

UNIVERSITY OF SOUTHAMPTON
Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science

A project report submitted for the award of
MEng Computer Science

Supervisor: Dr Andy Gravell
Examiner: Dr Julian Rathke

**An interactive application for
learning web development**

by **Daniel Berry**

April 23, 2022

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

A project report submitted for the award of MEng Computer Science

by Daniel Berry

This report describes the implementation of an interactive e-learning application for assessing front-end web development skills in a project-oriented manner. The application serves front-end programming projects within an online development environment. An online judge system automatically evaluates user submissions using unit testing. The application demonstrates one paradigm for assessing front-end software development skills that students could use to learn web development and enhance their software portfolios. Alternatively, organisations could use the application to provide web developers with a more relevant, immersive hiring process.

The report contains a literature search to provide context and insight into the field of e-learning for web development and the applications of online judge systems. A detailed description of the technical implementation is provided, outlining and justifying the decisions made during the application's development.

The project was managed using the Scrum methodology, following the principles and values of Agile. The application was implemented over three month-long sprints, and user evaluation was conducted in the fourth sprint. The application was then adapted to user feedback to improve the user experience. All of the primary project goals described in this report were met, and the original schedule was maintained for the project's duration.

Through the development of this application, a strong foundation has been created for an e-learning application that, given additional time and resources, could become an excellent resource for aspiring web developers.

Statement of originality

I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.

I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.

I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

I have acknowledged all sources, and identified any content taken from elsewhere.

I have not used any resources produced by anyone else.

I did all the work myself and have not helped anyone else.

The material in the report is genuine, and I have included all my data, code and designs.

I have not submitted any part of this work for another assessment.

My work involved human participants for the use of user evaluation interviews and questionnaires. Permission for this participation was granted by the Faculty of Engineering and Physical Sciences Ethics Committee on March 2, 2022, with research ethics application reference number ERGO/FEPS/70784.

Contents

Abstract	i
Statement of originality	ii
1 Introduction	1
1.1 Motivation	1
1.2 Scope	1
1.3 Goals	2
2 Background & literature review	3
2.1 E-learning for web development	3
2.2 Online judge systems	4
2.3 Inspirations & existing solutions	5
3 Design & analysis	6
3.1 Requirements elicitation	6
3.2 User interface design	8
3.3 System design	9
3.4 Cost & constraints	10
4 Technical Implementation	12
4.1 Live preview	12
4.2 Modular compiler	13
4.3 Submission evaluation	15
4.4 Project creation tools	17
4.5 Serverless API	18
4.6 Continuous deployment	18
4.7 In-browser file system	19
4.8 Text editor	20
4.9 Notification system	21
4.10 Gamification	22
5 Evaluation	23
5.1 User interviews	23
5.2 User survey	26
5.3 Software testing	27

5.4	Comparative evaluation	29
5.5	Critical evaluation	31
6	Project management	32
6.1	Project plan	32
6.2	Time management	33
6.3	Contingency plan	33
6.4	Repository statistics	35
6.5	Summary of progress	37
6.5.1	Sprint 1	37
6.5.2	Sprint 2	37
6.5.3	Sprint 3	38
6.5.4	Sprint 4	38
6.5.5	Sprint 5	39
7	Conclusion	40
7.1	Reflection	40
7.2	Future work	41
8	Bibliography	42
A	Original project brief	45
B	Supervisor meeting summary	46
C	Design and data archive	47

1 Introduction

1.1 Motivation

A study by [Scaffidi \(2018\)](#) found web development to be one of the most in-demand graduate skills, yet it is a skill that is rarely taught well at a university level, according to [Connolly \(2019\)](#). [Kadriu et al. \(2020\)](#) found that self-taught web developers are increasingly using free video tutorials to learn. However, [Wang and Zahadat \(2009\)](#) suggest that web development is best taught through interactive, project led approaches, stating that “students learn web technologies best by ‘doing’.”

An application offering interactive web development projects that can be completed in the browser addresses these problems: undergraduates could use it to supplement their studies. Without cost or barriers to entry, students may find it a suitable alternative to video tutorials.

1.2 Scope

The focus of this project is to create an application for the creation and completion of web development exercises. The scope of the project is restricted to the implementation and evaluation of the application.

A mobile version of the application will not be implemented during this project because of the limited practicality of code editing on a mobile device. A custom text editor will not be implemented due to the suitability and robustness of existing open-source alternatives.

This project will not support legacy browsers such as Internet Explorer because the application aims to teach modern web development using the latest user interface frameworks. As such, a modern browser is a fair requirement.

1.3 Goals

The MoSCoW prioritised goals for this project are outlined in table 1.1. Where possible, goals are measurable such that progress can be evaluated upon project completion.

ID	Goal	Priority
1	Implement a browser-based IDE that allows for the editing of exercise files.	M
2	Implement a containerised submission environment that evaluates user submissions with unit tests.	M
3	Implement a preview component that allows the user to inspect the user interface they are working on in real-time.	S
4	Implement a project creation suite.	S
5	Create 3 multi-stage projects.	S
6	Support 2 JavaScript user interface frameworks.	S
7	Conduct 5 user interviews and receive 20 anonymous survey responses.	S
8	Add 2 distinct gamification features, such as badges and certificates.	C
9	Provide 75% unit testing coverage of user interface components.	C

Table 1.1: MoSCoW prioritised goals

2 Background & literature review

2.1 E-learning for web development

Knowledge of web development continues to be an essential and desirable skill for computer science graduates ([Aasheim et al. \(2012\)](#)). However, [Connolly \(2019\)](#) identifies that computer science courses rarely offer in-depth web development modules with up to date curricula, writing that the constant evolution of state-of-the-art web development can make it challenging to form a consistent and worthwhile curriculum. Nevertheless, it is in the interest of computer science students to study web development: a study by [Scaffidi \(2018\)](#) found that web development is an essential skill for software engineers seeking employment. In this way, computer science courses that omit web development are not acting in the interest of their students. A reason for the high enrollment rate of students in online courses observed in recent years is to acquire an in-demand skill, showing that motivated students instead seek web development education through online content ([de Oliveira Fassbinder et al. \(2017\)](#)).

Furthermore, the three top non-technical skill gaps in computer science, as identified by [Groeneveld et al. \(2021\)](#), are creativity, continuous learning and solution-oriented thinking. A skill gap is a skill that employers desire but applicants often lack. The developed application addresses each of these non-technical skill gaps whilst improving the technical skills of its users. For example, creativity is encouraged by allowing the user to customise the look and feel of their application. Successful users will apply solution-oriented thinking to design their applications around the requirements.

When users complete an exercise using the developed application, they create a front-end application using tools and frameworks currently used in industry. This process has strong parallels with the tasks a front-end developer is likely to complete in the workplace, adding both context and purpose to the exercises ([Lindley \(2017\)](#)). [Shen and Lee \(2020\)](#) found this approach to work particularly well for learning “programming skills that can be visualised” because of the significant visual feedback derived from a live preview of the user’s work. [Limoncelli \(2017\)](#) emphasises the importance of this relationship, stating that “homework should generate a webpage, not text to the console” because it encourages best practices and better represents some of the most common software engineering tasks. Project-oriented interactive learning enables the student to

understand how multiple technologies and components of an application work together as a system, rather than in isolation or theory.

Organisations could also use this application to assess candidates within the hiring process for front-end development roles. Currently, problem-solving challenges are used as part of the hiring process by companies such as Google¹. Behroozi et al. (2019) argue that this problem-solving style “is not grounded in real-world code, constraints, or scenarios”. An interactive application that allows software engineering applicants to solve problems that are relevant to the role they are applying for is more likely to identify promising applicants (Behroozi et al. (2020)).

2.2 Online judge systems

A key design consideration when developing websites for interactive programming is that an online judge system is required to evaluate user submissions automatically (Wasik et al. (2018)). In the literature, online judge systems are also described as auto-graders or submission servers. The literature addresses the implementation of online judge systems largely in the context of competitive programming websites. As discussed by Pritchard and Vasiga (2013), a common approach is to use a centralised server to run a set of unit tests against the user’s submission. Building on the work by Leal and Silva (2003) in developing a web-based competitive programming system, they use the `safeexec` C++ utility² to execute untrusted user code on the server in a sandboxed environment.

The issue with using a centralised submission server is that if a malicious user were able to perform a privilege escalation attack, the entire submission environment could be compromised, putting the application’s users at risk. Julca-Mejia et al. (2018) found that a submission environment is most vulnerable “when executing or compiling source code that can be harmful to the host system”. To address this issue, Arifin and Perdana (2019) propose a reasonable alternative: performing submission evaluation using the user’s device. This approach is highly scalable and affordable because there will always be as many submission environments as application users, and no cloud computing resources or centralised servers are required. However, this solution is not suitable for this application because each user would have to install the evaluation environment independently. Such a process is uncommon and impractical for web-based applications.

Alternatively, an online judge system can be cloud-hosted using containerised environ-

¹<https://careers.google.com/how-we-hire/>

²<https://github.com/ochko/safeexec>

ments created with tools such as Docker³. Containers are a natural choice for an environment that runs potentially malicious user code because of their total isolation and flexibility of programming language environment (Špaček et al. (2015)). When used in combination with a Containers-as-a-Service (CaaS) utility such as Google Cloud Run⁴, containers can be allocated to inbound requests automatically (Miller et al. (2021)). There is no persistence between submissions, as instances can be spun up and destroyed over a single submission, erasing any damage caused by a malicious user. This approach is secure and affordable, making it an increasingly popular choice for modern online judge systems.

2.3 Inspirations & existing solutions

The main inspirations for this project draw from competitive programming websites such as LeetCode⁵, and browser-based IDE applications such as CodeSandbox⁶ and the Svelte REPL⁷, which allow for the creation and running of JavaScript projects in the browser, inspired the live preview component of the editor.

Popular interactive e-learning websites for web development such as FreeCodeCamp⁸ and Codecademy⁹ offer a 3-panel in-browser IDE with a file explorer, text editor and live preview section. The same structure is used for the application’s design as described in section 3.2. However, one of the main accessibility issues with Codecademy is its aggressive use of monetisation, locking vast quantities of the application’s learning content behind paywalls. The developed application is a free and open resource, allowing any aspiring web developer to complete automatically graded projects.

Another relevant resource is The Odin Project¹⁰, which walks the user through a set of projects in order to become more familiar with web development. However, the content on this site is not interactive. Instead, the user reads the required learning material and completes assignments on their local machine without any evaluation. The application seeks to combine the interactivity of resources such as FreeCodeCamp and Codecademy with the project-led learning approach of tutorials like those on The Odin Project to create a novel learning resource for aspiring web developers.

³<https://docker.com/resources/what-container>

⁴<https://cloud.google.com/run>

⁵<https://leetcode.com/>

⁶<https://codesandbox.io/>

⁷<https://github.com/sveltejs/svelte-repl>

⁸<https://www.freecodecamp.org/>

⁹<https://www.codecademy.com/>

¹⁰<https://www.theodinproject.com/>

3 Design & analysis

3.1 Requirements elicitation

Before creating a prototype, it was necessary to think about the IDE and its principal functions from the user's perspective, so personas were created portraying three distinct users. [Caballero et al. \(2014\)](#) describe personas as a “powerful tool to determine what a product should do and how it should behave”.




Persona	Description
 Jacob, 19	Jacob is a Computer Science undergraduate. He wants to become a software engineer but has no work experience in this field. Jacob is passionate about learning and is happy to refine his skills in his spare time to achieve his goals. Jacob has not completed any relevant projects outside of his course. He wants to build a portfolio of personal projects that he can reference during job interviews.
 Mia, 22	Mia is an experienced competitive programmer. She has a strong understanding of algorithms and data structures. She is interested in learning web development but finds beginner-level content unhelpful as she is already a confident programmer. Mia has multiple displays and uses key bindings and shortcuts wherever possible.
 Oscar, 30	Oscar is a professional software engineer proficient in database systems and server configuration. Oscar is learning front-end development to become a full-stack developer at the same company, as he believes this will allow him to play a more valuable role in the upcoming launch of a new mobile site. He enjoys learning new skills through video-based online courses at his own pace.

Table 3.1: Personas

The personas included in table 3.1 were used to create a set of user stories included in table 3.2. These user stories became the foundation of the initial product backlog.

MoSCoW	Requirement
M	As Jacob, I want to import third-party dependencies into my project so that I can create impressive submissions without reinventing the wheel.
M	As a user, I want to create new files so that I can personalise my submission.
M	As a user, I want to undo and redo changes so that I can amend mistakes.
M	As Jacob, I want to see a live preview of my submission as I am developing it so that I can learn and understand the impact of any changes I make in real-time.
M	As Jacob, I want to download my submissions so that I can include them in my portfolio and use them elsewhere.
S	As Oscar, I want to view console output from my submission so that I can catch any errors and easily debug my submission.
S	As a user, I want to resize the main application components so that they fit my screen in my preferred layout.
S	As Mia, I want to open the live preview in a new window so that I can use the application across multiple displays.
S	As Oscar, I want to change the application's appearance to look the same as the tools I am used to.
C	As Oscar, I want to view the live preview on my mobile device to create responsive submissions that work on various screen sizes.
C	As Mia, I want to change the application's settings to have granular control over the application's default behaviours.
C	As Jacob, I want to receive a unique certificate of completion for each project to have proof of my continuous learning that I can discuss during the job application process.
C	As Oscar, I want to use the application as a standalone editor to try out new ideas quickly.
C	As Jacob, I want the application to automatically generate a portfolio of my submissions and standalone creations so that I can share it with prospective employers.
C	As Mia, I want to control the application using keyboard shortcuts to maximise my productivity.

Table 3.2: MoSCoW prioritised user stories

3.2 User interface design

The application’s user interface was designed and refined iteratively in a process spanning the first three sprints of the project. Initially, only the application’s IDE was designed using the ‘Little Design Up-Front’ (LDUF) approach, as this makes products “easier to learn, easier to use and require less support to be able to use” when incorporated into agile projects (Adikari et al. (2009)).

The low fidelity prototype, shown in figure 3.1, was created during sprint 1 using Whimsical¹, considering the affordances and needs identified through the user stories in table 3.2. In this design, the three principal functions of the development environment, namely, displaying information about the current exercise to the user, providing a text editor, and showing a live preview, are split into three columns.

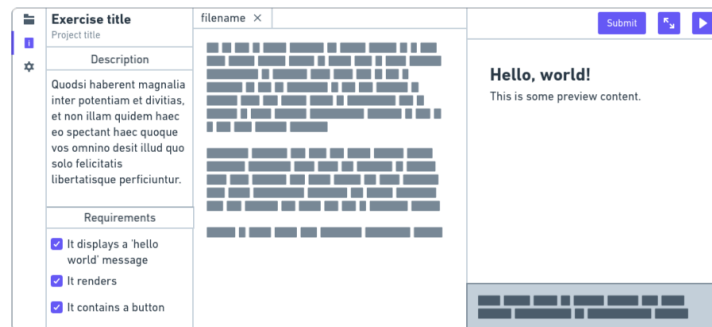


Figure 3.1: Low-fidelity editor prototype produced with Whimsical

Maintaining the LDUF approach, implementation work started immediately during sprint 1 to create this layout. At the same time, the prototype was shared with several Computer Science students at the University of Southampton to perform preliminary user evaluation. Valuable suggestions were provided, such as adding a heading to the preview section and the addition of icon labels. By combining these received comments with a greater depth of understanding resulting from the initial implementation, the user interface design was refined into a high fidelity prototype, shown in figure 3.2, using Figma².

The application was refactored to support multiple interactive projects at the start of sprint 3. The additional pages required for this were designed using the design language established by the high fidelity prototype.

¹<https://whimsical.com/>

²<https://www.figma.com/>

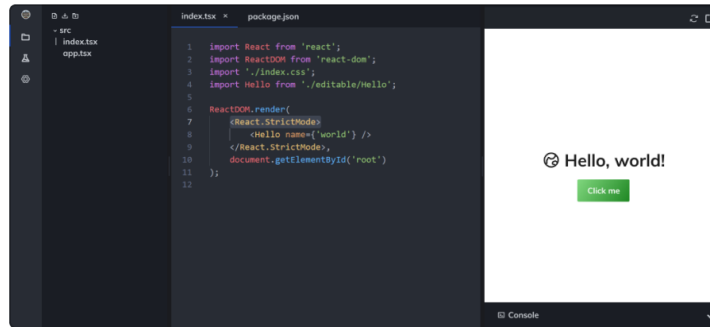


Figure 3.2: High-fidelity editor prototype produced with Figma

3.3 System design

The application is a cloud-based service that uses serverless frameworks and containerisation. These technologies allow for highly scalable, low-cost applications since there is no requirement for the constant running of a centralised server. This approach is particularly advantageous when running a longer computation, such as auto-grading user submissions, because powerful computing resources are expensive to configure and self-host. Instead, these resources are accessed for no longer than the duration of the computation task. For example, [Thorpe et al. \(2021\)](#) found that their serverless solution to neural network training, a traditionally computationally intensive task, outperformed competitors while being an order of magnitude cheaper. [Serth et al. \(2021\)](#) consider this approach state-of-the-art for applications that require auto-graders.

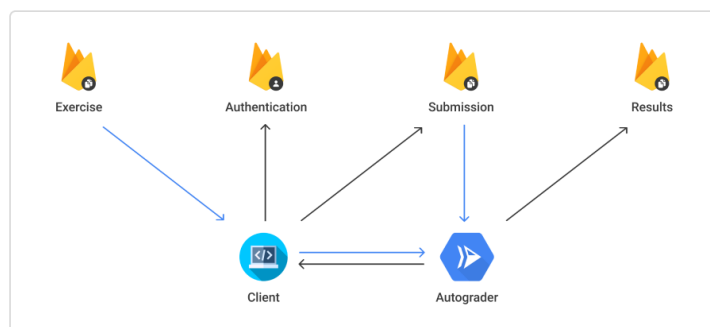


Figure 3.3: System architecture diagram

Firebase³ is used as the application’s user authentication and database provider because it is low cost and has official APIs that integrate with many modern programming languages. An added benefit of this approach is that there is no need to handle user passwords within the application by outsourcing authentication to Firebase.

³<https://firebase.google.com/>

Additionally, Google Cloud Run⁴ is configured to orchestrate the provisioning of auto-grader container instances. Figure 3.3 shows the relationship between each of these services. Firebase is a document-based database, so the relationship between data is not as clearly defined as a traditional table-based relational database. Instead, the structure of the application’s data, shown in figure 3.4, had to be strictly determined before implementation.

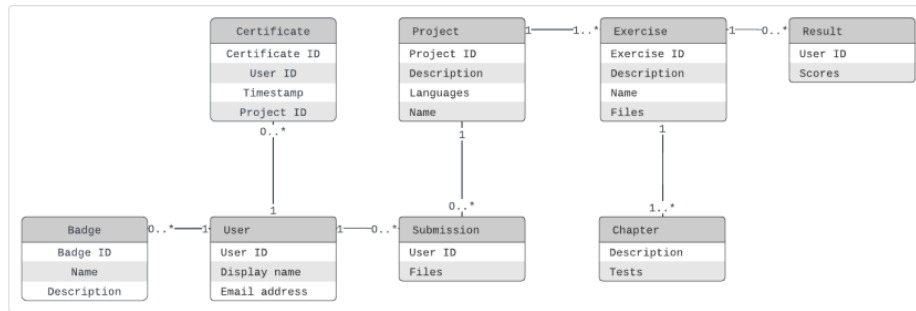


Figure 3.4: Entity relationship diagram

The relationships between data entities are enforced within the application using TypeScript types. TypeScript is used as the primary language for developing the front-end application and the auto-grader server since there is a significant intersection between the data consumed by the two services. Using a shared types library further reduces redundant code.

The user interface is built using the SvelteKit framework. SvelteKit was chosen for this project due to its high performance and excellent developer experience, as reported in the 2021 Stack Overflow developer survey⁵. Notably, SvelteKit uses server-side rendering, which Iskandar et al. (2020) identify as beneficial to search engine optimisation and application performance. SvelteKit applications can also be easily deployed to Vercel⁶, a free hosting provider, using the official plugin⁷.

3.4 Cost & constraints

The decision to use cloud computing resources for the application’s infrastructure requires consideration of potential costs. Firebase and Google Cloud offer a generous free tier,

⁴<https://cloud.google.com/run>

⁵<https://insights.stackoverflow.com/survey/2021>

⁶<https://vercel.com/>

⁷<https://kit.svelte.dev/docs/adapters>

significantly reducing the project cost. Table 3.3 lists the specific prices and constraints of resources used by the application.

Service	Provider	Cost	Constraints
Hosting	Vercel	Free	Up to 100 deployments per day and 100GB bandwidth.
Authentication	Firebase Auth	Free	-
Submission	Google Cloud Run	Free	After the first 180,000 seconds of CPU usage, the cost increases to $\$2.4 \times 10^{-5}$ /second.
API	Firebase Functions	Free	\$0.4/million after 2 million invocations.
Database	Firebase Firestore	Free	\$0.108 per additional GiB after 1 GiB. 20K writes per day, 50K reads per day.

Table 3.3: Application costs and constraints

Resource usage did not exceed the free tier of Firebase or Google Cloud during the development of this project, as the running cost of the application is proportional to the amount of activity on the application. A total of 43 users registered during the evaluation phase, during which time the application incurred no cost.

4 Technical Implementation

4.1 Live preview

When developing a website, it is common to use a dedicated text editor application to write code and a browser to preview the website itself. Serving the editor within the browser opens up new possibilities because any written JavaScript code is implicitly contained within its target environment. This environment synchronisation allows for the concurrent modification, compilation and rendering of code within a single browser window. In the developed application, the live preview component (the rightmost pane in figure 4.1) shows the rendered website built from the user's code. It is updated whenever the user saves their work by listening for changes within the application's file system, as described in section 4.7.

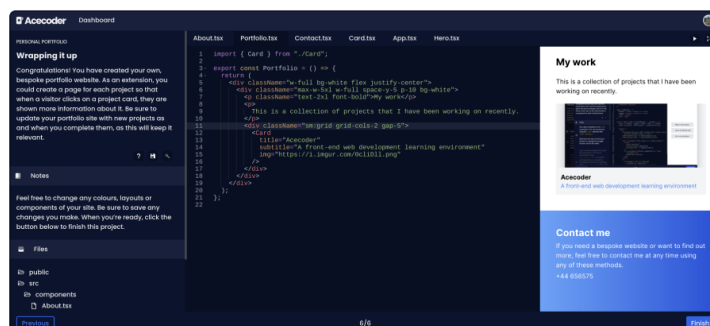


Figure 4.1: The application's IDE, with the live preview rendering the user's project

In order to generate a live preview using JavaScript user interface frameworks, compilation to native JavaScript is often required. This is not a problem from within a server-side development environment: build tools for React and Svelte can be installed with a single command, and a JavaScript bundler such as Webpack¹ can be used to wrap up the user's source into a format that the browser can understand. In the browser, however, there is no such easy solution. Since server-side development is the primary use case for these tools, they are usually written in JavaScript for the Node environment². Although the language is the same, this environment is separate from the browser and offers an entirely different set of global values, built-in tools and functionality. For example, in the browser,

¹<https://webpack.js.org/>

²<https://nodejs.org/>

any JavaScript code can access the global `window` variable, which holds information about the current browser window. No such variable exists for Node since there is no concept of a browser window in a server-side environment.

Environment-specific dependencies must be replaced with suitable alternatives to bridge this gap in environments so that a build tool can be used. The Browserify³ tool automates this process and is used in this project to create a browser-friendly build of Rollup⁴. Rollup is a build tool that bundles JavaScript source code into a single file by resolving dependencies and imported files. The resulting file is sent to an `iframe` tag within the preview component to be rendered as an embedded website within the IDE using JavaScript's `postMessage` API⁵. Additional scripts and tools which should be provided independently of any framework can be included as a link in the HTML of this `iframe` element. In the developed application, this method is used to enable TailwindCSS⁶ preprocessing to improve the developer experience.

The build pipeline for JavaScript frameworks is often a CPU intensive task. As such, it should not run on the main thread as this will decrease the performance of the user interface. Although JavaScript is a single-threaded language, the web worker API can be used in the browser to offload computationally intensive tasks to new threads by creating new browser processes in the background. For example, Wang et al. (2018) found that a web worker implementation can significantly reduce the execution time of complex rendering tasks within a web application. The build pipeline in this application is written to run on a web worker instance. As with the preview, files are sent to and from the web worker using the `postMessage` API.

4.2 Modular compiler

By default, Rollup is only able to bundle native JavaScript, so both frameworks currently supported by the application, React⁷ and Svelte⁸, require compilation into native JavaScript that the browser can execute. React is written using the JSX⁹ syntax, whereas Svelte uses a proprietary syntax and compiler. Compiled source files can then be bundled into a single JavaScript file that correctly references any external dependencies, enabling the user's work to be rendered by the live preview component.

³<https://browserify.org/>

⁴<https://rollupjs.org/>

⁵<https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage>

⁶<https://tailwindcss.com/>

⁷<https://reactjs.org/>

⁸<https://svelte.dev/>

⁹<https://reactjs.org/docs/introducing-jsx.html>

Rollup supports extensions of its functionality with a modular plugin API. This feature was used to write custom plugins that extend Rollup’s functionality, adding support for React and Svelte compilation as part of the build pipeline. In order to compile React, the user’s JSX source code is first passed to the Babel¹⁰ transpiler within a custom plugin, emitting plain JavaScript. Svelte code is passed to a plugin that uses the language’s compiler to produce an output that Rollup can bundle. The importing of standalone CSS files is also enabled through a plugin by concatenating any imported CSS files in the user’s submission and applying the generated stylesheet to the live preview window.

Support for modular extensions allows for the user’s source code to be processed by any JavaScript function that can run in the browser, enabling the simple integration of new frameworks in the future. The plugins to use for any single build are automatically determined based on both the file extensions of the submitted source code and the framework used to complete the current exercise using the `getPlugins` method, as shown in listing 4.1.

```
export const getPlugins = (
  framework: string,
  files: { [key: string]: File },
  dependencies: Record<string, string>
): Plugin[] => {
  // Check the current framework
  switch (framework) {
    // Return the Svelte compiler
    case 'svelte':
      return [svelteCompiler(files, dependencies)];
    // Return the React compiler with CSS support
    case 'react':
      return [
        reactCompiler(files, dependencies),
        cssCompiler(files)
      ];
    // Return no plugins if no framework is selected
    default:
      return [];
  }
};
```

Listing 4.1: Dynamic compiler selection based on framework

While this is a suitable solution to the problem, it still requires some manual work to support new frameworks and is not scalable. Ideally, the application would use the official build tool for each framework with minimal configuration. With the increasing support for the web assembly standard¹¹, allowing for compilation from various languages to low-level machine code that is interpreted by the browser, a build pipeline could be developed to use these tools natively within the browser. However, such an implementation is outside of the scope of this project.

¹⁰<https://babeljs.io/>

¹¹<https://webassembly.org/>

4.3 Submission evaluation

Several approaches to scalable, affordable submission evaluation using online judge systems are discussed in the context of the surrounding literature in section 2.2. A containerised server is widely considered the optimal choice for a cloud-based platform. Therefore, the first step in implementing this project component was to create a server that grades user submissions, which could be containerised once the desired functionality was achieved.

Upon receiving a request from a user for a specific exercise, the server downloads the user's submission for that exercise and the accompanying tests from the database. The exercise's dependencies are then installed using the Node package manager¹². The tests are run against the submission to generate a set of results. These results are written to the database and then returned to the user. This process is illustrated in greater detail in the sequence diagram included in figure 4.2. The commands executed on each submission are also shown in listing 4.2.

```
// Navigate to the unzipped submission
const navigate = 'cd ${dir}';

// Install the submissions's dependencies
const install = 'pnpm install --prefer-offline --silent';

// Run the downloaded tests
const tests =
  'pnpm --silent test -- --json --outputFile=${outputPath}'
  + ' --maxWorkers=50% --silent --watchAll=false'

// Synchronously execute the commands
execSync(
  `${navigate} && ${install} && ${tests}`,
  { stdio: 'inherit' }
);
```

Listing 4.2: Command execution for submission evaluation

The main benefit of this approach is that the online judge system can evaluate any JavaScript project, given a set of unit tests and a `package.json` file listing the required dependencies. In this way, it is language independent and can be reused for any submission on the application, regardless of the user's choice of framework.

The server is written as an HTTP service in Typescript for the Node environment using the Express framework¹³. It is containerised using Docker by pushing the server code onto a basic Node container image. The generated container image can then be uploaded to the Google Cloud container registry to be utilised by Google Cloud Run.

¹²<https://www.npmjs.com/>

¹³<https://expressjs.com/>

Notably, the only network egress from the server is to access the application’s database. Firebase is fully integrated into Google Cloud’s internal network, so traffic from the container never needs to leave this network. Because of this, traffic to the external network outside of Google Cloud can be entirely disabled, reducing the attack surface for malicious users.

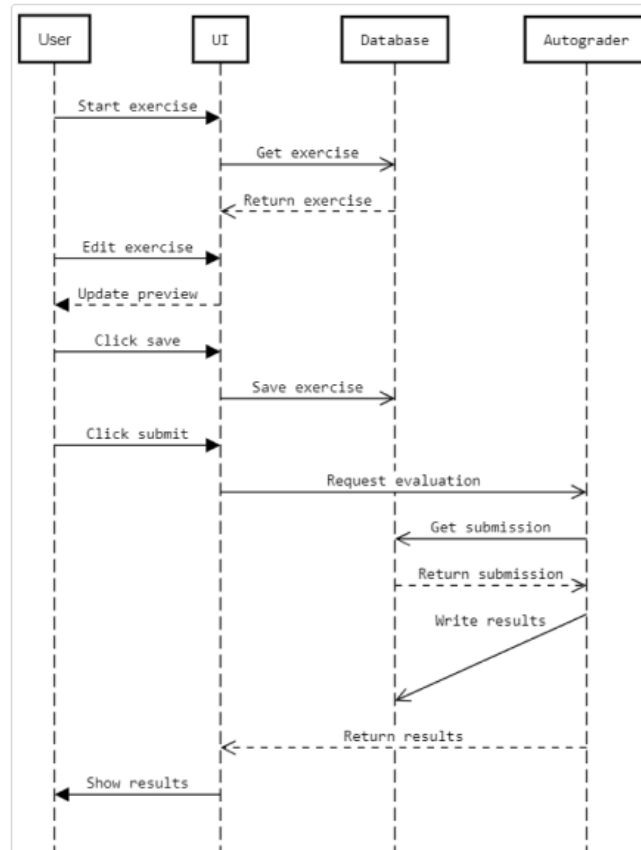


Figure 4.2: Sequence diagram for submission evaluation

In Google Cloud Run, a service is configured to spin up a container image instance for any inbound request. Each instance is allocated two vCPU cores and 2GB RAM. These values were found to be optimal through manual testing: sufficient memory is required to install a submission’s dependencies; unit tests can be run in parallel with 2vCPU cores, speeding up the submission time enough to mitigate any increase in cost. Google Cloud computes container costs based on the total number of vCPU cores and gigabytes of memory used per second, so if doubling the allocated resources halves the run time of a task, the cost of that task does not change.

4.4 Project creation tools

In order to facilitate the addition of new projects to the application during sprint 3, an admin control panel was developed, adding a suite of content creation features for the creation and modification of projects, exercises and achievements. This control panel, shown in figure 4.3, is only accessible to authenticated users that have an admin flag associated with their user profile.

The exercise, project and achievement editors are all forms with a field for each value in the corresponding type, providing an interface for interaction with application data without directly accessing the database. This approach improves the developer experience and allows for safer data modification since before these forms are submitted, their contents are validated, preventing invalid or incomplete data from being written to the database.

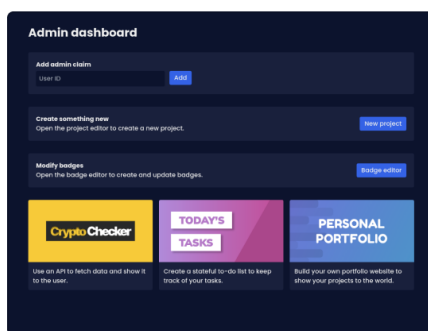


Figure 4.3: Admin dashboard

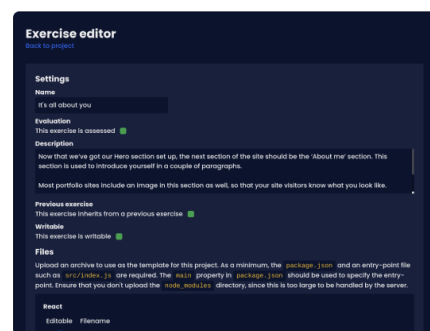


Figure 4.4: Exercise editor

The creation process used to develop the three projects currently on the application starts by first creating an example final submission. Each component of this example is then broken down into an exercise with corresponding unit tests assessing its functionality. The Notion¹⁴ document management system, shown in figure 4.5, is used to define templates for the required prose for both assessed and non-assessed exercises such that it can be quickly composed. Finally, the generated text, exercise files and tests are added to the application using the exercise editor, shown in figure 4.4.

These content creation tools allow for flexible, customisable projects with individual parameters that can be fine-tuned or updated from within the production deployment. Ultimately, this creates a more scalable, maintainable application, allowing for more time to create high-quality projects rather than configuring them.

¹⁴<https://www.notion.so/>

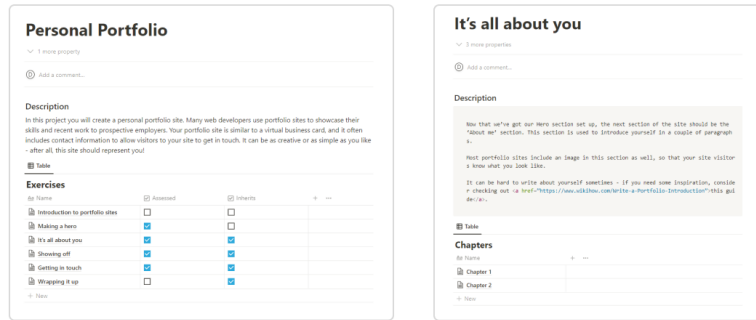


Figure 4.5: Notion document management system

4.5 Serverless API

A serverless API hosted with Firebase Cloud Functions¹⁵ is used within the application to perform updates to application data that should not be performed client-side, such as the awarding of badges and certificates. This protected data can then be marked as read-only from within the application using Firebase security rules¹⁶ to prevent any accidental modification.

Each cloud function is assigned an API endpoint and is only run when a request is sent to that endpoint. As such, there is no need for a constantly running centralised server, significantly reducing the running costs of the application. These endpoints are secured using Firebase authentication such that only authenticated users can use the API. Atomic transactions are used to ensure that sensitive application data is not updated if any errors are detected during the execution of the cloud function. This approach is both secure and highly scalable while minimising application costs.

4.6 Continuous deployment

Southampton Git supports the configuration of continuous deployment pipelines using the GitLab Runner system¹⁷. This tool automatically runs a sequence of actions whenever a new commit is made. Newly committed code is checked against the project's formatting guidelines using Prettier¹⁸ and ESLint¹⁹. Then, the application's unit testing suite is run against the committed code. If all tests pass and the commit was made to the

¹⁵<https://firebase.google.com/docs/functions>

¹⁶<https://firebase.google.com/docs/rules>

¹⁷<https://docs.gitlab.com/runner/>

¹⁸<https://prettier.io/>

¹⁹<https://eslint.org/>

`master` branch, the code is deployed to the application's hosting via Vercel²⁰. This improves productivity by removing the need to update the hosting server after each change manually. It also ensures that evaluation is always conducted on the latest changes such that failing changes are not pushed to the hosting site.

4.7 In-browser file system

One of the application's core functions is to serve and edit files. In the Node environment, JavaScript programs have access to the `fs`²¹ file system library, which would assist with such a task. However, there is no equivalent built-in library for the browser. Instead, an implementation was developed for this project using recursively defined types with Typescript as described in listing 4.3. This custom implementation allows for greater flexibility in how the application interacts with files than any existing third-party library offers.

```
type File {
  value: string;
}
type Folder {
  children: Filesystem;
}
type Filesystem {
  [path: string]: File | Folder;
}
```

Listing 4.3: Recursively defined file system types

With this definition, a top-level `Filesystem` object can be created in the application's state with an arbitrary number of files and folders as children. Wrapping the `Filesystem` in a writable store²² allows the application's components to listen for changes to the filesystem by simply importing this object. This is particularly helpful for updating the live preview whenever the user saves their work, as mentioned in section 4.1, and allows for a shared state between the editor and preview components. A utility library of helper methods was also implemented for more straightforward file creation, deletion, saving and retrieval.

An added benefit of this approach is that it is highly compatible with Firebase Firestore, which is document-oriented, meaning data is stored as collections of JSON documents²³. Files for project exercises and user submissions can also be represented using

²⁰<https://vercel.com/>

²¹<https://nodejs.org/api/fs.html>

²²<https://svelte.dev/tutorial/writable-stores>

²³<https://firebase.google.com/docs/database/web/structure-data>

the `Filesystem` type, allowing them to be serialised as JSON documents which can then be easily stored and retrieved from the database.

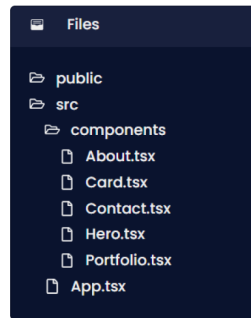


Figure 4.6: The file explorer ‘Tree’ component

To display the current filesystem to the user, a recursively defined `Tree`²⁴ component was created. This component, shown in figure 4.6, is interactive and allows for the deletion, creation and renaming of project files, and can display file systems of arbitrary depth. The component is used in the application’s IDE and within the project creation suite to manage exercise files.

4.8 Text editor

The text editor is one of the most important components of the application as it enables the user to interact with exercises by writing code. However, a full-featured text editor with support for multiple languages, syntax highlighting, and the many developer-experience features that users expect is a complex task beyond the scope of this project. As described in section 1.2, it was decided that a robust, open-source solution should be used instead. Initially, CodeMirror 6²⁵ was used as the application’s editor solution. However, CodeMirror is still in development and was not found to be stable enough for the application as frequent bugs hindered productivity.

Instead, during sprint 3, CodeMirror was swapped for the Ace editor²⁶, a hugely popular, open-source and embeddable code editor with well-written documentation and a large extension ecosystem. A notable feature of Ace is its native support for multiple editor states, which can be hot-swapped to allow for the concurrent editing of multiple open files. Ace also applies CSS classes to each token based on the generated parse tree for

²⁴<https://ant.design/components/tree/>

²⁵<https://codemirror.net/6/>

²⁶<https://github.com/ajaxorg/ace>

the editor contents. For example, constants are tagged with the `.ace_constant` class. A custom CSS theme was written using these classes, matching the editor's appearance with the rest of the application.

Ace supports the addition of custom keyboard shortcuts. Shortcuts were added to save the project with `Ctrl+S` and to automatically format the selected document with `Ctrl+Alt+L`. Automatic formatting was made possible by using a browser build of the Prettier²⁷ formatting tool. Whenever the user activates the format command, the text content of the editor is passed to Prettier along with the current language. The editor is updated with the returned result. Although this approach works for React, there is no working plugin to add automatic formatting support for Svelte. However, the syntax of Svelte is nearly identical to the `Handlebars.js`²⁸ framework, which is supported. This similarity was exploited to support automatic formatting and syntax highlighting for Svelte.

4.9 Notification system

Various asynchronous actions throughout the application, such as saving, formatting or task completion, require a method for reporting success or failure to the user. A notification system was added to the application to display messages to the user programmatically. These notifications are colour-coded to indicate their function, increasing the level of visual feedback these messages provide. For example, information notifications are blue, and error notifications are red.

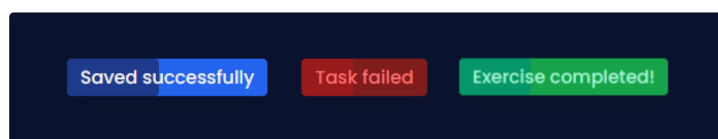


Figure 4.7: Example application notifications

A shared notification state object was implemented such that any script within the application can update the current message. The notification component listens for changes to this state, updating the view with the new message accordingly. Complex user interfaces such as the IDE implemented for this application can have many controls, which can be detrimental to the user experience. By adding clear explanations using the notification system, the consequences of user actions are made explicit, improving the application's ease of use.

²⁷<https://prettier.io/>

²⁸<https://handlebarsjs.com/>

4.10 Gamification

An analysis of gamification mechanics by [Sailer and Homner \(2020\)](#) found that there are “significant, positive effects of gamification on cognitive, motivational, and behavioural learning outcomes”. Such mechanics can often be applied to e-learning applications to create a more engaging experience. Standard gamification features include leaderboards, points, achievements and time-based challenges ([Xi and Hamari \(2019\)](#)). Deciding which features to implement into the application was a non-trivial task because, without sufficient purpose and relevance, they can detract from the user experience, whereas an insufficient quantity of gamification features would be unlikely to improve learning outcomes.

An experience and level progression system was initially considered for the application, rewarding the user with experience points after completing an exercise, which progresses them closer to the next level. With just three projects on the application, the user could only accumulate a small quantity of experience, limiting the sense of progression.

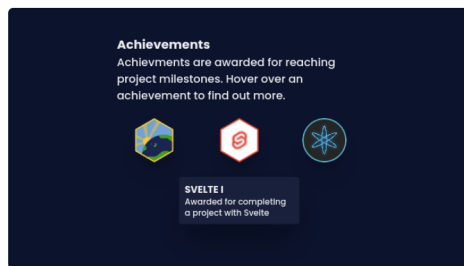


Figure 4.8: Project achievements



Figure 4.9: Certificate of completion

Instead, achievements and certificates were implemented as two separate systems. Achievements, shown in [figure 4.8](#), are awarded to the user for completing milestones independent of any project. These include completing a single project in a new language or completing a fixed number of projects. A document recording each user’s progress statistics is stored in the application’s database and updated whenever the user completes a project. When updated, the user’s statistics are evaluated to determine they have unlocked any new achievements, and these are added to the user’s profile if so.

Similarly, unique certificates are awarded to the user upon completion of a project. Each certificate is created with a public URL to be shared with external users. These certificates were integrated with [LinkedIn²⁹](#), enabling users to share them with their professional network. Certificates can also be downloaded as an image on the user’s device.

²⁹<https://www.linkedin.com/>

5 Evaluation

5.1 User interviews

Semi-structured user interviews were held during sprint 4 to improve the shared understanding of the application. After conducting five interviews, the majority of relevant codes were identified, and a point of saturation was reached (Francis et al. (2010)). Jakob (2000) advises five users as a sensible evaluation limit due to the diminishing returns of interviewing additional users. Each interview was recorded, transcribed and inductively coded to extract qualitative insights that can be applied to improve the application. Several of the key findings from these interviews are grouped by their code in table 5.1.

Code (count)	Examples
Experience (10)	I started web development for the first time this year. I haven't used Svelte before.
Learning (8)	I learned React by following along with tutorials from YouTube. I mainly learn programming using online videos.
Suggestions (15)	Collaborative coding could be very interesting. I think it would be a bit nicer if [your work is] saved when you submit it.
User interface (8)	The colour between the active tab and the non-active tabs is quite similar. The reset button looks like it should rerun the code.
Performance (8)	Submitting an exercise took so long the first time I did it. Moving on to the next task was quite slow.
Projects (7)	The basic structure of the project is going to be the same for everyone. My styling is probably quite different to other users.

Table 5.1: Example interview quotes grouped by code

Participants demonstrated a wide range of web development abilities and experience. The relevant suggestions and feedback received during this process were MoSCoW prioritised and can be found in table 5.2.

A tutorial detailing the application's controls was the most requested feature during the interview process, as although many users were able to use the application without any explanation, some features went unnoticed. This feature was implemented and is shown

in figure 5.1. User interface feedback was highly positive, as shown by comments such as “I thought it was really intuitive” and “I like the colour theme especially”. The critical area for improvement identified by participants was the application’s performance. In particular, the time taken to generate results after exercise submission and the time taken to navigate between exercises should be improved.

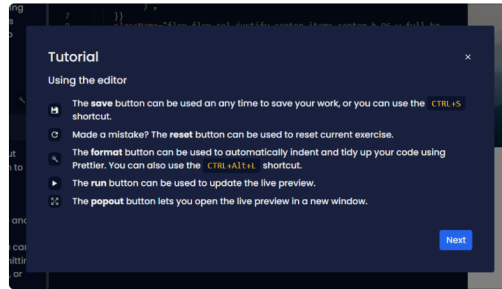


Figure 5.1: The tutorial pop-up, explaining the core functions of the editor

When asked about the uniqueness of the projects that participants created on the application, they identified that the structure of the projects would not vary much between users. By contrast, participants agreed that a project’s design is highly flexible, and in this sense, the level of uniqueness is down to the individual.

Regarding future projects that could be added to the platform, participants unanimously requested a mixture of smaller projects to familiarise themselves with new concepts and the application itself, along with a set of longer projects to demonstrate their skills and create something worth sharing.

MoSCoW	Suggestion
M	Skip package resolution to reduce submission time.
M	Reduce the time taken to navigate between exercises.
M	Fix broken external links within the application.
S	Create a tutorial section
S	Automatically save projects when submitted.
C	Add progress messages for pending submissions.
C	Enable editor auto-complete.
W	Add collaborative coding support.
W	Increase the contrast between open file tabs.
W	Add a range of shorter and longer projects.

Table 5.2: MoSCoW prioritised interview suggestions

All suggestions without ‘W’ priorities in table 5.2 were implemented during sprint 4, demonstrating how the agile value of “responding to change over following a plan” was observed in this project (Beck et al. (2001)). For example, skipping the package resolution step when installing submission dependencies by including a pre-generated `package-lock.json`¹ file in each exercise decreased submission time by 9 seconds on average. Allocating more memory and a second CPU core to submission container instances led to further improvements. Inter-exercise navigation was refactored to determine a user’s exercise permissions at the point of database access using the Firebase rules language², as shown in listing 5.1.

```
allow update:
  // The user must be authenticated
  if request.auth != null
  &&
  // The user must be updating their own progress
  request.auth.uid == uid
  &&
  (
    // The user must be incrementing their progress
    int(request.resource.data.progress) == int(resource.data.progress
    ) + 1
    &&
    // The user must have passed the exercise if it is assessed
    (
      getAssessed() == false ||
      (
        getChapterCount() in getResults() &&
        getResults()[getChapterCount()][‘passed’] == true
      )
    )
  )
  ||
  // The user has already completed this project
  (
    request.resource.data.completed == true
    &&
    resource.data.progress == getExerciseCount()
  )
```

Listing 5.1: Firebase rule for updating user progress

Previously, Firebase Cloud Functions were used to perform such validation. Cloud Functions offered poor user performance when used for frequent actions such as exercise navigation due to frequent cold starts. This change made navigation between exercises nearly instantaneous instead of taking several seconds with the original approach.

¹<https://docs.npmjs.com/cli/v8/configuring-npm/package-lock-json/>

²<https://firebase.google.com/docs/rules>

5.2 User survey

An 11-question anonymous survey was created with Qualtrics³ and shared with users of the application. Each question was answered with an 11-point Likert scale, with possible answers from 0 (strongly disagree) to 10 (strongly agree). The survey received 20 responses. The survey was shared within web development communities online and among computer science students at the University of Southampton. A single-sample t test was conducted on the survey results to determine their significance, with null hypothesis $\mu = 5$, significance level $\alpha = 2.50 \times 10^{-2}$ and 19 degrees of freedom. The findings of this test are shown in table 5.3. Significant results require a t -value greater than the critical value of 2.09.

Question	μ	σ	t	p
How experienced are you with web development?	5.330	2.74	0.490	3.15×10^{-1}
How familiar are you with JavaScript frameworks such as React and Svelte?	4.55	3.20	-0.628	7.31×10^{-1}
How likely would you be to recommend the application to someone else?	8.25	1.59	9.17	1.05×10^{-8}
How familiar did you find the text editor interface of the application?	7.75	1.97	6.24	2.69×10^{-6}
How would you rate the ease of use of the application?	8.55	1.10	14.4	5.33×10^{-12}
How would you rate the accessibility of the application?	7.80	1.32	9.47	6.25×10^{-9}
How would you rate the loading and submission performance of the application?	6.55	2.74	2.53	1.02×10^{-2}
How would you rate the design of the application?	8.55	1.50	10.6	1.09×10^{-9}
How unique do you feel that your completed projects are?	6.30	2.77	2.10	2.49×10^{-2}
How likely would you be to include a completed project in your portfolio?	6.40	3.39	1.84	4.03×10^{-2}
How useful is this application for front-end developer interview preparation?	7.45	2.56	4.27	2.06×10^{-4}

Table 5.3: Single-sample t -test of survey responses

The data for each participant's self-reported level of experience and familiarity with JavaScript frameworks was noisy, with a significant standard deviation ($\mu = 5.33$, $\sigma = 2.74$). This suggests a wide range of experiences and abilities, which helps gather results representative of all users. Participants rated the ease of use ($\mu = 8.55$, $\sigma = 1.10$),

³<https://www.qualtrics.com>

accessibility ($\mu = 7.80$, $\sigma = 1.32$), editor familiarity ($\mu = 7.75$, $\sigma = 1.97$) and design ($\mu = 8.55$, $\sigma = 1.50$) of the application very highly, suggesting that the iterative design process used for this project worked successfully.

The loading and submission times for exercises were identified as an area requiring improvement ($\mu = 6.55$, $\sigma = 2.74$). This issue was also identified in user interviews. The submission time is constrained mainly by the cost of cloud computing hardware used by the submission environment.

Participants agreed that the application would be significantly useful for job interview preparation ($\mu = 7.45$, $\sigma = 2.56$). However, no significant result was reached for the inclusion of completed projects in portfolios ($\mu = 6.40$, $\sigma = 3.39$). A key reason for this, as identified in user interviews, is due to the relative simplicity of the tutorial project, ‘Personal Portfolio’, used by most users for evaluation. A more sophisticated project could be worthy of such inclusion. However, this simpler project was deemed suitable for evaluation since it did not require significant time and effort from participants.

5.3 Software testing

Unit testing was used to ensure software quality and detect breaking changes throughout the application’s user interface development. Since a component-oriented approach was used to develop the user interface by defining its fundamental building blocks and reusing them wherever possible, each component can be considered a fundamental unit that can be tested in isolation. 52 tests were written in TypeScript with Jest⁴ spanning 17 test suites, with each suite testing a different component.

File		Statements		Branches		Functions		Lines	
components/auth	<div><div></div></div>	83.33%	15/18	63.63%	7/11	100%	0/0	81.25%	13/16
components/explorer	<div><div></div></div>	78.89%	172/218	53.46%	54/101	57.14%	8/14	79.47%	151/190
components/common	<div><div></div></div>	83.06%	103/124	52%	52/100	100%	2/2	84.54%	93/110
components/profile	<div><div></div></div>	92.3%	12/13	50%	2/4	0%	0/1	88.88%	8/9
components/loaders	<div><div></div></div>	100%	4/4	40%	2/5	100%	0/0	100%	4/4
components/preview	<div><div></div></div>	67.79%	40/59	37.03%	10/27	0%	0/2	71.15%	37/52

Figure 5.2: Coverage report

The coverage report in figure 5.2 summarises the extent of unit testing implemented for user interface components, with an average coverage of 84.2%. Automated integration

⁴<https://jestjs.io/>

testing was also used to verify that the application's systems and components work correctly in unison. The Cypress⁵ framework was used to achieve this. 15 automated tests were written for the application's core functions, including authentication and exercise submission. For both unit and automated tests, all tests pass successfully.

```
import { login } from './utils';
describe('Authentication', () => {
  it('The user is able to log in', () => {
    login();
    cy.contains('Get started creating eye-catching, responsive
    websites.');
```

```
  });
  it('Logging in with an invalid email shows an error', () => {
    login('testtest.com');
    cy.contains('Invalid email');
```

```
  });
  it('The user is able to log out', () => {
    login();
    cy.get('div[aria-label="open profile menu"]').click();
    cy.contains('Log out').click();
    cy.contains('Welcome back');
```

```
  });
});
```

Listing 5.2: Integration test for login functionality

Lighthouse⁶ was used to automatically audit the performance, use of best practices and accessibility of the various pages within the application. The results of the audit are summarised in the table 5.4. Lighthouse generates a score for each category from 0 to 100. Values between 50 and 89 are considered acceptable, and values between 90 and 100 are considered very good.

Page	Accessibility	Performance	Best practices
Landing	97	82	100
Login	97	82	100
Dashboard	100	67	100
Project page	100	65	100
Exercise page	93	57	92
Profile	100	65	100
Certificate	100	72	100

Table 5.4: Lighthouse audit scores

In order to improve performance scores, Lighthouse suggests using code splitting and dynamic imports where possible such that dependencies are only loaded when needed

⁵<https://www.cypress.io/>

⁶<https://developers.google.com/web/tools/lighthouse>

and converting `png` images into a web-optimised format such as `webp`⁷. Although code splitting was not implemented due to the large refactor it would require, all `png` images on the application were converted to the `webp` format, which was found to improve loading times.

5.4 Comparative evaluation

The application combines functionality found in several existing products to produce a unique, interactive e-learning experience. Relevant components of the application are compared to similar existing implementations to conduct a comparative evaluation.

CodeSandbox⁸, shown in figure 5.4, is an exemplar for the transpilation and live preview of JavaScript code with support for UI frameworks and third-party dependencies. It uses a modified version of the hugely popular Webpack build tool⁹ to process user code from within the browser. A benefit of this approach over Rollup, as used in the developed application, is that it allows for running JavaScript unit tests from within the browser.

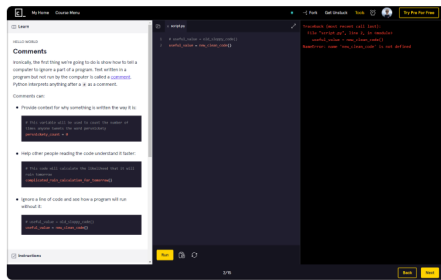


Figure 5.3: Codecademy

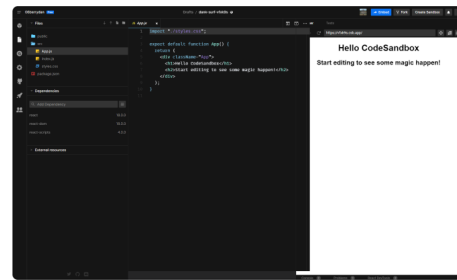


Figure 5.4: CodeSandbox

Evaluating user submissions client-side would likely offer significantly improved performance over an online judge system. However, client-side evaluation is more susceptible to cheating as malicious users could modify the evaluation result before it is returned. A notable feature of CodeSandbox is that it allows users to create and share templates for supporting new tools and frameworks, resulting in a significantly more extensible system than the application described in this report.

A widely used application for learning programming online is Codecademy¹⁰, shown in figure 5.3. Similar to the developed application, it uses a text editor and preview window

⁷<https://developers.google.com/speed/webp>

⁸<https://codesandbox.io/>

⁹<https://github.com/CodeSandbox/sandpack>

¹⁰<https://www.codecademy.com/>

to serve web development content. Rather than compiling user code client-side, it connects each user to a sandboxed container instance that offers a server-side environment natively supporting any required build tools. Such an approach is unique in that it supports an unlimited number of languages, allowing Codecademy to offer courses beyond web development with traditionally server-side languages such as Java. However, this approach is costly since a container instance is required for each concurrent user. Such an approach is too expensive for the developed application without monetisation.

Codecademy offers an extensive catalogue of high-quality interactive exercises and tutorials aimed mainly at the beginner level, written by a dedicated team of content writers. The educational value of the developed application in its current state is far lower than that of a resource such as Codecademy due to the limited quantity and depth of the application's projects. With sufficient time and resources to produce new projects targeting a range of abilities, the application could be enhanced to provide a similar offering to the top e-learning resources for web development, such as Codecademy.

The gamification and competitive programming aspects of the application are similar to those found on the HackerRank competitive programming site ¹¹. Both applications use badges and certificates of completion to reward the user. HackerRank also uses a points system to provide additional extrinsic motivation for exercise completion, but these points have little purpose within the application and do not deliver significant value to the user. HackerRank does include a social mechanic, allowing users to follow and send messages to each other. Although such functionality is missing from the developed application, user profiles are public and can be shared with their URL to share progress and achievements. Improving the social and competitive aspects of the application to be more in line with those found on a site such as HackerRank would likely create a more enjoyable and engaging user experience.

It is to be expected that an application produced in several months by an individual lacks the quantity of content and features found in successful existing products. For example, Codecademy was acquired at the end of 2021 for \$525 million¹². The resources available to a business of this scale are incomparable to those accessible for this project. However, it is promising that the developed application is still substantially comparable to these products. A strong foundation has been created that could become an excellent e-learning resource given additional time and resources.

¹¹<https://www.hackerrank.com/>

¹²<https://www.zdnet.com/education/skillsoft-acquires-codecademy-for-525-million/>

5.5 Critical evaluation

The primary goals for this project are included in section 1.3, each labelled with an integer index. All of these goals were met during the completion of this project. These goals are adapted from the original project brief, which can be found in appendix A. An explanation of how each goal was met is provided below, enumerated with the goal's index. A reference to the report section containing each goal's justification is also included.

1. A browser-based IDE has been successfully implemented. Files can be loaded from any exercise on the application, edited and saved (4.7, 4.8).
2. The containerised submission environment that evaluates user submissions with unit tests was created using design choices justified by the surrounding literature (4.3).
3. The preview component uses in-browser compilation, providing a real-time render of the current exercise (4.1).
4. A project creation suite supporting the configuration and modification of projects and exercises was designed and implemented during sprint 3 (4.4).
5. 3 unique projects, each with at least five stages, were created (4.4).
6. Both React and Svelte are supported, and projects are served using both of these frameworks (4.2). The compiler system is modular to support the addition of new frameworks in the future.
7. Five user interviews were conducted during sprint 4, in which time the required 20 questionnaire responses were collected (5.1, 5.2).
8. During sprint 3, badges and certificates were added as gamification features to the application (4.10).
9. 84.2% unit testing coverage was provided across the application's user interface components (5.3).

In retrospect, it would have been better to use strictly measurable goals, as meeting a general implementation goal has no constraints on the quality or functionality of that implementation. However, these goals were sufficient in providing direction and measuring progress throughout the project, and by meeting them, an application has been created that is faithful to the description in the original project brief.

6 Project management

6.1 Project plan

The Scrum methodology described by [Sutherland and Schwaber \(2007\)](#) was used for the management of this project, following the values and principles of Agile. The project was divided into 5 sprints, each 4 weeks in duration, as shown in the project timeline in figure 6.1.

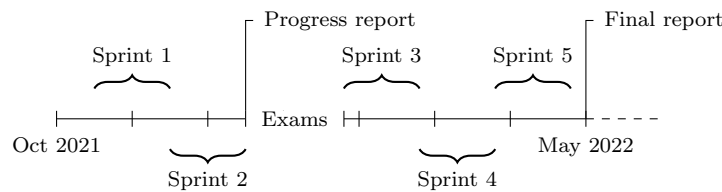


Figure 6.1: Project timeline showing the 5 project sprints

A pause in progress was observed during the exam period. A more detailed breakdown of the project timeline is provided in table 6.1, describing the start and end dates of each sprint along with an overview of the work completed during each sprint. Detailed sprint summaries can be found in section 6.5.

#	Focus	Start	End	Description
1	Editor	20/10/21	16/11/21	Implement the text editor and live preview.
2	Projects	17/11/21	17/12/21	Extend the application to serve projects.
3	Gamification	28/01/22	25/02/22	Add project creation, certificates and badges.
4	Evaluation	25/02/22	25/03/22	Conduct user evaluation interviews, implementing feedback. Start writing the final report
5	Finalisation	25/03/22	25/04/22	Conclude writing the final report.

Table 6.1: Project schedule with sprint descriptions

Through rigorous time logging, weekly supervisor meetings and task duration estimation, the original project plan was successfully followed for the project’s duration without modification. Weekly documents were created for supervisor meetings to report progress and document remaining work. An example of one of these documents from during sprint 3 can be found in appendix B.

6.2 Time management

For this project, the GitLab issues system¹ shown in figure 6.2 was configured to represent the product backlog, keep track of time spent on tasks and estimate time remaining.

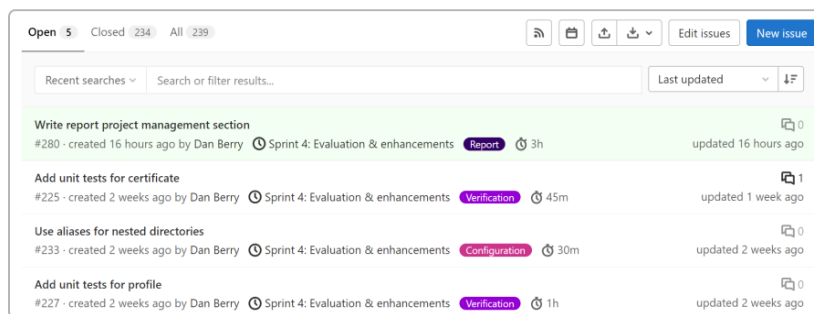


Figure 6.2: The GitLab issues system

Milestones describe important dates within the project, such as the end of each sprint or report deadlines. Issues represent outstanding tasks and can be assigned to milestones. Each has a title, description, associated milestone, due date and time estimate. Custom labels can be added to issues for further classification and analysis.

The issues for a given milestone in GitLab can be exported as a CSV file. A data visualisation script was created in Python to automatically generate the statistics and burndown charts at the end of each sprint, found in section 6.5, from this data.

6.3 Contingency plan

A contingency plan was created for the project to improve the probability of product success. When determining the project schedule shown in figure 6.1, sprint 5 was intentionally left as a buffer for the project in case of insufficient progress. This encouraged the creation of a draft final report by the end of sprint 4 and left sufficient time for refinement of the report during sprint 5. Furthermore, this contingency buffer mitigates any single project risk that could cause a loss of at most 4 weeks of progress.

The project risk assessment, shown in table 6.2, was completed to mitigate the potential impact of significant project risks. In this case, project risk R is calculated as a function of loss L and probability p : $R = p \times L$.

¹<https://docs.gitlab.com/ee/user/project/issues/>

Problem	<i>L</i>	<i>p</i>	<i>R</i>	Plan
Illness	2	2	4	Allow a contingency period before the final deadline to accommodate any project time lost.
Loss of application files	5	1	5	Store a backup of application files in a second location other than Southampton Git to allow for some redundancy. Keep a local copy of the application files.
Breaking changes in dependencies	2	2	4	Use Node package manager combined with version control to track dependency versions and roll-back breaking changes where necessary.
Loss of development hardware	1	1	1	If development hardware is lost or destroyed, any uncommitted changes will be lost. Changes should be committed at least once per day of development to minimise the potential loss.
Insufficient weekly progress	2	2	4	Insufficient progress is limited to a single week due to weekly meetings with the project supervisor. Goals can be re-prioritised with the project supervisor to ensure deliverables are met.
Change of supervisor	1	3	3	Weekly progress reports should be well documented and retained to provide to a new supervisor if required.
Ethics approval not received in time for evaluation	2	2	4	Prepare required ethics approval forms as early as possible. Determine some evaluation techniques that do not require ethics approval, such as anonymous questionnaires.
Loss of report files	5	1	5	The report is stored on Overleaf. A Southampton git repository should be created to synchronise with the Overleaf project, adding a level of redundancy. A local copy of the report should also be kept.
Deprecation of dependencies s	2	1	2	Dependencies included in the final application should only be used where necessary. Multiple candidates should be identified such that an alternative can be substituted for a deprecated library.

Table 6.2: Project risk assessment

A skills audit was also completed at the start of the project to support the justification of key decisions. This audit identified weaknesses that, once addressed, improved the probability of project success. Table 6.3 shows the skills audit. Skills are rated from 1 to 5, where 1 is 'little to no skill' and 5 is 'highly skilled'. 9 essential project skills were audited.

Skill	Rating	Description
TypeScript	4	TypeScript is used as the principal programming language for this project.
Firebase	3	Firebase is used as a database and user authentication provider.
Svelte	4	Svelte is a JavaScript user interface framework used for the project's front-end.
LaTeX	3	LaTeX is a document preparation system used for writing the report.
Writing	4	Writing to convey information in a clear, concise manner.
Referencing	2	Referencing is used to provide sources for external information.
Testing	4	Software testing is used to functionally evaluate the application. This will be done using Jest and Svelte Testing Library .
Figma	4	Figma is used to design high fidelity prototypes.
Docker	2	Docker is used to create a containerised online judge system.

Table 6.3: Project skills audit

The weakest identified skills which introduced a level of project risk were academic referencing skills and containerisation using Docker. Additional time was allocated to tasks requiring these skills throughout the project to address this risk.

6.4 Repository statistics

Repository statistics were generated from the application's source code using GitLab repository analytics and the GitStats tool². A total of 150 commits were made to the project, culminating in 10510 lines of code (excluding configuration files) by the end of the project. The distribution of commits over time is shown in figure 6.3, where darker shades indicate a higher level of project activity. 181 files were created throughout the project. A large refactor at the start of sprint 3 resulted in a decrease in file count visible in figure 6.4. However, the addition of new gamification features and the implementation of evaluation feedback further increased the file count in sprints 3 and 4.

²<http://gitstats.sourceforge.net/>

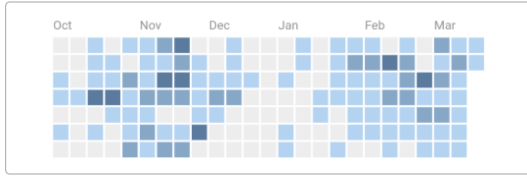


Figure 6.3: Heat-map of repository activity

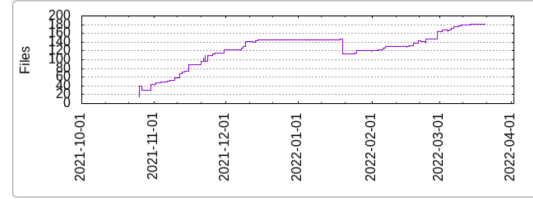


Figure 6.4: Repository file count

73 svelte components were created, contributing to 40.33% of the total file count. TypeScript is used for the submission server, back-end cloud functions and utility scripts within the application, comprising another 27.07% of the project.

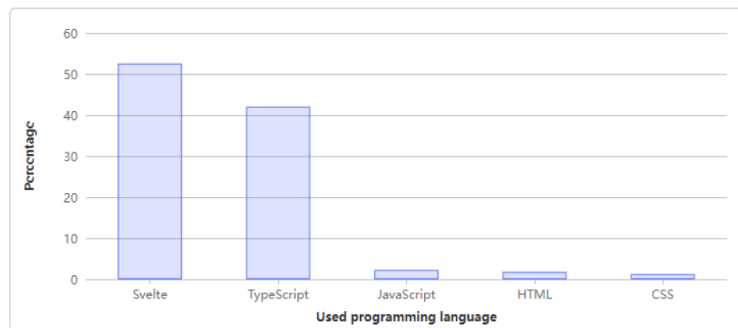


Figure 6.5: Bytes of code by language

Figure 6.5 shows the bytes of code by language for this project, excluding generated and vendor files. As with the distribution of languages by file count, Svelte and TypeScript make up most of the project’s codebase. Regular expressions were used to generate some additional useful statistics from within the project’s directory, shown in table 6.4.

Type	Count
UI Component	52
Function	215
Type	23

Table 6.4: Object counts within the project’s source

Types were used to model data handled by the application, such as submissions or projects. User interface components were frequently refactored into fundamental components for reuse, following the component-driven pattern used for this project.

6.5 Summary of progress

Project progress has been documented since commencement on October 20, 2021. This includes time spent and outstanding issues for implementation work, report writing and miscellaneous tasks. The following sections provide a summary of the work completed, a burndown chart and a chart of remaining issues over time for each sprint. 324 hours of work have been logged on GitLab during this project. The actual time spent is likely to be greater than this due to preparation work completed before the time management system was established and weekly supervisor meetings, for which time was not logged.

6.5.1 Sprint 1

51 tasks were completed over 70 hours. The application's IDE was successfully designed and implemented, providing a proof of concept and foundation for continuing the project.

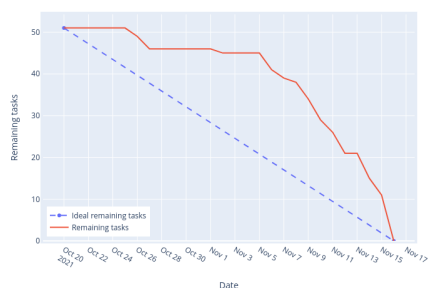


Figure 6.6: Burndown chart

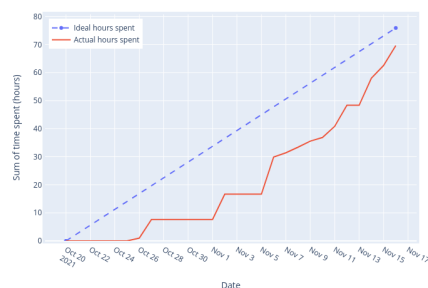


Figure 6.7: Cumulative time spent

It took several weeks to establish a practical schedule and daily workload for the project during this initial sprint, resulting in an uneven distribution of remaining tasks in the burndown chart shown in figure 6.6.

6.5.2 Sprint 2

70 tasks were completed over 66 hours. The online judge system was created and integrated with the application, and the interim progress report was written. The tasks for this sprint were completed slightly earlier than expected. However, this allowed sufficient time to refine the interim progress report, which was due simultaneously.

At the end of this sprint, the application contained a single demo project which allowed for submission using the online judge system. This left the application in a strong position

for adding content during the next sprint and allowed for a natural breakpoint to focus on exams.

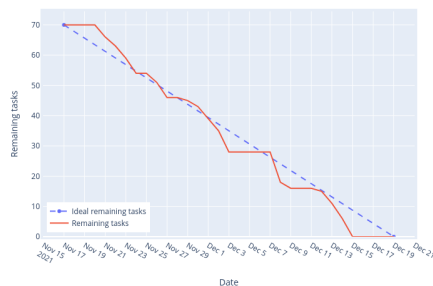


Figure 6.8: Burndown chart

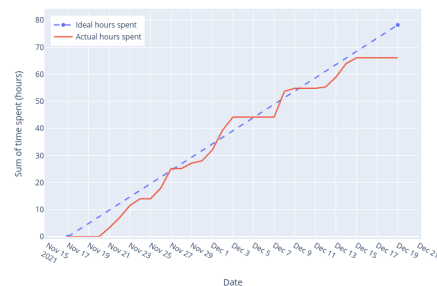


Figure 6.9: Cumulative time spent

6.5.3 Sprint 3

70 sprint backlog tasks were completed in 94 hours. The estimated time spent for this sprint was 78 hours. At the start of the sprint, an extensive application refactor was completed to reduce scope creep and tidy up any work completed in the previous two sprints.

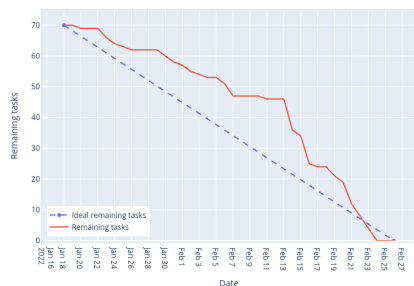


Figure 6.10: Burndown chart

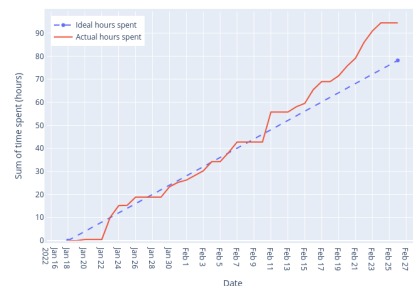


Figure 6.11: Cumulative time spent

The creation and testing of projects took significantly longer than expected, causing an underestimate of the total time required for this sprint. However, all of the tasks from the sprint backlog were still completed by the end of the sprint.

6.5.4 Sprint 4

64 sprint backlog tasks were completed in 62 hours. The estimated time spent for this sprint was 61 hours. During this time, 5 user interviews were conducted, work started on the final report and user suggestions such as adding a tutorial were implemented into the

application. Task duration estimation during this sprint was significantly better than in previous sprints.

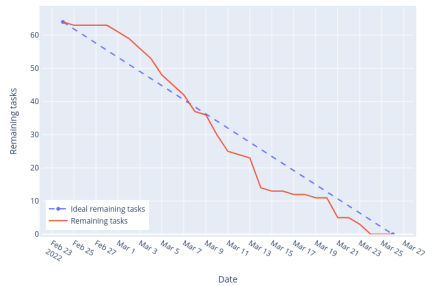


Figure 6.12: Burndown chart

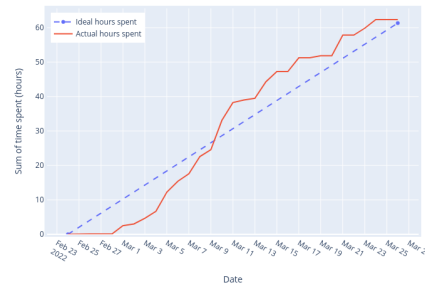


Figure 6.13: Cumulative time spent

The research ethics application required for user evaluation was not approved until March 2, 2022, explaining the lack of progress at the start of this sprint.

6.5.5 Sprint 5

The focus of this sprint was to finish the final report and complete the project. 13 report-related tasks were completed in 27 hours, making the average task duration twice as long as in the previous sprint.

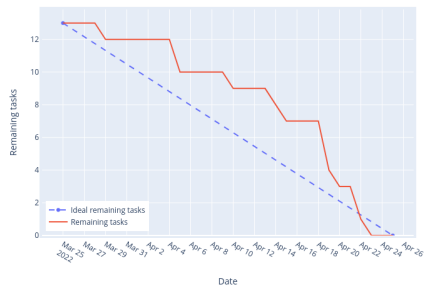


Figure 6.14: Burndown chart

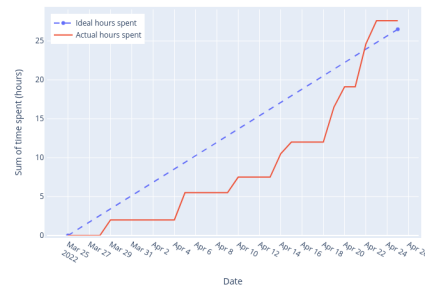


Figure 6.15: Cumulative time spent

This sprint was left as a buffer in the contingency plan, and the total allocated time was not required to complete the project due to progress made in previous sprints. Because of this, the time spent during this sprint is significantly lower than in any other sprint.

7 Conclusion

This project aimed to create a web application that serves interactive web development projects in the browser. The project was managed in an agile way. The application was implemented over three month-long sprints, and user evaluation was conducted in sprint 4. The application was then adapted to user feedback to provide a better user experience. All of the project goals described in section 1.3 were met, and the original project schedule was maintained for the duration of the project.

7.1 Reflection

Initially, I was doubtful that I would be able to implement the live preview and the on-line judge system for the application within the allocated time. These ambitious features introduced a reasonable level of project risk, making the project consistently challenging and interesting whilst maintaining sensible goals. However, these features, both prerequisites for later work, required implementation at the start of the project. This made the first two sprints disproportionately challenging, distributing project risk unevenly. Ideally, more time should have been spent identifying task dependencies and prioritising work accordingly to spread the most challenging tasks across the project.

The project goals were successfully achieved within the given time constraints. Given this, combined with the wide range of positive feedback received from user interviews and the significance of the questionnaire responses, I think that the project has been well-executed, and the finished application is highly satisfactory and pride worthy. However, many decisions were made during the application’s development that directly affected the project’s outcome. It is a worthwhile exercise to consider whether, in retrospect, these were optimal decisions and what could have been done differently.

For example, the application supports both the React and Svelte user interface frameworks. However, during evaluation, very few users opted to use Svelte. This is expected as Svelte is a far newer and less well-known framework than React. React is the most used web framework globally, with a 40.14% market share. Svelte, by comparison, has only a 2.75% market share ([Statista \(2021\)](#)). Instead of supporting multiple frameworks, a better choice might have been to only target React. I could have then used the time spent configuring projects in numerous languages and developing the modular compiler

system to improve the user experience with React and further optimise the application.

Another core decision for this project was whether users should be able to create new projects on the application. The natural incentive of this approach is that the developer only needs to focus on application features, outsourcing content creation to users. However, the main issue is quality control. Creating an exercise is a complex process that requires a working project as a template and is generally prone to human error. Spending time verifying and maintaining user-created exercises to ensure the average quality of projects on the application does not decrease could take more time than the potential savings gained through outsourcing. For this reason, user-generated content was deliberately not enabled in the developed application.

7.2 Future work

The primary limiting factor of the application is now the number of exercises available to the user. This project has created a framework for the running, editing and creation of these exercises; however, the task of developing new exercises remains non-trivial. Several users indicated that they would like to see a range of smaller and longer format projects during user evaluation. Such a change would undoubtedly improve the quality and value of the application as a learning resource.

The monetisation of the application is a viable option for future work, enabling access to more performant cloud computing resources to decrease submission times. However, the implementation of monetisation should be carefully considered as it can be detrimental to the accessibility and priorities of a learning application ([Daniel et al. \(2015\)](#)).

A wider range of content should ideally be available on the application. The projects created on the application currently target individuals seeking to demonstrate their existing skills. However, the application could benefit from a selection of beginner-friendly educational resources that use the live preview component to teach web development in a kinesthetic manner.

Another future consideration is the range of user interface frameworks supported by the language. As described in section [4.1](#), new technologies such as web assembly would likely enable a more powerful compilation pipeline for the application while also improving performance. The popularity and quantity of JavaScript user interface frameworks frequently change, so a more straightforward, modular approach to adding new frameworks should be incorporated to create an application that stays relevant for years to come.

8 Bibliography

- Aasheim, C., Shropshire, J., Li, L., and Kadlec, C. (2012). Knowledge and skill requirements for entry-level it workers: A longitudinal study. *Journal of Information Systems Education*, 23(2):193–204.
- Adikari, S., McDonald, C., and Campbell, J. (2009). Little design up-front: A design science approach to integrating usability into agile requirements engineering. In *International Conference on Human-Computer Interaction*, pages 549–558. Springer.
- Arifin, J. and Perdana, R. S. (2019). Ugrade: Autograder for competitive programming using contestant pc as worker. In *2019 International Conference on Data and Software Engineering (ICoDSE)*, pages 1–6. IEEE.
- Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al. (2001). Manifesto for agile software development.
- Behroozi, M., Parnin, C., and Barik, T. (2019). Hiring is broken: What do developers say about technical interviews? In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1–9. IEEE.
- Behroozi, M., Shirolkar, S., Barik, T., and Parnin, C. (2020). Debugging hiring: What went right and what went wrong in the technical interview process. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, pages 71–80. IEEE.
- Caballero, L., Moreno, A. M., and Seffah, A. (2014). Persona as a tool to involving human in agile methods: contributions from hci and marketing. In *International Conference on Human-Centred Software Engineering*, pages 283–290. Springer.
- Connolly, R. (2019). Facing backwards while stumbling forwards: The future of teaching web development. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 518–523.
- Daniel, J., Vázquez Cano, E., and Gisbert Cervera, M. (2015). The future of moocs: Adaptive learning or business model? *International Journal of Educational Technology in Higher Education*, 12(1):64–73.

- de Oliveira Fassbinder, A. G., Fassbinder, M., Barbosa, E. F., and Magoulas, G. D. (2017). Massive open online courses in software engineering education. In *2017 IEEE Frontiers in Education Conference (FIE)*, pages 1–9. IEEE.
- Francis, J. J., Johnston, M., Robertson, C., Glidewell, L., Entwistle, V., Eccles, M. P., and Grimshaw, J. M. (2010). What is an adequate sample size? operationalising data saturation for theory-based interview studies. *Psychology and health*, 25(10):1229–1245.
- Groeneveld, W., Vennekens, J., and Aerts, K. (2021). Identifying non-technical skill gaps in software engineering education: What experts expect but students don’t learn. *ACM Transactions on Computing Education (TOCE)*, 22(1):1–21.
- Iskandar, T. F., Lubis, M., Kusumasari, T. F., and Lubis, A. R. (2020). Comparison between client-side and server-side rendering in the web development. In *IOP Conference Series: Materials Science and Engineering*, volume 801, page 012136. IOP Publishing.
- Jakob, N. (2000). Why you only need to test with 5 users. *Nielsen Norman Group, Nielsen*.
- Julca-Mejia, W., Calderon-Vilca, H. D., and Cárdenas-Mariño, F. C. (2018). Evaluation of source code in acm icpc style programming and training competitions. In *Avances en Ingenieria de Software a Nivel Iberoamericano, CibSE 2018*.
- Kadriu, A., Abazi-Bexheti, L., Abazi-Alili, H., and Ramadani, V. (2020). Investigating trends in learning programming using youtube tutorials. *International Journal of Learning and Change*, 12(2):190–208.
- Leal, J. P. and Silva, F. (2003). Mooshak: A web-based multi-site programming contest system. *Software: Practice and Experience*, 33(6):567–581.
- Limoncelli, T. A. (2017). Four ways to make cs and it more immersive. *Commun. ACM*, 60(10):50–52.
- Lindley, C. (2017). Frontend developer handbook 2017. *Frontend masters*.
- Miller, S., Siems, T., and Debroy, V. (2021). Kubernetes for cloud container orchestration versus containers as a service (caas): Practical insights. In *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 407–408. IEEE.
- Pritchard, D. and Vasiga, T. (2013). Cs circles: an in-browser python course for beginners. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 591–596.

- Sailer, M. and Homner, L. (2020). The gamification of learning: A meta-analysis. *Educational Psychology Review*, 32(1):77–112.
- Scaffidi, C. (2018). Employers’ needs for computer science, information technology and software engineering skills among new graduates. *International Journal of Computer Science, Engineering and Information Technology*, 8(1):1–12.
- Serth, S., Köhler, D., Marschke, L., Auringer, F., Hanff, K., Hellenberg, J.-E., Kantusch, T., Paß, M., and Meinel, C. (2021). Improving the scalability and security of execution environments for auto-graders in the context of moocs. In *Proceedings of the Fifth Workshop” Automatische Bewertung von Programmieraufgaben”(ABP 2021), virtual event, October 28-29, 2021*.
- Shen, R. and Lee, M. J. (2020). Learners’ perspectives on learning programming from interactive computer tutors in a mooc. In *2020 IEEE symposium on visual languages and human-centric computing (VL/HCC)*, pages 1–5. IEEE.
- Špaček, F., Sohlich, R., and Dulík, T. (2015). Docker as platform for assignments evaluation. *Procedia Engineering*, 100:1665–1671.
- Statista (2021). Most used web frameworks among developers worldwide, as of 2021.
- Sutherland, J. and Schwaber, K. (2007). The scrum papers. *Nuts, Bolts and Origins of an Agile Process*.
- Thorpe, J., Qiao, Y., Eyolfson, J., Teng, S., Hu, G., Jia, Z., Wei, J., Vora, K., Netravali, R., Kim, M., et al. (2021). Dorylus: Affordable, scalable, and accurate {GNN} training with distributed {CPU} servers and serverless threads. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 495–514.
- Wang, Y. D. and Zahadat, N. (2009). Teaching web development in the web 2.0 era. In *Proceedings of the 10th ACM conference on SIG-information technology education*, pages 80–86.
- Wang, Z., Deng, H., Hu, L., and Zhu, X. (2018). Html5 web worker transparent offloading method for web applications. In *2018 IEEE 18th International Conference on Communication Technology (ICCT)*, pages 1319–1323. IEEE.
- Wasik, S., Antczak, M., Badura, J., Laskowski, A., and Sternal, T. (2018). A survey on online judge systems and their applications. *ACM Computing Surveys (CSUR)*, 51(1):1–34.
- Xi, N. and Hamari, J. (2019). Does gamification satisfy needs? a study on the relationship between gamification features and intrinsic need satisfaction. *International Journal of Information Management*, 46:210–221.

A Original project brief

An interactive web application for learning front-end development

Student: Daniel Berry

Supervisor: Dr Andy Gravell

Brief

Problem

There exists a wide range of beginner tutorials on e-learning platforms that individuals can complete to acquire new skills, although it can be difficult to develop new skills past this point.

Furthermore, students often lack professional experience and so require an alternative way of demonstrating their skills when interviewing for software engineering roles.

Aim

This project aims to provide an interactive e-learning application for the development of intermediate level front-end development skills in a project-oriented manner. The application should provide the user with projects that allow them to better demonstrate their proficiency whilst gaining useful experience and knowledge.

For any given project on the application, there will be a set of functional requirements that must be met to progress. However, the design and implementation is left up to the individual, such that on completion the user will have a unique submission to incorporate into their portfolio.

Goals

The below goals have been assigned a priority using MoSCoW prioritisation, outlining the expected results of this project and corresponding importance:

Goal	MoSCoW
Develop an in-browser IDE that allows the user to write front-end code for supported languages and frameworks.	M
Provide exercises that can be loaded into the IDE. An exercise consists of a description, a list of requirements and a set of unit tests corresponding to the requirements.	M
Develop an environment to securely evaluate user submissions by running a set of unit tests corresponding to the requirements for the attempted exercise.	M
Create a database to store application information such as user submissions and results.	M
Provide a preview section within the application's IDE, allowing the user to inspect the the user interface they are working on in real time.	M
Evaluate the functionality of the application through automated testing.	M
Wire-frame a low-fidelity user interface prototype.	M
Conduct a literature review of published research and existing, similar applications.	S
Implement user authentication within the application.	S
Evaluate the application through the use of anonymous questionnaires and focus groups.	S
Provide unique certificates which allow the user to demonstrate completion of a project.	C

B Supervisor meeting summary

Student: Daniel Berry
Supervisor: Dr Andy Gravell

15 Feb, 2022

1 Project creation

In order to facilitate the creation of new projects, a suite of project creation tools were added to the application. Specifically, there is a form to create a new project, to create a new exercise for a project and to create a new achievement. The application has been updated to include designs for these pages.

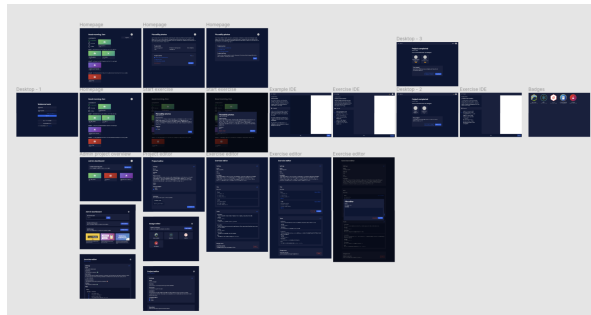


Figure 1: Application storyboard

During the addition of the application's second project, these tools were tested and updated based on any identified issues, which made the creation of this project take significantly longer than expected.

2 Gamification

Adding gamification features to the application continues to be a focus of this sprint. For example, vibrant project thumbnails have been created to make the application appear more fun. Animations have also been added to the application where possible to improve the user experience and make it more enjoyable to use.



Figure 2: Project thumbnails

Certificates of completion still need to be implemented into the application. Other gamification features such as a levelling up system or a leader-board do not fit the application well at this time due to the limited catalog of projects, so will not be implemented.

C Design and data archive

The design and data archive included alongside this report provides supporting evidence and documentation of the work completed during the application’s development. Table [C.1](#) details the contents of each directory in this archive. Instructions on how to build, run and deploy the project are also provided in the `src/app` directory.

Directory	Description
<code>design</code>	Low-fidelity and high-fidelity storyboards used to design the application.
<code>documents</code>	Documents prepared for weekly supervisor meetings.
<code>evaluation</code>	Interview transcripts, consent forms and research ethics documents from the application’s evaluation process.
<code>src</code>	Source code for the application’s user interface, cloud functions and online judge system.
<code>visualization</code>	Python code and exported GitLab data from each sprint used to generate project management charts used in the report.

Table C.1: Design and data archive contents