NF7 - Communicate securely with cloud provider APIs

Priority: Could-have

Met: No

This was another feature that the application was meant to include, at least partially but it had to be removed from the plan due to the limited development time and the large amount of work required to implement this feature.

8.6.8 NF8 - Support generating ‘.tf.json’ files and ‘.tf’ files

Priority: Could-have

Met: Partially

This feature is partially met as the application has support for ‘.tf’ type files but not ‘.tf.json’ type. From early on in the project, ‘.tf’ type files had been identified as the main file type that the application needed to support.

8.7 Future Work

Although I managed to achieve my main objectives with this project, much can still be improved or added.

Firstly, the resource nodes on the React client can be improved to add better error handling and validation for dynamic resource attributes. More form input types could be added and support for more data types. As well as just general improvements to the design of the client and the code architecture itself.

Adding support for additional cloud providers such as GCP or AWS would give the user a better range of options to choose from.

Providing built-in documentation to guide the user into setting up and using Terraform CLI would greatly improve the usability of the application. The documentation would also include using TerraFormer and configuring resources within the web application.

Server-side storage of Terraform configurations. This would allow users to sign in and view their previous configurations to make it quicker and easier for them to continue working with the website. I decided against trying to implement this functionality in the original design as this would require authentication, which would take too much time to configure correctly. Support for converting already-developed Terraform files into their equivalent nodes would also improve the site and make working with Terraform files easier.

Server-side storage of resources and admin panel. This change would store all of the resource data in a database instead of directly in the client, as it is now. This would reduce the amount of unneeded data sent to the client and would allow resource management to be carried out by using an admin page. Currently, this has to be done by manually editing the resource files. I would choose a NoSQL database over a relational database because the data will be in JSON format and follow a semi-regular structure.

8.8 Reflection & Conclusion

The initial project objective was to create an application to improve the usability of provisioning and managing cloud resources through Terraform. This also meant that it should be quicker and easier for beginners to create specific cloud resources.

This project has given me the opportunity to demonstrate much of what I have learnt while on this Software Engineering course. Most notably web development, usability design and project management. However, I'm sure I used something from almost every module somewhere in this project.

I also think the technology choice was right for this project, which included Typescript, React, Redux, Tailwind CSS, Node and Express. This is because it is well suited for developing modern web applications. I was also very confident using these languages and frameworks when I started this project and have used similar technologies when working on my projects, which meant that I could quickly start building the prototype and have a working version halfway through development. This gave me a lot of time to improve it and continue adding new features to the application.

Throughout this entire project, I have had to deal with a huge amount of issues. These

issues ranged from the management of the project to issues with the code itself. Management issues were usually solved by discussing the issue with my project supervisor and listening to her advice, and code issues were solved by using online resources such as documentation, blogs and forums like StackOverflow.

If I had to develop this project again, I would reduce the scope of what I was trying to support with the Terraform HCL language. This is because the language has support for many features, such as dynamically referencing attributes which was the source of many issues towards the end of the application development. Looking back on this issue, I should have duplicated the value in the generated code when required. I would also significantly reduce the scope of cloud providers I was trying to support and would just develop it for Microsoft Azure as I severely underestimated how long it takes to add support for cloud resources to my application and foolishly assumed I would be able to add support for dozens on each of the three main providers.

Overall, I am very happy with how this project and application progressed. I have also learnt more about project management, usability design and the technology used in the development, which I'm sure will be useful in the future.

References

- Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2017). *Agile software development methods: Review and analysis*. <http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf>.
- Ahmad, M. O., Markkula, J., & Oivo, M. (2013). Kanban in software development: A systematic literature review. *Proceedings - 39th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2013*, 9–16. <https://doi.org/10.1109/SEAA.2013.28>
- Ahmed, S., & Eduard, M. (2004). The obstacles and myths of usability and software engineering. *Communications of the ACM*, 47, 71–76. <https://doi.org/1035134.1035136>
- Anderson, J., Fleek, F., Garrity, K., & Drake, F. (2001). Integrating usability techniques into software development. *IEEE Software*, 18, 46–53. <https://doi.org/10.1109/52.903166>
- Anonymous. (n.d.). *Time to interactive* | gtmetrix. <https://gtmetrix.com/time-to-interactive.html>
- Avouris, N. M. (2001). An introduction to software usability. *Proc. 8th Panhellenic Conference on Informatics*, 2. www.ee.upatras.gr/hci/usabilitynet
- Baker, K. (2022). *11 website page load time statistics you need [+ how to increase conversion rate]*. <https://blog.hubspot.com/marketing/page-load-time-conversion-rates>
- Beck, K. (1999). Embracing change with extreme programming. *Computer*, 32, 70–77. <https://doi.org/10.1109/2.796139>
- Brown, S. (2022). *The c4 model for visualising software architecture*. <https://c4model.com/>
- Buyya, R., Srivasta, S. N., Casale, G., Calheiros, R., Simmhan, Y., Varghese, B., Gelenbe, E., Javadi, B., Vaquero, L. M., Netto, M. A., Toosi, A. N., Rodriguez, M. A., Llorente, I. M., Vimercati, S. D. C. D., Samarati, P., Milojicic, D., Varela, C., Bahsoon, R., Assuncao, M. D. D., ... Shen, H. (2019). A manifesto for future generation cloud computing: Research directions for the next decade. *ACM Computing Surveys*, 51. <https://doi.org/10.1145/3241737>

- Carvajal, L., Moreno, A. M., Sánchez-Segura, M. I., & Seffah, A. (2013). Usability through software design. *IEEE Transactions on Software Engineering*, 39, 1582–1596. <https://doi.org/10.1109/TSE.2013.29>
- Clark, M. (2018). *How the bbc builds websites that scale | creative bloq*. <https://www.creativebloq.com/features/how-the-bbc-builds-websites-that-scale>
- Coleman, C., Griswold, W. G., & Mitchell, N. (2022). Do cloud developers prefer clis or web consoles? clis mostly, though it varies by task. <http://arxiv.org/abs/2209.07365>
- Cusumano, M. A., & Smith, S. (1995). *Beyond the waterfall: Software development at microsoft*.
- Data protection act 2018*. (2018). <https://www.legislation.gov.uk/ukpga/2018/12/contents/enacted>
- Edeki, C. (2015). *Agile software development methodology* (1). Progressive Academic Publishing. www.idpublications.org
- Ferré, X., Juristo, N., Windl, H., & Constantine, L. (2001). Usability basics for software developers. *IEEE Software*, 18, 22–29. <https://doi.org/10.1109/52.903160>
- Fowler, M., & Highsmith, J. (2001). *The agile manifesto*. www.martinfowler.com/articles/newMethodology.html
- Functional vs non-functional requirements - understand the difference*. (2012). <https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/>
- Gene, K., Jez, H., Patrick, D., & John, W. (2021). *The devops handbook: How to create world-class agility, reliability, & security in technology organizations* (Second Edition). IT Revolution.
- Hatton, S. (2008). Choosing the "right" prioritisation method. <https://doi.org/10.1109/ASWEC.2008.22>
- Holzinger, A. (2005). Usability engineering methods for software developers. *Communications of the ACM*, 48, 71–74. <https://doi.org/10.1145/1039539.1039541>
- ISO/IEC 25010. (2011). *ISO/IEC 25010:2011, systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models*.
- Kavya, S., Nath, G. P., & Cherukuri, N. (2021). *Comparison of gcp and aws using usability heuristic and cognitive walkthrough while creating and launching virtual machine instances in virtual private cloud*. www.bth.se
- Kirkpatrick, D. (2016). *Google: 53% of mobile users abandon sites that take over 3 seconds to load | marketing dive*. <https://www.marketingdive.com/news/google-53-of-mobile-users-abandon-sites-that-take-over-3-seconds-to-load/426070/>
- Kirovska, N., & Kočeski, S. (2015). *Usage of kanban methodology at software development teams*.

- Lin, H. X., Choong, Y. Y., & Salvendy, G. (1997). A proposed index of usability: A method for comparing the relative usability of different software systems. *Behaviour and Information Technology*, 16, 267–277. <https://doi.org/10.1080/014492997119833>
- Machiraju, S., & Gaurav, S. (2018). *Devops for azure applications*. Apress. <https://doi.org/10.1007/978-1-4842-3643-7>
- Mariage, C., Vanderdonckt, J., & Pribeanu, C. (2006). *State of the art of web usability guidelines*.
- Moreno, A. M., Seffah, A., Capilla, R., & Sánchez-Segura, M.-I. (2013). *Hci practices for building usable software*. <http://usabilitybok.org/>
- Murillo, S. R., & Sánchez, J. A. (2014). Empowering interfaces for system administrators: Keeping the command line in mind when designing guis. *ACM International Conference Proceeding Series*, 10-12-September-2014. <https://doi.org/10.1145/2662253.2662300>
- Nichols, D. M., & Twidale, M. B. (2002). *Usability and open source software*.
- Nielsen, J. (2003). *Return on investment for usability*. <https://www.nngroup.com/articles/return-on-investment-for-usability/>
- Nielsen, J. (2005). *Heuristic evaluation ten usability heuristics*.
- Nielsen, J. (2012). *Usability 101: Introduction to usability*. <https://www.nngroup.com/articles/usability-101-introduction-to-usability/>
- Olofsson, R., & Hultstrand, S. (2015). *Git-cli or gui which is most widely used and why?* Blekinge Institute of Technology. www.bth.se/didd
- Parameshwar, A., & Yudelman, H. (2022). *Api, cli, gui, oh my! understanding user behavior across surfaces*.
- Richards, M., & Ford, N. (2020). *Fundamentals of software architecture: An engineering approach fundamentals of software architecture*.
- Riti, P., & Flynn, D. (2021). *Beginning hcl programming*. Apress. <https://doi.org/10.1007/978-1-4842-6634-2>
- Sabharwal, N., Pandey, S., & Pandey, P. (2021). *Infrastructure-as-code automation using terraform, packer, vault, nomad and consul*. Apress. <https://doi.org/10.1007/978-1-4842-7129-2>
- Saeed, S., Jhanjhi, N. Z., Naqvi, M., & Humayun, M. (2019). Analysis of software development methodologies. *International Journal of Computing and Digital Systems*, 8, 445–460. <https://doi.org/10.12785/ijcds/080502>
- Sampath, H., Merrick, A., & MacVean, A. (2021). Accessibility of command line interfaces. *Conference on Human Factors in Computing Systems - Proceedings*. <https://doi.org/10.1145/3411764.3445544>
- Saxena, S., & Dubey, S. K. (2013). Impact of software design aspects on usability. *International Journal of Computer Applications*, 61.

- Silva, G. C., Ré, R., Aurélio, M., & Silva, G. (2018). *Evaluating efficiency, effectiveness and satisfaction of aws and azure from the perspective of cloud beginners*. ACM. <https://azure.microsoft.com/en-us/pricing/>
- Stanton, B., Theofanos, M., & Joshi, K. P. (2015). Framework for cloud usability. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9190, 664–671. https://doi.org/10.1007/978-3-319-20376-8_59
- van Merode, H. (2023). *Continuous integration (ci) and continuous delivery (cd)*. Apress. <https://doi.org/10.1007/978-1-4842-9228-0>
- Web content accessibility guidelines (wcag) 2 level aaa conformance | web accessibility initiative (wai) | w3c.* (n.d.). <https://www.w3.org/WAI/WCAG2AAA-Conformance>
- What's new in wcag 2.1 | web accessibility initiative (wai) | w3c.* (n.d.). <https://www.w3.org/WAI/standards-guidelines/wcag/new-in-21/>

Appendix A

Ethics Certificate



Certificate of Ethics Review

Project title: Improving the Usability of Terraform

Name:	Bailey Search	User ID:	UP93768 3	Application date:	14/10/2022 10:35:39	ER Number:	TETHIC-2022-103751
--------------	---------------	-----------------	--------------	--------------------------	------------------------	-------------------	--------------------

You must download your referral certificate, print a copy and keep it as a record of this review.

The FEC representative(s) for the **School of Computing** is/are [Haythem Nakkas, Dalin Zhou](#)

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- [University Policy](#)
- [Safety on Geological Fieldwork](#)

It is also your responsibility to follow University guidance on Data Protection Policy:

- [General guidance for all data protection issues](#)
- [University Data Protection Policy](#)

Which school/department do you belong to?: **School of Computing**

What is your primary role at the University?: **Undergraduate Student**

What is the name of the member of staff who is responsible for supervising your project?: **Kirsten Smith**

Is the study likely to involve human subjects (observation) or participants?: Yes

Will you gather data about people (e.g. socio-economic, clinical, psychological, biological?): No

Will you gather data from people about some artefact or research question (e.g. opinions, feedback?): No

Will the study involve National Health Service patients or staff?: No

Do human participants/subjects take part in studies without their knowledge/consent at the time, or will deception of any sort be involved? (e.g. covert observation of people, especially if in a non-public place): No

Will you collect or analyse personally identifiable information about anyone or monitor their communications or on-line activities without their explicit consent?: No

Does the study involve participants who are unable to give informed consent or are in a dependent position (e.g. children, people with learning disabilities, unconscious patients, Portsmouth University students?): No

Are drugs, placebos or other substances (e.g. food substances, vitamins) to be administered to the study participants?: No

Will blood or tissue samples be obtained from participants?: No

Is pain or more than mild discomfort likely to result from the study?: No

Could the study induce psychological stress or anxiety in participants or third parties?: No

Will the study involve prolonged or repetitive testing?: No

Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?: No

Are there risks of significant damage to physical and/or ecological environmental features?: No

Are there risks of significant damage to features of historical or cultural heritage (e.g. impacts of study techniques, taking of samples?): No

Does the project involve animals in any way?: No

Could the research outputs potentially be harmful to third parties?: No

Could your research/artefact be adapted and be misused?: No

Will your project or project deliverables be relevant to defence, the military, police or other security organisations and/or in addition, could it be used by others to threaten UK security?: No

Supervisor Review

As supervisor, I will ensure that this work will be conducted in an ethical manner in line with the University Ethics Policy.

Supervisor comments: **##supervisorComments##**

Supervisor's Digital Signature: **##supervisorSig##** Date: **##supDate##**

Appendix B

Project Initiation Document



UNIVERSITY OF
PORTSMOUTH

**School of Computing
Final Year Engineering Project**

Project Initiation Document

Bailey Search

**Developing a Web App to improve the
Usability of Terraform**

1. Basic details

Student name:	Bailey Search
Draft project title:	Developing a Web App to Improve the Usability of Terraform
Course and year:	BSC (Hons) Software Engineering
Project supervisor:	Kirsten Smith
Client organisation:	N/A
Client contact name:	N/A

2. Degree suitability

This project is very suitable for my Software Engineering course because it will provide a challenging software project for me to attempt. This will allow me to demonstrate some of the skills and things I've learnt over the last couple of years on this course and from university. To do this, I will develop a full stack javascript web application which will consist of a react frontend and a Node REST API.

Another reason I settled for this project is that it will result in a unique tool that will improve the usability of cloud computing and Terraform, which should make it easier to work with and develop.

3. Outline of the project environment and problem to be solved

What is the problem that you will investigate?

Terraform is a popular command line tool which allows users to provision and manage complicated cloud infrastructure through code instead of using a GUI or each cloud provider's CLI. To do so, you must write a "main.tf" file which defines each required resource in a code block. The issue with this is that it is difficult to know what key-value pairs are required in each resource block. This isn't the same as learning a programming language, as each resource requires specific keys, which you probably won't be able to guess without having worked with the type before.

Currently, most people achieve this by searching online for the resource's official Terraform documentation and using it to work out what changes are required to their files. But it is not always clear what keys are required to provision a resource and what keys are optional. It can also be difficult to tell what values a specific key can accept and what the default value is if a key is not set by the user in the resource block. In some situations, this can lead to unexpected costs as cloud resources can default to provisioning a greater amount of hardware resources than the user would configure, such as more memory or CPU cores, or when resources default to using

safer contingency options such as geo-replications and backups, which might not necessarily be required for the users need but does increase the cost.

This is also exacerbated when it comes to more complex resources, which require multiple resource blocks to be correctly configured, such as with storage accounts or Kubernetes and for blocks to correctly reference other blocks.

Why is it worth working on?

I think this is worth working on as it will allow more straightforward and quicker development of basic environments in Terraform and will also provide a better overview of required settings, default settings, possible values for each setting and pricing.

I also think this will be valuable to people who frequently use Terraform as it will allow them to save time when quickly developing the base file, which they can then configure further.

This will also provide an easier way for people who want to learn more about cloud computing and how to deploy their projects to the cloud. It could also improve the ease of using other cloud providers since you wouldn't have to learn a new CLI or tool.

4. Project aim and objectives

The primary aim of this project is to speed up the development of a basic cloud environment with Terraform by programmatically generating Terraform files for the user from an interactive user interface and, by doing so, provide a more accessible way to get started with cloud computing for new users.

The main objectives of this project are

- Develop a web application using the prototypes and requirements to improve the usability of Terraform.
 - Provide support for resources from a range of cloud providers. Primarily the main three; AWS, Azure and GCP.
 - Allow the user to change the standard settings of resources. (Each resource supports many options, so I doubt I can add complete support within this limited timeframe.)
 - Allow the user to export this configuration as a valid main file (main.tf), variable file (variables.tf), and output file (outputs.tf).
 - Provide support for stand-alone resource blocks and support for resources which require multiple resource blocks to be configured.
- Deploy the end project to the cloud so I can easily demonstrate it.
- Carry out research into the usability of my software.

- Evaluate how successful my software is.

5. Project constraints

- Terraform, its documentation and other software, such as node packages, are primarily open source and are developed for free by thousands of developers. This means that I am heavily dependent on this continuing and no massive breaking changes, which would break the work I have already done.
- It is also possible that Terraform could change significantly in the time it takes me to develop this project. So, in this case, I would have to alter my current plans for the platform, or I would have to limit it to a specific compatible version.
- The biggest constraint I have for this project is the limited timeframe to complete this.

6. Facilities and resources

- No university resources, such as a lab or equipment, are required.
- I already have access to all the software I need on my computers. Software I will be using for this project includes, WebStorm, DataGrip, Postman and GitKraken.
- I will require various cloud computing resources to test my project since my project is about provisioning cloud resources easily. These will be more expensive than my deployment resources as many of these will be provisioned compute resources (i.e. cost a fixed amount per hour regardless of use) but should still be affordable for short durations.
- I also plan on deploying my finished project to the cloud. These resources will either be within the free tier or reasonably cheap. Due to the type of workload, I will utilise serverless options where available to reduce cost.
 - Node express API - Azure App Service (Or Azure Functions)
 - MongoDB database - Serverless Azure CosmosDB w/ Mongo adapter
 - React client - Azure static web app

7. Log of risks

Description	Impact	Mitigation/Avoidance
I could run up huge costs on a cloud provider and would be unable to pay it off.	It would be expensive to pay off.	<ul style="list-style-type: none"> • Manage resources carefully - Provisioning things only when I need them. • Be more restrictive regarding the set configuration options on provisioned resources to lower costs such as CPU/Memory etc.
Unable to finish the prototype due to limited development time.	I would not be able to complete the report.	<ul style="list-style-type: none"> • Carefully plan how to make a minimal prototype for it. Plan and test how to make each section of

		the website.
Lack of frontend experience could make development more time-consuming than originally planned.	The development would be slower, and I might not be able to implement some features.	<ul style="list-style-type: none"> Allocate my time than expected for this so that any unexpected delays won't significantly affect the project timeline. Focus on the MVP for the project to ensure I only spend time on what is important. Continue to work on projects in my own time to gain more experience, so hopefully, I can hit the ground running when I start development work on this.
Towards the end of the project, I could accidentally break some work I did earlier, consequently removing functionality from my web app.	It would take away functionality from my work and lower the software's quality as it doesn't work as intended.	<ul style="list-style-type: none"> Use GitHub for version control, so I can carefully manage and revert changes. Create regular backups of everything to another cloud provider, such as on Google Drive. Develop unit/integration tests for functionality to make it easier to test the whole application to ensure it all works. Develop CI/CD pipelines to automatically run unit/integration tests for changes on pull requests - so I can't merge changes into master without these tests passing.
Can't find enough students to participate in my primary research	It would prevent me from evaluating the usability of Terraform and the success of my project.	<ul style="list-style-type: none"> Contact students from other courses/years. Use societies to try to reach relevant people.
MacBook could break, and I would lose all work for this project.	Huge since I would have to start again for the report and artefact.	<ul style="list-style-type: none"> Use GitHub for version control and cloud storage for both the artefact source code and the LaTeX report document, so I can access it from other devices and revert changes if necessary. Create regular backups of everything to another cloud provider, such as on Google Drive. Don't rely on anything too specific to macOS, so if I have to change computers and operating systems halfway through, I should be able to.

My supervisor or I could become ill.	Very big since it would prevent me from doing my report or from getting regular feedback on my work.	<ul style="list-style-type: none"> Prioritise tasks effectively so if this were to happen, it shouldn't affect anything too important. Regularly talk in the catch-up meetings so that we would both get notice if anything could occur in the next few weeks.
--------------------------------------	--	--

8. Project deliverables

The software artefact produced from this project will consist of a React front-end web application and a node express REST API. The react front end will be what runs on the client browser and will be where the user can configure their resources and cloud providers. The backend API will fetch cloud resources from a database, will handle the generation of the HCL file and will also send API requests to cloud providers to gain extra data. I also plan on adding Swagger API Documentation which will provide detailed information on the API routes, status codes and what data should be sent/received.

I will produce a project report, literature review, and user guide for documents.

9. Project approach

For project management, I plan on using the Agile methodology and specific elements of extreme programming. This will focus on the development of features required immediately and will put less of a focus on maintainability and future-proofing the application. This will allow me to save time as I will only spend time on what is required.

I also plan on using kanban boards to split the project into smaller parts and work out the priority and order for each part.

My primary research will include an evaluation of the current usability level of Terraform and then an evaluation of my software, so I can then compare them both and evaluate the success of my project.

My secondary research will be for evaluating the usability of cloud computing, in general, to try to identify areas my application will help in and target better.

10. Project plan

- Carry out a literature review - For this, I will be evaluating the current level of usability within cloud computing. This will give me a better understanding of what areas I should focus on to make the biggest difference.
- Do initial research on the usability of Terraform CLI. For this, I will get people to try to perform a task and measure how long it takes them. I will do this with people who are familiar with Terraform and with people who are not.
 - Co-discovery with 2 experienced people and 2 inexperienced people.

3. Establish your requirements
4. Create artefact
5. Do research on the usability of my platform instead.
 - Again using co-discovery with 2 experienced people and 2 inexperienced people
6. Create a survey to get quantitative data on the usability of my platform.
7. *Write the report + Analyse the research to see if it was successful.*

See Appendix B for the detailed Gantt chart.

11. Supervisor Meetings

We are currently having weekly face-to-face 15-minute long meetings, but this is likely to change throughout the duration of the project.

12. Legal, ethical, professional, social issues (mandatory)

- The app will only produce a valid HCL Terraform file which the user will then execute from their command line. The app itself will not be linked to any user cloud accounts and will not generate any infrastructure directly. Storing account data for cloud providers would make my database a valuable target, as each account must have a valid debit or credit card linked.
- I will also store/cache the generated data to see what gets generated for debugging purposes and to prevent the same data from being generated multiple times, which will have the potential to include secrets, credentials, IP addresses/domains and other sensitive data.
 - I could add a way to “Clean” this data, so I remove any possible sensitive values from the generated data.
 - Because of this, I will have to comply with GDPR and the computer misuse act.
- My software will not specifically store any personal or personally identifiable information such as names or emails, as my website won’t require accounts or have any concept of users/profiles.
 - I will need to securely store some personal data from the user participants in my research and comply with GDPR.
 - They will be aware of my research and will have given their permission to participate in my research.
- I will also have to ensure that the frontend side of my application will be secure against XSS and CSRF attacks. I want to use React for my front end, which is already protected against cross-site scripting attacks (XSS). To solve the CSRF issue, I plan on using CSRF tokens with sessions on the backend. Although, since this is a prototype, I may not have enough time to add this functionality.

Appendix A: Ethics certificate

The ethics certificate can be found in

 Certificate of Ethics Review - Bailey Search (UP937683) - TETHIC-2022-103751.pdf



Certificate of Ethics Review

Project title: Improving the Usability of Terraform

Name:	Bailey Search	User ID:	UP937683	Application date:	14/10/2022 10:35:39	ER Number:	TETHIC-2022-103751
-------	---------------	----------	----------	-------------------	---------------------	------------	--------------------

You must download your referral certificate, print a copy and keep it as a record of this review.

The FEC representative(s) for the School of Computing is/are [Haythem Nakkas](#), [Dalin Zhou](#)

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- [University Policy](#)
- [Safety on Geological Fieldwork](#)

It is also your responsibility to follow University guidance on Data Protection Policy:

- [General guidance for all data protection issues](#)
- [University Data Protection Policy](#)

Which school/department do you belong to?: **School of Computing**

What is your primary role at the University?: **Undergraduate Student**

What is the name of the member of staff who is responsible for supervising your project?: **Kirsten Smith**

Is the study likely to involve human subjects (observation) or participants?: Yes

Will you gather data about people (e.g. socio-economic, clinical, psychological, biological?): No

Will you gather data from people about some artefact or research question (e.g. opinions, feedback?): No

Will the study involve National Health Service patients or staff?: No

Do human participants/subjects take part in studies without their knowledge/consent at the time, or will deception of any sort be involved? (e.g. covert observation of people, especially if in a non-public place): No

Will you collect or analyse personally identifiable information about anyone or monitor their communications or on-line activities without their explicit consent?: No

Does the study involve participants who are unable to give informed consent or are in a dependent position (e.g. children, people with learning disabilities, unconscious patients, Portsmouth University students?): No

Are drugs, placebos or other substances (e.g. food substances, vitamins) to be administered to the study participants?: No

Will blood or tissue samples be obtained from participants?: No

Is pain or more than mild discomfort likely to result from the study?: No

Could the study induce psychological stress or anxiety in participants or third parties?: No

Will the study involve prolonged or repetitive testing?: No

Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?: No

Are there risks of significant damage to physical and/or ecological environmental features?: No

Are there risks of significant damage to features of historical or cultural heritage (e.g. impacts of study techniques, taking of samples?): No

Does the project involve animals in any way?: No

Could the research outputs potentially be harmful to third parties?: No

Could your research/artefact be adapted and be misused?: No

Will your project or project deliverables be relevant to defence, the military, police or other security organisations and/or in addition, could it be used by others to threaten UK security?: No

Supervisor Review

As supervisor, I will ensure that this work will be conducted in an ethical manner in line with the University Ethics Policy.

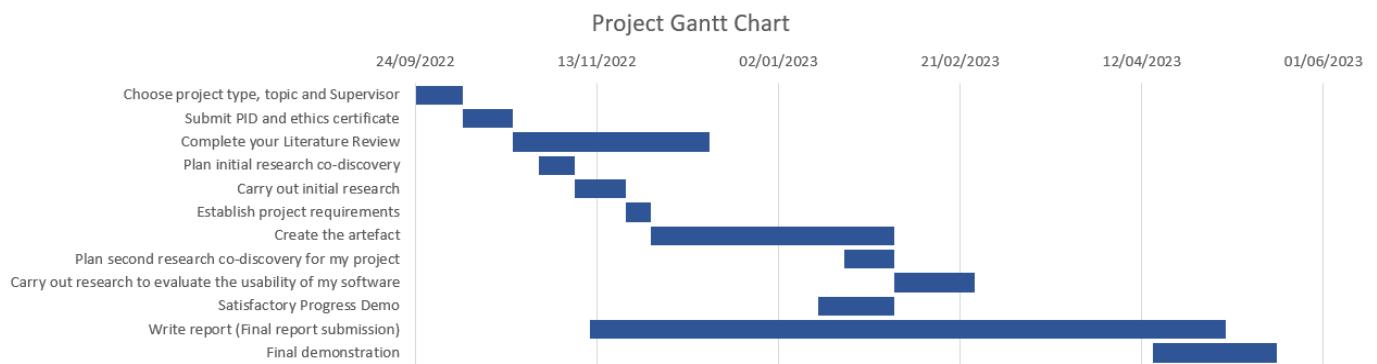
Supervisor comments: **##supervisorComments##**

Supervisor's Digital Signature: **##supervisorSig##** Date: **##supDate##**

Appendix B: Gantt chart

Place here your detailed project plan

Do this in excel for the project plan. Use deadlines to work out specific dates.



Appendix C

Initial Study Co-Discovery Plan & Tasks

C.1 What is Terraform?

- Terraform is an Infrastructure as Code (IaC) tool to easily create reproducible infrastructure in the cloud.
- Uses a declarative language called HashiCorp Configuration Language (HCL).
- Declarative language -> Means you define what the final cloud state should be (What resources should be provisioned and their configurations) and Terraform modifies it to match.
- Terraform also tracks the state of its managed resources and performs updates on them when small changes are made. E.g. Adding a tag to a VM
- 3000 different providers
- Support for: Azure, AWS, GCP, IBM, DigitalOcean, Oracle, Heroku

C.2 Commands

`terraform init` - Initialise terraform and download required providers

`terraform init -upgrade` - Update providers and download any new ones.

`terraform plan` - Show the plan for terraform and any required changes

`terraform apply` - Execute the plan on the cloud service providers.

`terraform destroy` - Delete all terraform-managed resources from the cloud service providers.

C.3 Tasks

C.3.1 Task 1

Using the Azure provider, provision a resource group and any Azure Linux virtual machine inside it.

The file should now contain the definitions for the following resources:

- An Azure resource group
- An Azure Linux virtual machine

Apply the changes.

Docs:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/linux_virtual_machine

C.3.2 Task 2

Using the configuration from the previous task, add a minimal MS SQL server and database to the same resource group and file. (Minimal meaning only the required arguments according to the docs).

Add an optional argument to specify the license type of the SQL database so it is included in the cost.

The file should now contain the definitions for the following resources:

- An Azure resource group
- An Azure Linux virtual machine
- A Azure MS SQL server
- A Azure MS SQL database

Apply the changes.

Docs:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/mssql_database

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/mssql_server

C.3.3 Task 3

Delete the virtual machine from the file and any of its unused subresources (for example, network interfaces etc.) and add a Linux web app to the same resource group. The Linux web app should have no tags.

The file should now contain the definitions for the following resources:

- An Azure resource group
- An Azure MS SQL server
- An Azure MS SQL database
- A Linux web app with no tags.

Apply the changes.

Docs:

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/linux_web_app

https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/service_plan

C.3.4 Task 4

Add a tag to the Linux web app with a key-value pair of “environment” and “staging”.

Apply the changes.

C.3.5 Task 5

Output the default_domain attribute of the Linux web app so you can see where the webpage is.

Apply the changes.

C.3.6 Task 6

Destroy all provisioned resources without removing them from the file.

C.4 Questions for each Participant:

1. Before this study, how much experience do you have with cloud computing?
2. How confident would you be to deploy a small node JS application and database to a cloud provider?
3. What did you like about using Terraform?
4. What did you dislike about using Terraform?
5. What would make your experience using Terraform better?
6. Would you ever consider using Terraform when provisioning resources for a project? Why/Why not?
7. Would a graphical user interface make creating Terraform files easier? Why or why not?

Appendix D

ChatGPT Terraform Test

The screenshot shows a ChatGPT interface. At the top, there is a teal button labeled "BS" and a text input field containing the instruction: "Generate Terraform code to provision a linux web app on Azure". Below this, a response from ChatGPT is displayed, starting with a green icon of a person thinking. The response text reads: "Sure, here's an example Terraform code to provision a Linux web app on Azure:" followed by a "Copy code" button. The code itself is a Terraform script:

```
provider "azurerm" {
  features {}
}

resource "azurerm_resource_group" "example" {
  name      = "example-rg"
  location  = "eastus"
}

resource "azurerm_app_service_plan" "example" {
  name          = "example-appserviceplan"
  location      = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name

  sku {
    tier = "Basic"
    size = "B1"
  }
}

resource "azurerm_app_service" "example" {
  name          = "example-webapp"
  location      = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
  app_service_plan_id = azurerm_app_service_plan.example.id

  site_config {
    linux_fx_version = "DOCKER|nginx"
  }

  app_settings = {
    "WEBSITES_ENABLE_APP_SERVICE_STORAGE" = "false"
  }
}

output "webapp_url" {
  value = azurerm_app_service.example.default_site_hostname
}
```

Figure D.1: ChatGPT defaults to using outdated resources

BS Can you use azurerm_linux_web_app instead?

Yes, you can use `azurerm_linux_web_app` resource instead of `azurerm_app_service` resource to provision a Linux web app on Azure using Terraform.

Here's an example Terraform code to provision a Linux web app using `azurerm_linux_web_app` resource:

```
java
provider "azurerm" {
  features {}
}

resource "azurerm_resource_group" "example" {
  name      = "example-rg"
  location  = "eastus"
}

resource "azurerm_app_service_plan" "example" {
  name          = "example-appserviceplan"
  location      = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name

  sku {
    tier = "Basic"
    size = "B1"
  }
}

resource "azurerm_linux_web_app" "example" {
  name            = "example-linux-webapp"
  location        = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
  app_service_plan_id = azurerm_app_service_plan.example.id

  site_config {
    always_on = true
  }

  app_settings = {
    "WEBSITES_ENABLE_APP_SERVICE_STORAGE" = "false"
  }
}
```

Figure D.2: Guiding ChatGPT to use more up-to-date resources

BS Can you use azurerm_service_plan instead of azurerm_app_service_plan?

Yes, you can use `azurerm_service_plan` resource instead of `azurerm_app_service_plan` resource to provision a hosting plan on Azure using Terraform.

Here's an example Terraform code to provision a Linux web app using `azurerm_linux_web_app` resource and `azurerm_service_plan` resource:

```
java
provider "azurerm" {
  features {}
}

resource "azurerm_resource_group" "example" {
  name      = "example-rg"
  location  = "eastus"
}

resource "azurerm_service_plan" "example" {
  name          = "example-appserviceplan"
  location      = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
  kind          = "Linux"
  reserved      = true
  sku {
    tier = "Basic"
    size = "B1"
  }
}

resource "azurerm_linux_web_app" "example" {
  name          = "example-linux-webapp"
  location      = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
  app_service_plan_id = azurerm_service_plan.example.id

  site_config {
    always_on = true
  }
}
```

There was an error generating a response

Regenerate response

ChatGPT Mar 14 Version. Free Research Preview. Our goal is to make AI systems more natural and safe to interact with. Your feedback will help us improve.

Figure D.3: ChatGPT generates invalid resources and crashes