University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

AutoAssign – Online Coding Assignment Automation Platform

Will Murphy
March 28, 2024

Abstract

Coding skills are now in high demand due to their relevance in top jobs and growing digitalisation across industries. As student interest in computing increases and classes grow in size, it has highlighted the need for more effective ways to teach and assess coding. This paper proposes *AutoAssign*, an all-in-one assignment automation platform that handles the assignment process from start to finish. More specifically, it provides standardised tooling for assignment creation, unit testing, and submission comparison. Through a comprehensive user evaluation, it was found that users not only find the platform intuitive but also appreciate the ease with which it integrates into existing course workflows.

Education Use Consent

Contents

1 Introduction

1.1 Motivation

In the early days of computers, proficient coding ability was a rare talent reserved mostly for cutting-edge software development and research purposes. However, today the landscape is vastly different. In 2022 it was reported that, in terms of salary, satisfaction, and availability, 10 of the top 20 'best' jobs in the UK were related to computing or coding [1]. With more and more industries leveraging digitalisation to optimise business processes [2], it is no surprise that the demand for coding skills has been steadily on the rise for a long time.

These statistics do not appear to be lost on anyone. Student interest in computing courses continues to grow [3], and for those no longer at university, online learning platforms such as *Codeacademy* have seen meteoric success reaching over 45 million users in under 10 years of operation [4].

This increased demand for coding education has highlighted the need for more efficient ways to deliver and assess programming assignments. Current methods, which often rely on manual marking, are time consuming and inconsistent, especially when it comes to larger courses with more students. For teachers and lecturers, this manual approach can often prove too lengthy; this has led to the neglect of various vital parts of the learning process, such as personalised feedback.

The lack of student-teacher engagement has not gone unnoticed by students, who have reported increased levels of stress surrounding their coding assignments due to the perceived lack of adequate support from staff [5]. Through this, it is clear that there is a problem with the current approach of the coding education system and it is herein that the motivations for this project lie.

1.2 Aims

In light of the above motivations, this project proposes *AutoAssign*, an online platform to automate the creation, grading and feedback processes of assignments. The website aims to standardise, simplify and streamline all 3 of the aforementioned stages as follows:

Creation
Lecturers and teachers on the platform will be presented with a clear and straightforward interface to create assignments. This will include specifying a title and problem description, as well as optional extras such as enforcing a language to be used or providing a skeleton code template. This process will not only be easy for the creators of the assignments, but assignees will also benefit in that all their assignments will be presented in the same format.

Marking
Various marking tools will be provided to the creators of the assignments. At a fundamental level, the platform will support automated evaluation based on predefined criteria. This predefined criteria will come in the form of provided unit test files. Additionally, the platform will monitor execution time and memory usage statistics, providing lecturers additional ways to assess their student's submissions. This flexibility ensures that submissions can be accurately assessed in a way that truly reflects the effectiveness of the solution.

Feedback
The platform will take the hassle of guidance and feedback away from lecturers/teachers and provide a trained AI assistant for students to engage with. This model will be instructed to provide guidance rather than solutions, encouraging the learning process. Additionally, assignment creators can define feedback for individual unit tests that will be used should they fail. This tooling will significantly alleviate the pressure on lecturers/teachers of large classes while still providing students with sufficient feedback to help them learn.

In addition to streamlining these 3 standard stages of coding assignments, this platform also aims to introduce a new element: competition. Through the monitoring of average execution time and memory consumption, a leaderboard will be generated for each assignment. After an assignment's deadline has passed, all submissions will be ranked and visible to all assignees. Authors of submissions on the leaderboard will be able to opt to remain anonymous if they wish. The goal of this functionality is to inspire students to seek not just the most basic solution to a problem, but a more optimised one. This is a vital part of coding in the real world and should therefore be encouraged throughout all stages of the learning process.

2 Background

This chapter will discuss the background of the project by examining related products or solutions that have similar goals or functionality to *AutoAssign*. The chapter will then go over how it is possible to automatically assess code and bring in related research to propose improvements.

2.1 Existing Products

2.1.1 Overview

There are a wide variety of programs and platforms available that automatically assess code. The most popular of these technologies can be split into two types: educational platforms and recruitment tools. The first type is understandably geared more towards supporting learning and, therefore, has an emphasis on feedback and collaboration. The second is more industrial and competitive with features such as wide language support and participant ranking. Four examples were chosen for this evaluation that all have similar goals to *AutoAssign* and encapsulate different feature sets that can be drawn from. The ultimate goal of the evaluation is to learn from existing solutions and later apply that knowledge to create an educational site with not only the supporting environment of the learning platforms, but also the competitive aspect of the recruitment tools.

2.1.2 Gradescope by Turnitin

Turnitin is primarily an established plagiarism detection platform that leverages AI to find similarities in submissions [6]. One of their many products is *Gradescope*, geared towards automatically assessing assignments of all types this platform aims to streamline the grading and feedback process for educators [7]. One type of assignment that can be automatically graded with *Gradescope* is coding problems.

As an industry leader, *Gradescope* is feature rich, with facilities for:

- GitHub and Bitbucket code submission integration
- Automated unit testing
- Optional manual grading
- Inline comments for feedback

Advantages

- The assignment creation process is straightforward. The refined interface shared between all *Turnitin* products is easy to understand and most likely quite familiar to many educators.
- There is easy configuration for how heavily weighted the automatic grading should be compared to its manual counterpart. This allows for a wide variety of marking styles.
- Being built on top of *Turnitin*'s vast underlying plagiarism detection framework, the platform is extremely good at catching copy and pasted code.
- The inclusion of analytics helps instructors identify trends, such as which questions students struggled with the most. This type of information can be important when it comes to refining future assignments or providing additional support on challenging topics.

Disadvantages

- The autograder requires manual configuration that is difficult without specific technical knowledge. This can be extremely discouraging for educators who are seeking a straightforward solution.
- *Gradescope* is a paid platform. The costs can be a significant budget consideration for some institutions and, in particular, independent third parties in the market.
- Code quality can only be assessed manually. In many cases, although a submission may fulfil automated unit testing requirements, it will still not be worthy of full marks. This is especially important when assessing algorithm implementations, where efficiency is paramount.
- Students can only obtain feedback if the assignment creator manually adds inline comments. This is not particularly common in large classes of hundreds of students and can lead to a lack of support and increased levels of confusion among students who perform poorly on assignments.

Summary
Gradescope is a mature and feature-rich platform that, after its acquisition by *Turnitin* [8], has excellent plagiarism detection functionality. It has proven to be effective and is used by many educators around the world in major institutions. However, it is unfortunately locked behind a paywall and therefore is less accessible to smaller organisations or individuals.

2.1.3 HackerRank

HackerRank is an all-in-one solution to screen, interview, and hire for coding roles [9]. It features a variety of assessment tools, one of which is automatically graded coding problems. Recruiters can either choose from a variety of industry-standard questions, or create their own custom ones fit for the role for which they are hiring. The interface is quite similar to popular coding interview practice sites such as *LeetCode*, minimising candidate confusion.

Advantages

- The platform boasts extensive language support, with over 40 languages being available. This opens up the product to a much greater market and could attract a lot more organisations.
- *HackerRank* has a diverse library of pre-existing problem sets. This allows recruiters with less technical knowledge to easily create assignments for potential employees.
- The site leverages AI to detect forms code plagiarism and AI usage. This is a highly sought after feature in the recruitment space as it can help highlight true candidates for a position.
- There is extensive tooling on the site to compare and contrast candidates and their submissions. This allows for recruiters to make clear distinctions in who may be suitable for a role.

Disadvantages

- There is no functionality on the platform to provide feedback to candidates or their submissions. This means that those who submit failing code will not be able to learn from their mistakes unless they figure it out themselves.
- The site relies entirely on predefined test cases when it comes to determining a mark for a submission. This does not allow much flexibility in assignments.
- There is no support for the assessment of code quality. This is a key skill when it comes to professional software development and yet it lacks assessment on a platform that is intended to hire for such environments.

Summary
HackerRank is a successful recruitment tool adopted by many top organisations in their interview processes. At its core, it leverages automatic code assessment to streamline the candidate screening process. Key features that contribute to its success are the extensive tools provided to compare candidates and the extensive list of supported languages.

2.1.4 GitHub Classroom

Arguably the most similar to this project, *GitHub Classroom* is an online tool to automate the assessment of programming assignments [10]. It has a structure similar to the widely adopted *Google Classroom*, with students being invited to class groups and assignments being set within them. It has functionality for both individual and collaborative assignments and has direct integration with the *GitHub* platform.

Advantages

- Due to direct integration with *Git*, version control is natively supported for student submissions. This allows students to observe failing tests and correct their mistakes. This also allows assignment creators to understand where students may be getting stuck.
- The site has extensive functionality for providing feedback, with assignment creators being able to request changes, give line-by-line feedback, or add general comments on the code. This allows educators to easily guide the learning process in whatever manner they wish.
- The automated test creation process is straight forward and not much additional technical understanding is required. The simple and form-like process means that educators do not have to spend hours learning how to use the platform before creating their first assignment.

Disadvantages

- The marking process is solely based on the automated tests, there is no functionality for assessing code quality or optimisation. This means that the platform is not particularly flexible when it comes to grading certain types of assignments.
- Due to the platform being owned by *GitHub*, all educators and students are forced to use the site. Some may prefer other workflows and ways of managing their code.
- The platform does not take steps to combat plagiarism. This means that students could easily copy and paste or AI generate code for their submissions.
- The user interface is based heavily on the existing *GitHub* website, which was designed for a different purpose. Because of this, some of the interfaces feel somewhat cluttered with additional buttons and functionality.

Summary
GitHub Classroom leverages a massive existing platform to help manage and grade coding assignments in a more efficient manner. Educators can create, distribute, track, grade, and collect assignments with greater efficiency over the manual counterpart. Its functionality for group projects means support for a variety of assignment types, however, the over reliance on automated testing leave much to be desired in terms of assessment.

2.1.5 CodeAcademy

CodeAcademy is one of the most popular online code learning platforms [11]. It teaches programming skills through interactive lessons and hands-on projects with automatic assessment. Although its main focus is individual learning, the site also provides tools for educators and institutions to guide students through structured courses.

Advantages

- The ability to write, run, and test code directly in the browser and receive instant feedback means that students can quickly and repeatedly iterate on their solutions. This approach allows for quick building of understanding through trial and error.
- The platform includes gamification elements, such as points, badges, and streaks, to encourage engagement. This keeps students motivated and helps maintain consistent practice, which is critical in coding education.
- *CodeAcademy* has respectable language support, with 12 of the most popular ones being supported. This allows the platform to be relevant to a variety of coding courses.

Disadvantages

- While *CodeAcademy* includes a wide range of problem sets, educators cannot create their own custom problems. This means that any teachers or lecturers seeking to set assignments of their own will have to look elsewhere.
- The platform does not assess code quality, only predefined test cases. This means that important factors in the learning process will go unchecked unless manually reviewed by educators.
- The site does not have a built-in plagiarism detection system. This means educators must be vigilant or rely on external tools to ensure students submit original work.
- Although the platform offers free courses, its advanced features and content, including real-world projects and skill paths, are available only through a paid subscription. This can be a barrier for schools or individuals with limited budgets.

Summary

CodeAcademy is a robust platform that fosters self-paced learning through interactive lessons and a user-friendly environment, making it a solid choice for introducing programming skills. However, the platform's limited assessment capabilities and lack of plagiarism detection may not meet the needs of institutions seeking more comprehensive grading and evaluation tools.

2.2 Leveraging AI to Automate Feedback

The practice of automatically checking code is not a recent development by any means. Automated tests in CI/CD pipelines have been a critical part of development cycles for many years. However, this approach is purely functional and is designed only to catch errors before they enter live environments. This process is insufficient for education, as simply telling a student that their code fails tests offers little guidance on next steps to fix it and learn.

As mentioned in the introduction of this dissertation, the aim of this platform is to alleviate pressure from teachers and lecturers by automating the entire coding assignment process. This includes the feedback stage. After initial brainstorming, it became clear that the ideal solution to this automation problem would be some form of AI integration.

After initial research on AI/LLM-driven code feedback systems, this specific use case seemed to be relatively unexplored. There have been many experiments conducted concerning AI in the code review process [12] [13], however these do not align entirely with this project. The code review procedure focuses on adhering to best practices and ensuring efficiency, rather than offering constructive debugging criticism. For this reason, some investigation unique to this project would have to be carried out.

Further research on the topic led to the extremely useful resource *HuggingFace* [14], a central repository for a wide variety of AI models that allows users to easily compare and contrast their behaviours. The site also offers tools for both client- and server-side prompt processing, but this will be discussed in greater detail in the implementation chapter.

2.2.1 Prompt Engineering

Prompt Engineering can be summarised as the process of developing and optimising prompts to maximise a large language model's performance in a certain task. In this specific use case, the criteria would be consistent, concise, and accurate feedback for code. To achieve this, the LLM will be passed the same prompt structure every time, with the assignment specification and student's code being the only variable parts.

According to *promptingguide.ai* [15], the key components of an effective prompt are:

- Instruction - a specific task or instruction you want the model to perform
- Context - external information or additional context that can steer the model to better responses
- Input Data - the input or question that we are interested to find a response for
- Output Indicator - the type or format of the output.

Applying this to the context at hand:

- Instruction:
  - Explain the task of identifying errors and helping students to understand them
- Context:
  - Give an example of an error being found and advice being given
- Input Data:
  - The problem statement and the student's code
- Output Indicator:
  - Two sections, one for identified errors and one for pointers to help students fix them

With this prompt structure, the aforementioned goals of consistency, conciseness, and accuracy should be achieved. After multiple iterations, the following prompt proved the most consistent:

```
Given the following task description and a student's attempt at it, your task is to give
feedback on the student's solution. The feedback must not contain any code snippets and
should be split into two sections: One for errors of any type (syntax, runtime, logical) and
one for pointers on what the student should investigate to fix their problem. The errors
section should simply state the problem and not offer any further insight. The pointers
section should be concise (only one pointer per error found) and only tell the student
where to look. If there are no errors in the code, simply output "No problems found" and
skip the pointers for fixing mistakes section.

For further context, if the student code contains a logic error on line 39 the re-
sponse should look like:

Errors:
- [state the error]
Pointers:
- [explain the related logic that needs to be investigated]

Task description:

TASK GOES HERE


Student's Code Attempt:

CODE GOES HERE
```

Figure 1: *The engineered prompt template that will be used to give students feedback and guidance*

2.2.2 Model Comparison

At the time of writing, *HuggingFace* currently has 1, 274, 848 listed models with many different use cases. The type of model best suited for this task is one tailored for text generation. In order to find the best model for the feedback generation process, the output of various LLMs must be evaluated. This can be achieved by utilising the *Playground* tool built into *HuggingFace* [16]. Through this, it is possible to pass the same prompt to multiple models and compare their outputs. However, before starting the model comparisons, the prompt engineered above needed an example task and student submission. For this purpose, a simple calculator exercise and attempt were created and passed into the prompt. With the prompt ready, the next step was to pick models to compare.

As a starting point, filtering the text generation model directory by most downloads seemed reasonable. Once a suitable model with *Playground* compatibility was found (in this case MistralAI's `Mistral-7B-Instruct-v0.2`), the model comparison procedure simply consisted of passing the prompt into a different model and choosing the favourable side until there were no more models to test.

In order to determine the 'favourable' side, a few factors were taken into consideration:

- Adherence to the prompt criteria
- Consistency of responses
- Model size
- Compute times

These criteria ensured a solid and fair framework with which models could be compared.

After rigorous testing with different variations of both working and partially incomplete solutions to the calculator exercise with various models, the two standout models were the initial *MistralAI* model and *NousResearch*'s `Hermes-3-Llama-3.1-8B`. Initially, they appeared similar as both gave consistent responses and had similar compute times. However, `Hermes-3-Llama-3.1-8B` ultimately prevailed in its adherence to the prompt criteria as visible below:

`Mistral-7B-Instruct-v0.2`	`Hermes-3-Llama-3.1-8B`
Errors: - Incorrect calculation for addition and subtraction operations, they should be performed using '+' and '-' symbols respectively. Pointers: - For addition, investigate the line where the result is assigned, it should use the '+' symbol instead of subtraction. - For subtraction, similarly, investigate the line where the result is assigned, it should use the '-' symbol instead of subtraction.	Errors: - The subtraction operation is used for both addition and subtraction. - Division operation is used for both multiplication and division. Pointers: - Investigate the logic for handling addition and subtraction operations separately. - Investigate the logic for handling multiplication and division operations separately.

Figure 2: *A comparison of LLM responses to a user's submission where the '-' operator is used in both addition and subtraction, and the '/' operator is used in both multiplication and division*

As visible above, `Hermes-3-Llama-3.1-8B` catches both logic errors and gives much more concise responses in comparison to its competitor. Furthermore, the model from *NousResearch* also is smaller in size making it the clear choice of the two and the overall winner of the model comparisons.

3 Analysis/Requirements

This chapter will explore the various requirements of the project, along with the methods used to obtain them. The structure of the chapter will be to first outline user stories and scenarios, and then use those to formulate requirements.

3.1 User Scenarios

The following is a run-through of a typical assignment's life cycle. It will be told from different user perspectives in-order to encapsulate all the features that will be required.

User Scenario 1: Lecturer Creates an Assignment
Persona: Dr. Emily Carter, a university lecturer in Computer Science.

> Dr. Emily Carter logs into the *AutoAssign* platform with her faculty credentials. She navigates to the "Assignments" tab, where she fills out an assignment creation form with the title: "Optimised Sorting Algorithms". She writes a problem description explaining the task: implementing and analysing different sorting techniques.
>
> To guide students, she uploads a Python skeleton code template with function names and a set of predefined unit tests to automatically check correctness.
>
> Before publishing, she previews the assignment to ensure clarity. Satisfied, she assigns it to her class group and saves the changes, making the assignment available to her students.

User Scenario 2: Student Submits a Solution
Persona: Alex Johnson, a second-year Computer Science student.

> Alex logs into *AutoAssign* and sees that Dr. Carter has assigned a new task. Curious, he clicks on it and reads the problem statement. He downloads the provided Python template and begins coding.
>
> After implementing the required algorithms, he performs some local tests before uploading his solution. Once uploaded, AutoAssign provides instant feedback that Alex's solution has passed 9 tests but failed one (an edge case he forgot to account for).
>
> Slightly confused, Alex requests automated feedback from the AutoAssign assistant and has the error explained to him, as well as receiving some pointers as to what might help fix it.
>
> Alex revisits his code, refactors it to account for the edge case, and resubmits. This time, all tests pass. Feeling accomplished, he waits for the final grading after the deadline.

User Scenario 3: Lecturer Reviews and Provides Feedback
Persona: Dr. Emily Carter, a university lecturer in Computer Science.

> After the deadline, Dr. Carter accesses the assignment's submissions panel. The platform has already auto-graded everything based on her predefined tests, however she notices Alex's submission took much longer than the average student's to run.
>
> She clicks on Alex's submission and sees that although he optimised memory usage, his execution time could be improved. She makes a note of this and sends him his feedback with the comment: *"Great job reducing memory usage! Try exploring different pivot strategies for the quicksort implementation to further improve efficiency."*

User Scenario 4: Student Views Leaderboard and Improves Code
Persona: Alex Johnson, a second-year Computer Science student.

> A few days after submission, Alex logs into *AutoAssign* to check his results. His score is good, but his ranking on the efficiency leaderboard is lower than expected.
>
> He checks his E-Mail and sees Dr. Carter's feedback. Curious, he examines the best-ranked solutions (anonymised, unless students choose to reveal their names). He sees that the highest-ranked submission used a dual-pivot quicksort, which is more efficient for large datasets.
>
> Motivated, Alex modifies his code, implements the new technique, and tests it. Excited by his improvement, he makes a note to apply this optimisation in future assignments.

3.2 User Stories

Based on the above user scenarios, the following user stories provide an extracted set of features that encapsulate expectations from both lecturers and students alike.

For Lecturers/Course Owners:

- As a lecturer, I want to easily create coding assignments so that students have a structured and standardised problem to solve.
- As a lecturer, I want to upload unit tests so that the platform can automatically validate student submissions.
- As a lecturer, I want to review student submissions and see their performances so that I can more easily provide them with feedback.
- As a lecturer, I want to enable AI-generated feedback so that students receive timely and detailed explanations for errors.
- As a lecturer, I want to generate a leaderboard after the deadline so that students are encouraged to optimise their solutions.

For Students

- As a student, I want to view all my assignments in a structured format so that I can easily understand and complete them.
- As a student, I want to submit my code and receive instant feedback so that I can correct errors before the deadline.
- As a student, I want to resubmit my assignment multiple times before the deadline so that I can improve my solution based on feedback.
- As a student, I want to view personalised feedback so that I can understand what I did wrong and how to improve.

- As a student, I want to compare my solution with others on a leaderboard so that I can learn from better implementations and improve my coding efficiency.
- As a student, I want to have the option to remain anonymous on the leaderboard so that I can participate without feeling pressured.

3.3 Requirements

After establishing the requirements, the next step is to prioritise them. This is to ensure that key functionality is delivered no matter what, and *"nice to haves"* do not take precedence during development. The *MoSCoW* method is a labelling system by which this can be achieved, splitting requirements into four key categories:

- **Must Have**: Requirements that are vital to the project's success
- **Should Have**: Requirements that are important, but the success of the project does not depend on their completion.
- **Could Have**: Requirements that are desirable, but their absence from the final product would be acceptable
- **Won't Have**: Requirements that will not be delivered on, but are stated nonetheless to make the scope of the project known.

System:

- **Must Have**
  - Automatic unit test execution on submissions
  - Ranking of student submissions based on execution time and memory efficiency
  - Account creation for all user groups
- **Should Have**
  - AI-Assisted automated targeted feedback and assistance provision upon request
  - Manual user-initiated unit test re-execution
  - Ability to let any user change their display name
- **Could Have**
  - Badges, XP points, and reward systems for achievements
  - Notify students when assignments are published, graded, or when feedback is available
- **Won't Have**
  - Automated detection and flagging of suspiciously similar code submissions
  - Mobile friendly interface
  - Real-time messaging for student-teacher help and communication
  - Direct coding in the platform rather than using external editors

For Lecturers/Course Owners:

- **Must Have**
  - Ability to create coding assignments with a problem description, title, unit tests, and due date
  - Ability to review submissions
  - Ability to create groups
  - Ability to assign assignments to groups
  - Ability to edit a created assignment
  - Ability to edit a created group
  - Ability to delete created assignments
  - Ability to delete created groups

- **Should Have**
  - Option to provide a skeleton code starting template on assignments
- **Could Have**
  - Ability to provide manual comments on student work
  - Ability to view a student's number of submission attempts
  - Option to duplicate existing assignments for reuse in future courses
  - Ability to define custom ranking criteria for the leaderboard
- **Won't Have**
  - Ability to generate downloadable reports on student performance and progress
  - Ability to define penalties for late submissions

For Students:

- **Must Have**
  - Ability to view all active and completed assignments in a structured format
  - Ability to upload solutions and receive automated test results
  - Ability to refine and resubmit code before the due date
  - Ability to view execution time, memory usage, and rankings compared to peers
  - Choose whether to display their name or remain anonymous on the ranking board
- **Should Have**
  - Ability to see other student's solutions past the assignment due date for learning purposes
- **Could Have**
  - Ability to access previous attempts to track progress and improvements
- **Won't Have**
  - Ability for students to give and receive feedback to and from peers

4 | Design

This chapter will go into the details of the process that took *AutoAssign* from a rough idea to a solid blueprint that could be implemented. All aspects of the system will be discussed, from the architecture of the back-end to the layout of the front-end. Where relevant, design choices will be linked to requirements mentioned in the previous chapter.

4.1 System Architecture

Before making a start on the user interface, it is important to consider the underlying architecture that will facilitate it.

4.1.1 Database and Permissions

As a first step, considering the **Must Haves** from the requirements chapter serves as a solid guideline. Through items such as: *"lecturers must have the ability to create coding assignments"*, *"lecturers must have the ability to assign assignments to groups"* and *"students must have the ability to upload solutions"* we can already establish the structure of the core database.

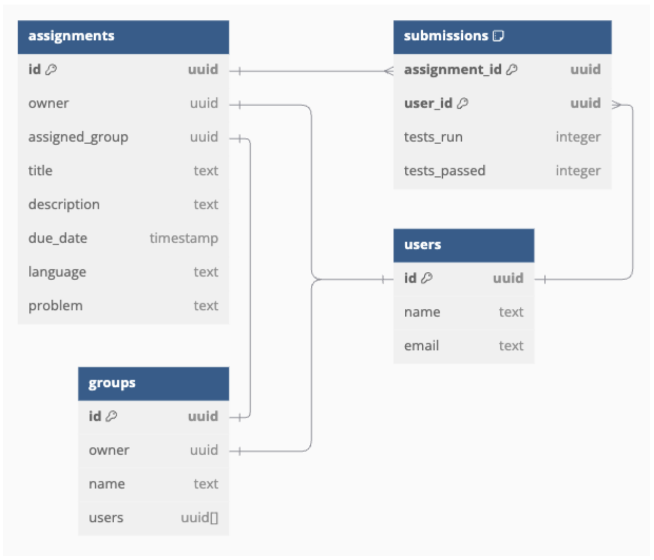


Figure 3: *An initial database mock-up, demonstrating both the tables and their field relations*

The tables in Figure 3 not only meet several of the outlined requirements, but also lay down a few limitations. One such example would be the one-to-one relationship between an assignment's `assigned_group` field and a group's `id` field. This limits an assignment such that it can only be assigned to a single class/group at a time. While it would be technically possible to implement logic for multi-group assignments, this would cause problems when keeping submission and performance data separate from one another. Ultimately, given the time constraints of the project, such functionality is not a priority.

Another key consideration to take into account is permissions. Obviously, students should not be able to download or view certain content until after the deadline of the assignment has passed. There are many ways to implement such checks, such as comparing the value of the currently logged in user's `id` and the `user_id` of a submission in the front-end. However, such solutions often leave vulnerabilities exposed. Instead it is much better to implement Row-Level Security (RLS) directly into the database, shutting down any unauthorised activity directly at the source of the data. Precautions such as these are not only important to prevent cheating, but also ensure the privacy of course material that should not be distributed without permission.

4.1.2 Code Execution

AutoAssign's primary functionality is to automatically assess coding assignments. In order to achieve this, a code execution pipeline must be established that will handle student's submissions.

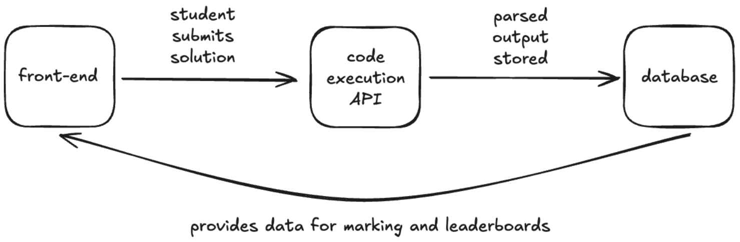
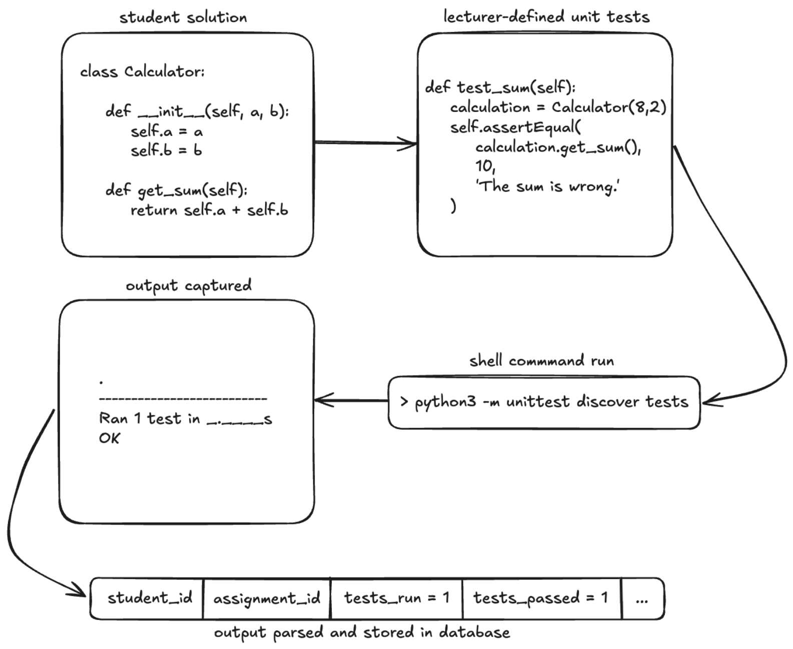


Figure 4: *A rough depiction of the code execution workflow, from submission to data storage*

While online code execution platforms such as *Judge0* [17] are available to abstract this code execution process, a conscious design decision was made to minimise the dependencies of the project. Reducing dependencies enhances system stability and facilitates the efficient diagnosis of issues within the application. These characteristics are essential for a system designed for continuous operation throughout the year. Therefore, the plan for automated code execution is instead to create TypeScript API endpoints that directly interface with the host system.


student solution
lecturer-defined unit tests
output captured
shell command run
output parsed and stored in database

```
class Calculator:

    def __init__(self, a, b):
        self.a = a
        self.b = b

    def get_sum(self):
        return self.a + self.b
```

```
def test_sum(self):
    calculation = Calculator(8,2)
    self.assertEqual(
        calculation.get_sum(),
        10,
        'The sum is wrong.'
    )
```

```
> python3 -m unittest discover tests
```

```
.
--------------------------
Ran 1 test in _._____s
OK
```

student_id	assignment_id	tests_run = 1	tests_passed = 1	...

Figure 5: *A student solution passing through the pipeline, from submission to the data being updated*

4.1.3 AI-Assisted Feedback

Another listed requirement of the system is *"AI-Assisted targeted automated feedback and assistance provision"*. In order to not introduce unnecessary complexity to the project, a more abstracted approach to this system is favourable. Since *HuggingFace* was used during initial research conducted in the background chapter, their serverless inference API [18] seemed a logical extension of that.

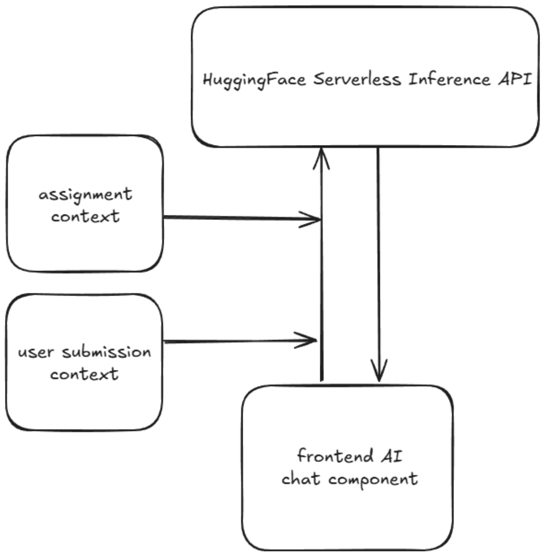

HuggingFace Serverless Inference API
assignment context
user submission context
frontend AI chat component

Figure 6: *A diagram of the interaction between AutoAssign and HuggingFace, demonstrating the integration of assigment and user submission contexts into the outbound requests*

The aforementioned high level of abstraction is clearly demonstrated in Figure 6. The processing and return of the query is out of scope for this project and is therefore left to the API to handle. Worth noting, though, is the inclusion of the assignment and submission contexts in the user's prompts. This is vital as without these additions, the AI would not be able to provide anything more than generic advice which would clearly violate the requirement of *"targeted"* feedback and assistance.

4.2 User Interface

With the foundations established, the next step in bringing the project to life was to create a site structure and design language. This process would prove to be instrumental in the project's success as it acted as a reference to compare against at any point in the development process.

4.2.1 Site Structure

As before, a good place to start in this process is to turn to the requirements. Items such as *"students must have the ability to view all active and completed assignments in a structured format"* and *"lecturers must have the ability to create groups of students"* provide an elementary understanding of what pages will be required:

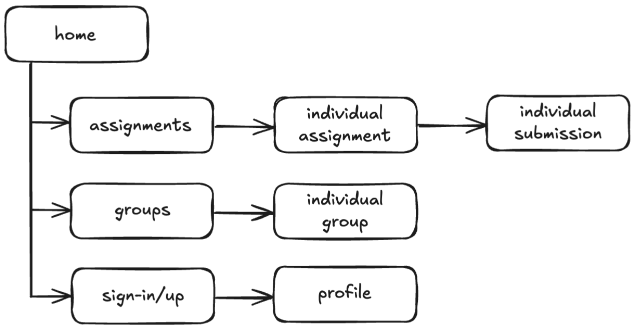


Figure 7: *A preliminary sitemap, structured by page type*

This simple structure depicted in Figure 7 ensures minimal confusion in both students and lecturers alike, which is vital for the user adoption process. The linear nature of the links between the pages is intuitive, and prevents users from getting lost when navigating several interconnected pages.

4.2.2 Page Layout

Continuing with the trend of simplicity, the design and layout of the interface should also incorporate that philosophy. To bring such a design to life, the *shadcn/ui* Figma component library was used [19]. This is a port of the *shadcn* UI system [20] that will be used later on in the implementation.

Most websites incorporate a navigation bar positioned at the top of the screen, optimising the horizontal space available for content below. This layout is particularly advantageous for businesses such as e-commerce platforms, where multiple items are displayed side by side, allowing for efficient utilisation of the available space. However, this project focuses on code, which is inherently structured as a vertical sequence of lines of text.

A **Must Have** requirement of the site is to provide lecturers with the *"ability to review submissions"*. In order to facilitate this process as efficiently as possible, it is essential to maximise the effective use of vertical space on the page.

Depicted in Figure 8 is a sidebar design. This approach provides the same functionality as the aforementioned navigational bar, but it does not subtract from the vertical real estate of the page.

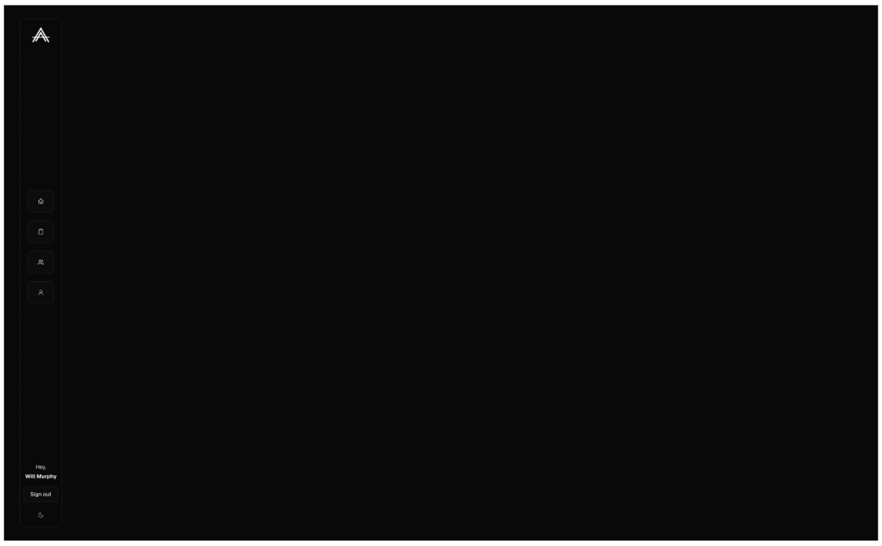


Figure 8: *A navigational sidebar designed in Figma using shadcn/ui components*

4.2.3 Landing Page

AutoAssign is a marketable product, and its landing page should therefore be designed in-line with that. In terms of requirements, the page should provide both a visual and textual description of the site's purpose, as well as what sets it apart from competitors. Failure to do so could result in poor first impressions, making potential users infinitely more difficult to persuade to sign up and onboard.

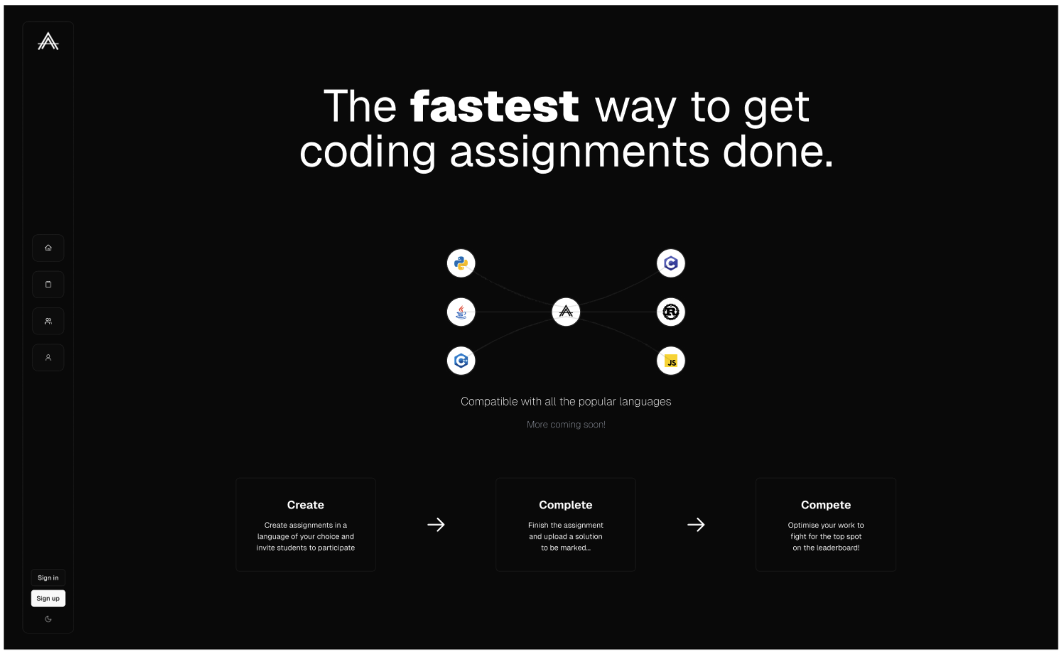

The **fastest** way to get
coding assignments done.
Compatible with all the popular languages
More coming soon!
Create
Create assignments in a language of your choice and invite students to participate
Complete
Finish the assignment and upload a solution to be marked...
Compete
Optimise your work to fight for the top spot on the leaderboard!
Sign in
Sign up

Figure 9: *A landing page designed in Figma using shadcn/ui and MagicUI components*

The bold and centred headline visible in Figure 9 clearly describes the site's purpose, communicating to the user what exactly it is that *AutoAssign* does. The diagram beneath demonstrates the application's compatibility with well-known languages, letting the user know whether the platform is applicable to their use case or not. Finally, the three boxes at the bottom clearly represent the workflow and goals of the site, emphasising the streamlining of the assignment process. The text in the boxes also describes the unique selling point, the competition aspect.

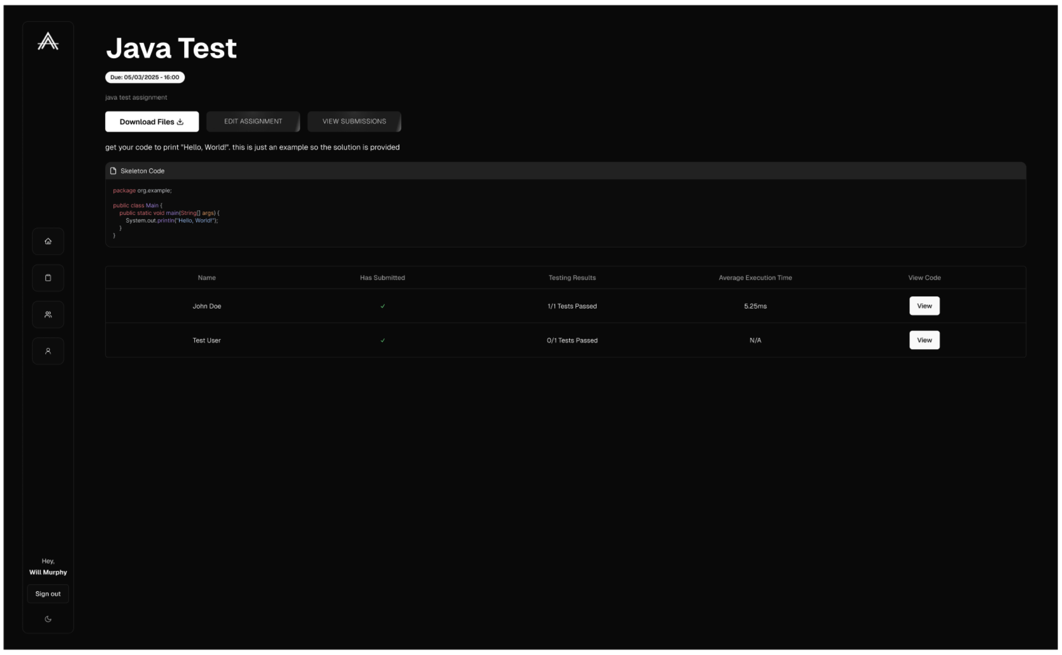
Together, these three components form a minimal yet effective landing page that does not overwhelm the user but also pitches the product to them.

4.2.4 Assignment Page

Arguably one of the most important pages on the site is the assignment page. When consulting the requirements of the project, the following becomes clear:

- There must be space for a title, problem description and due date
- There must be a way for lecturers to edit the assignment
- There must be space for skeleton code
- There must be a leaderboard
- There must be a way for students to submit their solution
- There must be a way for a lecturer to view a student's submission

As this is quite a few requirements, it is important to carefully consider the design so as not to confuse users



Java Test

Due: 05/03/2025 - 16:00

java test assignment

get your code to print "Hello, World!". this is just an example so the solution is provided

```
package org.example;

public class Main {
  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }
}
```

Name	Has Submitted	Testing Results	Average Execution Time	View Code
John Doe	✓	1/1 Tests Passed	5.25ms	View
Test User	✓	0/1 Tests Passed	N/A	View

Figure 10: *An assignment page designed in Figma using shadcn/ui and MagicUI components*

Pictured in Figure 10 is the final design that was settled upon. The information on the page is ordered vertically from most important to least important for clarity. The title, due date and description are at the very top, indicating the most important aspects of any assignment: what needs to be done and when it needs to be done by.

Underneath that section are an array of buttons. To students, only the download one will be visible but the all will be accessible to the assignment creator. These facilitate the ability to amend the assignment and moderate student submissions before and after the deadline of the assignment.

Optionally, the assignment creator will also be able to provide a skeleton code file and an accompanying additional description. This is to accommodate for more standardised solutions from students, which makes the marking process easier. The code will be displayed in its own component, with the appropriate syntax highlighting for maximum readability.

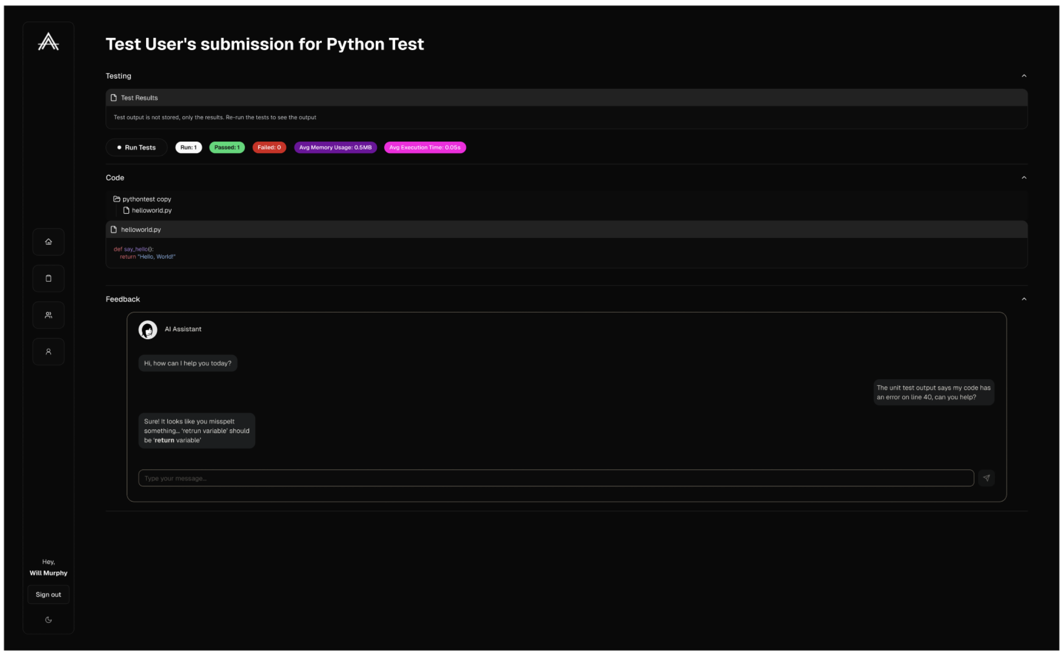
Finally, the leaderboard is shown at the bottom. This will only become visible to students once the assignment deadline has passed. This ensures, to the best of the website's ability, that students do not cheat by copying each other's solutions or get in contact with each other based on their solution's performance.

4.2.5 Submission Page

The last important page design to cover is the submission page. This will implement many must have requirements from both student and lecturer perspectives:

- Lecturers must be able to review submissions
- Students must be able to upload solutions and receive automated test results
- Students must be able to view the execution time and memory usage

As discussed above, students will not have access to their peer's performance statistics until after the assignment deadline has passed. However, they must still be able to see their own so that they can iterate on their solution effectively and improve their submission.


Test User's submission for Python Test

Testing

Test Results

Test output is not stored, only the results. Re-run the tests to see the output

```
def say_hello():
  return "Hello, World!"
```

Hi, how can I help you today?

The unit test output says my code has an error on line 40, can you help?

Sure! It looks like you misspelt something... 'retrun variable' should be '**return** variable'

Figure 11: *A submission page designed in Figma with shadcn/ui and MagicUI components*

Visible in Figure 11 is the desired design for the submission page. The title will contain the name of the user and the assignment for which the submission was made, and the content will consist of testing, code, and feedback.

The testing section will have a dedicated component for the returned terminal output from the code execution API detailed in section 4.1.2, a button to make the execution call, and information on unit test output and code performance. The labels provide a clear indication of the status of the submission, as manually understanding it from the terminal output can sometimes be difficult, especially when many errors are present.

The code section will contain a file tree of the student's submission as well as a file viewer for the selected file. This will not have any editing capabilities but will simply serve as a visual inspection tool for both the student and lecturer.

Lastly, the feedback section will contain a chat with an AI assistant that can assist the student in the debugging process. As mentioned in the background chapter, this model's purpose is not to provide the student with the solution. But simply to correct basic errors and guide them toward success.

4.3 Technologies

Lastly, before beginning the implementation, it is important to consider the currently available technologies and opt for those that best facilitate the requirements of the project.

4.3.1 Supabase

As an application that could be deployed as a real world product, it is important to consider the requirements that this demands of the database solution. A traditional local database requires large amounts of manual configuration and bespoke API development to facilitate a connection with the front-end. However, a fully managed PostgreSQL database solution such as *Supabase* [21] aims to abstract much of this complexity and offer a more streamlined and scalable alternative.

Features such as built-in functionality for easy Row-Level Security (RLS) configuration perfectly meet *AutoAssign*'s requirements for student access restriction and copyrighted course material protection. Furthermore, out-of-the-box support for user authentication with services such as Google, Apple and GitHub make the platform much more accessible to potential customers.

4.3.2 Next.js

When choosing a web development framework, the key considerations are always performance and community support. Both of these aspects are important to ensure both a high-quality final product and a painless development experience. *Next.js* [22] achieves this well with its server-side rendering capabilities and long open-source history, making it an ideal choice for this project. Being react based, it also opens the door to countless large component libraries and tutorials that can be leveraged to keep code clean and effective while also speeding up development.

4.3.3 MagicUI and shadcn

On the topic of component libraries, these two stood out amongst alternatives as they aligned well with the design goals of the project. *shadcn* [20] is a minimal user interface library with clean components and straightforward customisation. These would be used throughout the site to tailor a stress-free and intuitive experience for the user.

On the other hand, *MagicUI* [23] offers elements of animation and interactivity. Examples of components include animated text and buttons, custom cursors, and backgrounds. These will be valuable additions to the interface to pique user interest and not bore them with a lifeless monochrome experience.

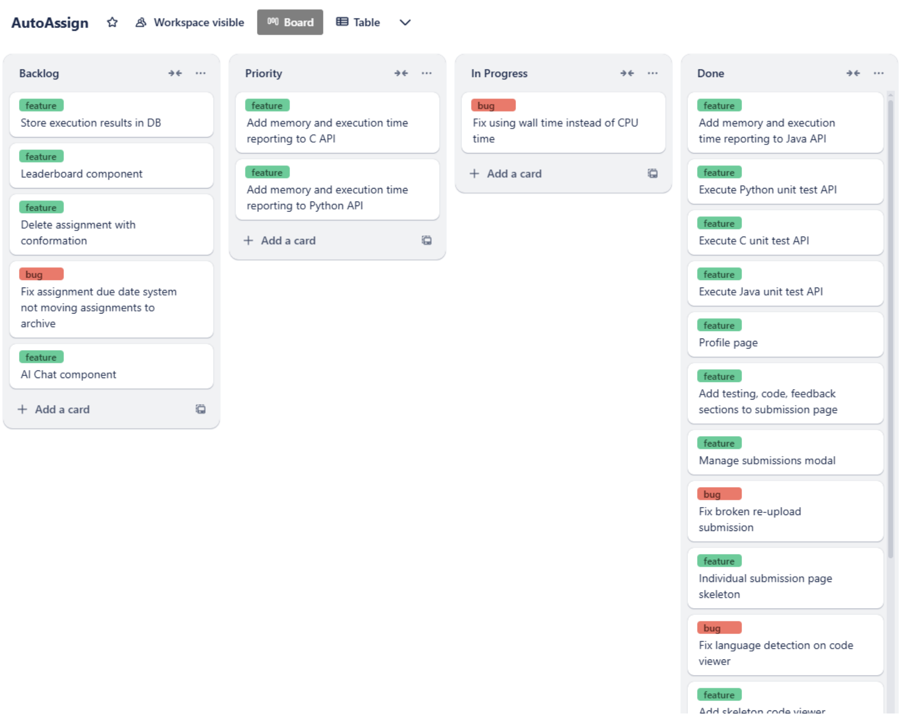
5 Implementation

The following will be an account of the development process, from the highest level down to individual features of note. The chapter will also cover the software engineering practices utilised, the challenges faced, and how they were overcome.

5.1 Software Engineering Process

Before any code was written, it was important to first initialise a Git repository, ensuring that throughout development all code is both backed up and version-controlled. This ensures that all changes are tracked, making it easier to identify and revert to previous versions if needed. Even for an individual project, version control provides a safety net against accidental mistakes and helps maintain a structured development process.

Furthermore, a *Trello* board was established with sections for backlog, priority, in-progress tasks, and completed work to systematically track development progress. This approach ensured that features and bug fixes were effectively managed and not overlooked amid other academic responsibilities. It also gave a good overall indication of progress, as the backlog shrunk and the list of completed items grew at different rates over time.



Backlog
- feature — Store execution results in DB
- feature — Leaderboard component
- feature — Delete assignment with conformation
- bug — Fix assignment due date system not moving assignments to archive
- feature — AI Chat component
- + Add a card

Priority
- feature — Add memory and execution time reporting to C API
- feature — Add memory and execution time reporting to Python API
- + Add a card

In Progress
- bug — Fix using wall time instead of CPU time
- + Add a card

Done
- feature — Add memory and execution time reporting to Java API
- feature — Execute Python unit test API
- feature — Execute C unit test API
- feature — Execute Java unit test API
- feature — Profile page
- feature — Add testing, code, feedback sections to submission page
- feature — Manage submissions modal
- bug — Fix broken re-upload submission
- feature — Individual submission page skeleton
- bug — Fix language detection on code viewer
- feature — Add skeleton code viewer

Figure 12: *A snapshot of the project's Trello board during development, featuring various columns for different stages of issue tracking*

In addition to the feature and bug labels visible in Figure 12, there were also labels for setup- and testing-related tasks at different stages during the project.

5.2 Assignments

As has been consistently iterated in this paper, the motivation of *AutoAssign* is to streamline and simplify the coding assignment process. It is therefore vital that the workflow of the primary feature of the site is as straightforward and painless as possible. The following subsections will discuss the creation and viewing processes, highlighting adjustments made in the interest of accommodating the user.

5.2.1 Creation Modal

As mentioned in the design chapter, the number of pages on the site is to be minimised so as to avoid complex navigation that could leave users lost. To address this, the assignment creation process was implemented as a multi-step modal pop-up that is accessible on the assignments listing page.

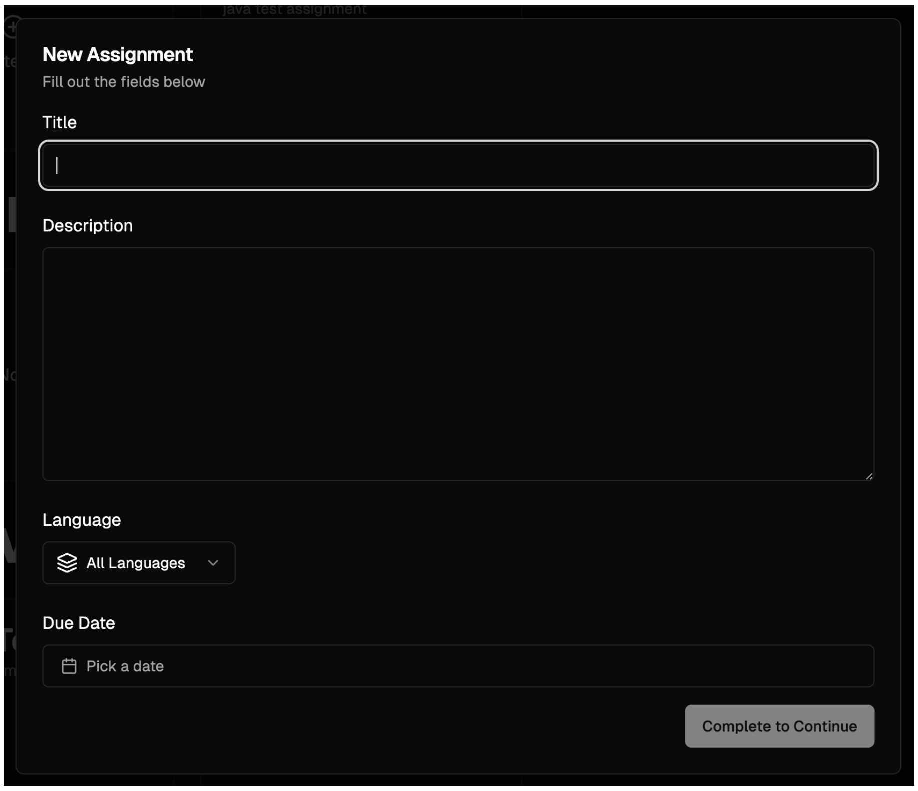


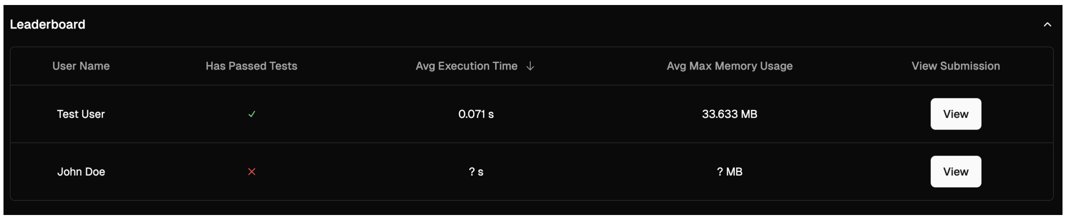
Figure 13: *The first page of the assignment creation modal, prompting the user for assignment details such as title, description, language and due date*

Demonstrated in Figure 13 is the simplicity of the modal, only requiring the user to fill in the most basic elements to set up their assignment on the platform. Real-time validation was implemented into the fields of the form to ensure that it is not possible to progress with incomplete data.

One issue that presented itself during testing was the deletion of input when accidentally closing the modal by clicking outside of it. To address this, the code was modified such that the user's inputs are stored in state, meaning that in the event of accidental navigation away from the modal, the data will be saved (subject to page refresh).

5.2.2 Viewing Assignments

Equally as important as the creation process is the ability to view assignments in a structured and clear format. One implementation detail of the page worth highlighting is the leaderboard.



Leaderboard

User Name	Has Passed Tests	Avg Execution Time ↓	Avg Max Memory Usage	View Submission
Test User	✓	0.071 s	33.633 MB	View
John Doe	×	? s	? MB	View

Figure 14: *A leaderboard for an assignment, with one valid and one invalid solution*

The competitive element of *AutoAssign* is entirely optional. As such, the leaderboard was implemented to be collapsible. It features both ascending and descending customisable sorting for all columns (except view submission), allowing both students and course coordinators to single out submissions they wish to take a closer look at.

Additionally, due to the optional nature of the competitive aspect mentioned above, the asynchronous data retrieval for the table was modified to determine the anonymity setting of each of the fetched users. This is then used to either show the student's user name, or replace the name with *"Anonymous"*. This is a favourable implementation over giving students the option to retract their submissions from the board, as a greater number of submissions to compete against ensures competitive integrity and pushes students to devise more optimised solutions to their assignments.

One further optimisation that was made to the leaderboard component is the utilisation of cache to store fetched leaderboard data. Due to the collapsible nature of the table, react re-renders and therefore re-fetches the data every time the leaderboard is collapsed and re-opened. To prevent API spam, a cache with an expiry time is created that is only re-fetched if it expires or the URL changes (*i.e.* the user has navigated to a different assignment).

5.3 Submissions

In order to maintain the simplicity of *AutoAssign*'s workflow, student submissions are nothing more than `.zip` files containing the root folder of the source code. Although this eliminated complexities from the submission upload process, it presented new challenges when it came to viewing them online.

Part of the platform's goal as an all-in-one assignment handling platform is to facilitate not only the marking process, but also feedback. In order to do this, the owner of the assignments must be able to inspect the source code from within the site instead of downloading every submission for themselves.

To address this concern, *JSZip* [24] (a JavaScript library for creating, reading and editing `.zip` files) was used. After using the library to load the data contained within the `.zip` file, it is traversed to build a tree that represents the directory. This process can be summarised with the following pseudocode:

```
FUNCTION constructFileTree(zipFile):

   SET fileTree TO empty list

   FOR EACH (path, entry) IN zipFile:

      IF path = HIDDEN FILE TYPE:

      IF entry IS A FILE:
         SET parts TO path SPLIT BY "/"
         SET current TO fileTree
         FOR EACH part IN parts EXCEPT LAST:
            FIND OR CREATE folder IN current WITH name = part
            SET current TO folder.children
         ADD FILE { id: path, name: LAST part, isSelectable: true } TO current

   RETURN fileTree
```

Figure 15: *A simplified pseudocode representation of the file tree construction logic, showing how* `.zip` *submission files are parsed*

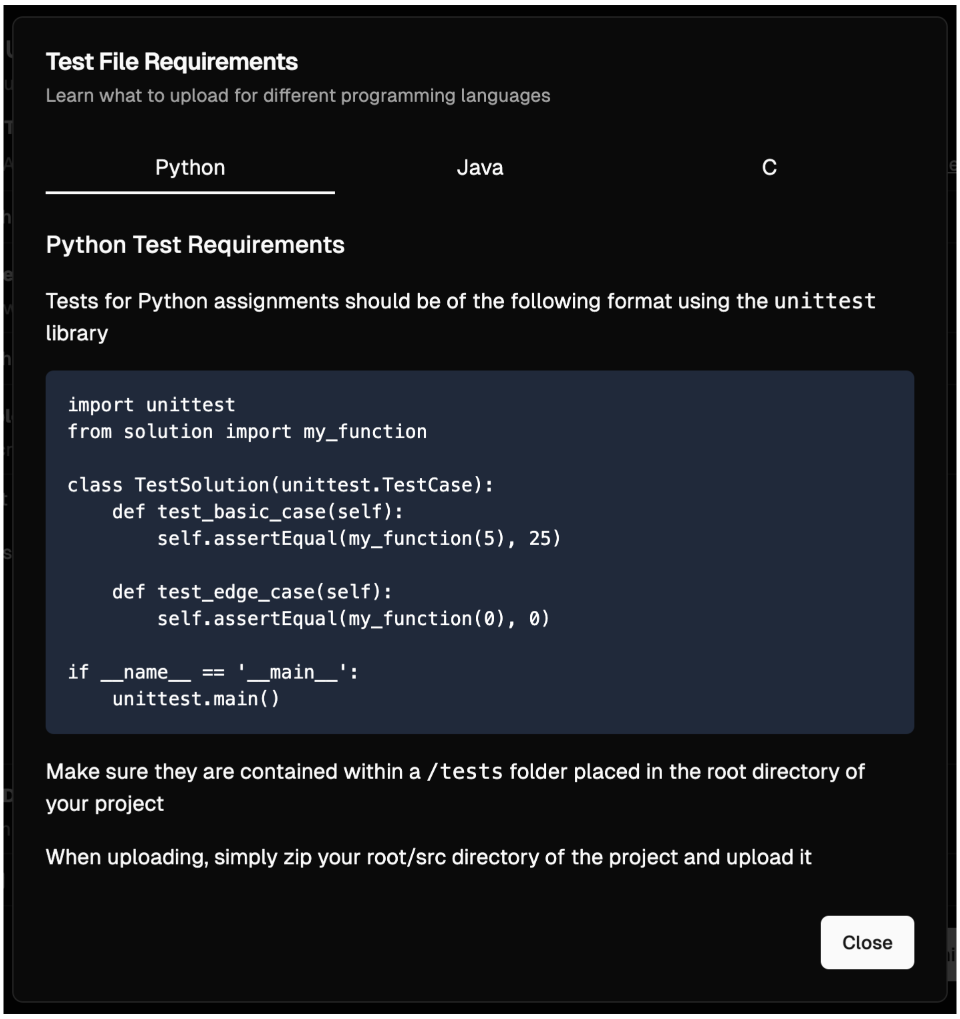
To explain what is happening, the relative path of each entry found in the `.zip` file is split into parts, and the function iterates through these parts to build a nested folder structure. This process ensures that files and folders are properly nested in a hierarchical format, allowing the UI to display the submission's structure accurately. This data structure is then used in combination with the *"File Tree"* component from *MagicUI* to act as the file explorer for the submission.

Upon selecting a file in the explorer, the `id` (which is set to the path of the file) is used to retrieve the contents of the file. The contents is then displayed in a code window below with syntax highlighting based on the assignment's specified language.

5.4 Unit Test Execution API

The automation of unit test execution was by far the most technically challenging implementation challenge in the project. Coding projects are very intricate in their nature, with factors such as the position and names of files being of great importance. Not only is it a given that each student will interpret an assignment in a slightly different way, subsequently affecting the structure of their solution. But it is also important to consider that each assignment creator will set up their unit tests differently too. It is therefore vital that the unit test execution logic is well equipped to deal with these inconsistencies.

The first step to designing this complex architecture is to establish a basic set of guidelines that *AutoAssign* will assume of users who create assignments. Given the limited time frame of the project, it is not feasible to take into account all possible unit testing frameworks and configurations. This means that the rules for creating assignments will have to be relatively strict.



Python Test Requirements

Tests for Python assignments should be of the following format using the `unittest` library

```
import unittest
from solution import my_function

class TestSolution(unittest.TestCase):
    def test_basic_case(self):
        self.assertEqual(my_function(5), 25)

    def test_edge_case(self):
        self.assertEqual(my_function(0), 0)

if __name__ == '__main__':
    unittest.main()
```

Make sure they are contained within a `/tests` folder placed in the root directory of your project

When uploading, simply zip your root/src directory of the project and upload it

Figure 16: *A screenshot of AutoAssign's Python assignment unit testing guidelines, specifying the requirements for compatibility with the platform*

Figure 16 is an example of the aforementioned guidelines. They represent a precise balance of freedom for the user vs. limitations of the platform. Once these had been decided, the implementation of the execution API could begin with the structure depicted in Figure 17.

While Figure 17 describes a more general overview of the API, each supported language required its own modifications to ensure compatibility with their unit testing frameworks. Due to the intricate nature of the API and the limited time that had been allocated to complete it, this led to the decision to drop language support down to 3: C, Java and Python. This meant that valuable project work hours were freed up, while the platform would still support the vast majority of coding courses and assignments.

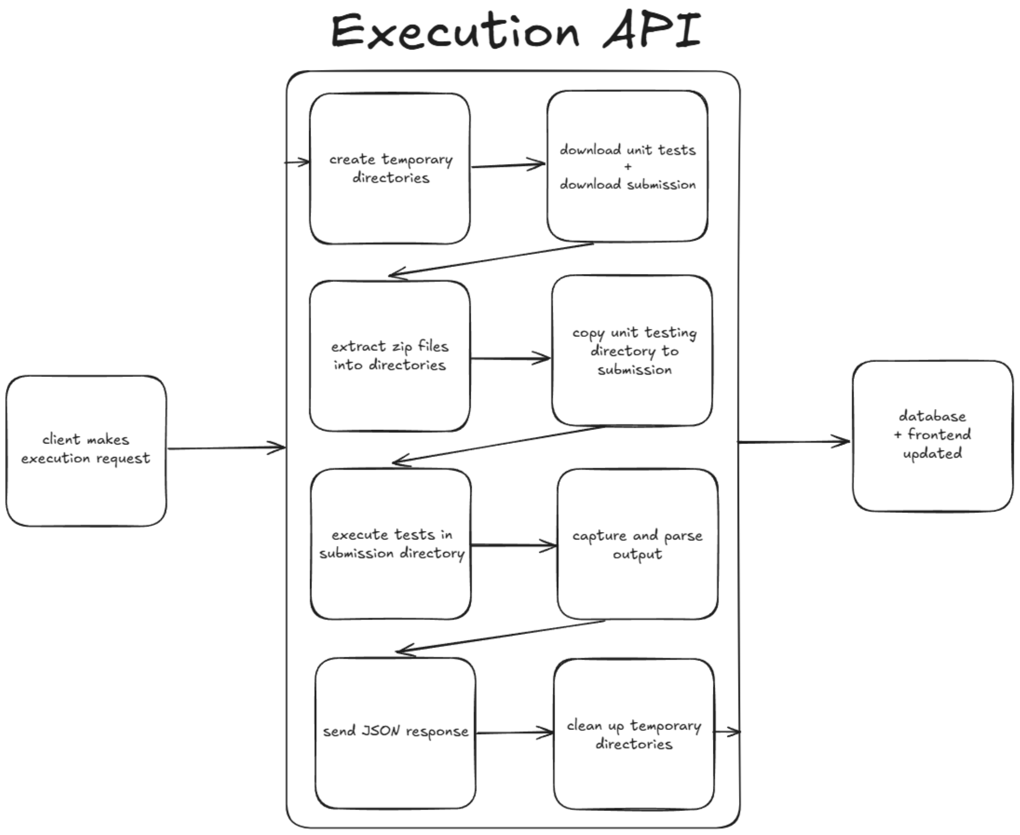

Execution API
client makes execution request
create temporary directories
download unit tests + download submission
extract zip files into directories
copy unit testing directory to submission
execute tests in submission directory
capture and parse output
send JSON response
clean up temporary directories
database + frontend updated

Figure 17: *A diagram of AutoAssign's automatic test execution API, demonstrating the flow of logic from the client request through to the end result of the data being updated*

The implementation of the submission page that sends the request was intentionally coupled as loosely as possible with the API. An example of this is the dynamic formation of the endpoint to which the execution request is sent. Instead of using a `switch` or series of `if` statements to decide between a set of hard-coded endpoints, the assignment language retrieved from the database is directly embedded in a template literal:

```
const response = await fetch(‘/api/execute-${assignmentLanguage}‘)
```

Figure 18: *A line of code demonstrating a dynamic execution API call*

This was a conscious decision as it promotes extensibility and means that it is much easier to add support for new languages down the line. Instead of having to modify the source code of the page, a new endpoint can be created with the name of the language, and the pre-existing logic will handle the rest.

5.5 AI-Assisted Feedback

As mentioned in the design chapter, the *HuggingFace* serverless inference API was the tool of choice to integrate the AI feedback chat into the site. During the implementation of the feature, many workflows were tested to determine the ideal approach that would appear the most constructive to the student. It was important to remember the goal of offering pointers and a helping hand, rather than outright providing the entire solution as many other models do.

Ultimately, the final implementation of the feature resulted in the following process for the user:

1. Open the AI chat drop-down section
2. Select a file in the file explorer for which assistance is required
3. Receive an initial rundown of issues if there are any and possible steps toward fixing them
4. *(Optional)* Use the chat box to continue the conversation

The purpose of step 3 coming before the chat box becomes available is to promote critical thinking from the student's end. For example, if a student were stuck and opened the drop-down to find a blank AI chat, it may be their first instinct to simply instruct the AI to solve the problem for them. However, with this approach the AI may say something first that confirms a suspicion or sparks an idea within the student, prompting them to return to their IDE and solve the problem themselves.

In order to achieve this workflow, a react `useEffect` hook tied to the updates of a currently selected file sends a prompt to the model. This prompt is the template designed as a result of testing and research in the background chapter, with the currently selected file and problem statement embedded within. From there, the chat UI component triggers an asynchronous send message function every time the user submits a prompt, displaying the responses as formatted incoming message bubbles on the left-side of the chat.

Another feature that is quite common when communicating with AI chatbots is to refresh the chat, which clears the session context. This is an efficient mechanism to reset the model from tangents, hallucinations, and confusion. In order to implement this functionality, the entire chat feature and its context were kept contained in a separate component which was then placed in the drop-down on the submission page. This exploits the fact that react only loads and renders components and their state when they are visible, inadvertently creating an intuitive refresh action of simply collapsing and re-expanding the drop-down containing the chat.

5.6 Row-Level Security in the Database

As previously discussed in this paper, a platform such as *AutoAssign* presents several security challenges. Improper configuration could allow students to access solutions prior to submission deadlines, enable lecturers to retrieve confidential course materials from other classes, and permit students to manipulate one another's submissions. Therefore, it is imperative to implement appropriate security measures. In this case, many safeguards were enforced using Row-Level Security at the database level.

This approach provides a more robust and reliable security mechanism compared to enforcing access controls at the web application level. By implementing security directly within the database, it is ensured that access restrictions are consistently applied, regardless of how data is queried or accessed. This approach mitigates the risks associated with potential oversights and vulnerabilities in the web application, such as insufficient authorisation checks or by-passable client-side restrictions. Furthermore, Row-Level Security simplifies security management by centralising access control policies, reducing the likelihood of misconfiguration.

```
ALTER POLICY "Enable delete for users based on user_id"
ON "public"."assignments"
TO public
USING (
   ((SELECT auth.uid() AS uid) = user_id)
);
```

Figure 19: *The Row-Level Security for the assignments table, checking the assignment creator ID against the authorised user's ID*

Figure 19 illustrates one of the many Row-Level Security policies deployed throughout the project. This particular policy ensures that only the creator of an assignment has permission to delete it, preventing unauthorised modifications by other users.

5.7 Maintainability

As a product with a potential future, it is important that the codebase remain in a maintainable state.

5.7.1 Code Comments

Other than exercising best practices of react development throughout the project, it is also vital to leave descriptive comments. This ensures that upon returning to the project, or adding new members to the team, precious time does not have to be spent re-learning the function of code components. For this reason, additional time was spent documenting the key functionalities and particularly complex details of the implementation in the code.

5.7.2 Project Structure

Another vital aspect of maintainability is code organisation. A well-structured project makes it easier for developers to both navigate and understand the different components. This includes using a logical folder and file structure that clearly separates concerns, such as grouping components, utilities, and pages in ways that make sense for the project's scale.

This was incorporated into *AutoAssign*'s implementation in many ways, with one example being the `ui/` sub-folder within the `components/` folder to further differentiate the foundational building blocks (*e.g.* button, label, select) from the bespoke components (*e.g.* AI chat, create assignment modal).

6 Evaluation

6.1 Unit Testing

Before conducting any testing with real world users, it must first be ensured that the system operates reliably and correctly.

6.1.1 Design

To achieve this, the JavaScript testing framework *Jest* was used to implement unit tests and monitor code coverage throughout the project. The questions this evaluation method seeks to answer are:

- Do components behave as expected under normal and edge case scenarios?
- Are API responses consistent and accurate across different requests?
- What percentage of the codebase is covered by unit tests?

6.1.2 Results

File	**% Stmts**
All files	81.8
app/page.tsx	100
app/(auth-pages)/forgot-password/page.tsx	100
app/(auth-pages)/sign-in/page.tsx	100
app/(auth-pages)/sign-up/page.tsx	100
app/assignment/[id]/page.tsx	61.47
app/assignments/page.tsx	80.95
app/group/[id]/page.tsx	88.57
app/groups/page.tsx	100
app/home/page.tsx	100
app/reset-password/page.tsx	100
components/account-form.tsx	100
components/workflow.tsx	100
components/ui	86.42
components/ui/accordion.tsx	100
components/ui/animated-subscribe-button.tsx	93.33
components/ui/button.tsx	52.94
components/ui/card.tsx	52.63
components/ui/dialog.tsx	100
components/ui/input.tsx	100
components/ui/shimmer-button.tsx	100
...	...

Figure 20: *An extract of the code coverage report generated with* `npx jest --coverage`. *Highlighted in yellow are files that have low (below 80%) code coverage.*

Unit testing an application as dynamic as *AutoAssign* proved to be more of a challenge than initially anticipated. Pages with many different states and edge cases made for extremely complex mocking setups that attempted to mimic all possible scenarios.

A good example of this is the assignment page, listed as `app/assignment/[id]/page.tsx` in Figure 20. It is a clear outlier amongst other similar pages such as assignments, groups, and home with only 61.47% unit test code coverage. This is a direct result of the many stateful components that make up the page, each with their own logic based on the user's status as assignment owner and whether the assignment has passed it's due date or not.

A significant portion of project time was dedicated to the development of the unit tests. With a total of 44 passing tests, 0 failing ones, and an overall 81.56% code coverage report, it is not unreasonable to state that they were implemented to an acceptable level.

Note: *In order to execute the unit tests, the* `"jsx"` *value in* `tsconfig.json` *must be changed fro*m `"preserve"` to `"react-jsx"`.

6.2 User Evaluation

As *AutoAssign*'s hypothetical end goal is to be a marketable product, it is important to assess user satisfaction and address concerns at an early stage. After the first version of the product was finished, a series of experiments were designed to attempt to obtain valuable insights into user behaviour on the site. The entire evaluation process was conducted whilst strictly adhering to the University of Glasgow School of Computing Science's ethics checklist.

The overall aim of the user evaluation process was to obtain answers, in some capacity, to the following:

- How intuitive the platform is
- How quickly users can complete both student and lecturer tasks
- Are there any common pitfalls?
- Would users recommend *AutoAssign* to others?

To achieve this, the following two experiments and a questionnaire were designed. All of these can be found in full in the appendices of this paper.

6.2.1 Design

Experiment A - Student Tasks In the first experiment, participants were asked to complete five tasks:

1. Create an account
2. Find and navigate to the 'Evaluation' assignment
3. Submit the provided `src.zip`
4. View the unit testing results
5. Use the built-in AI feedback assistant to determine the cause of the failed unit test

Each of these steps was designed around the intended student user experience. This provides both insightful feedback into core features, and demonstrated to the subject the use cases of the system, informing their response as to whether they would recommend it.

Six participants, who had no prior experience with the platform, were selected for the experiment and divided into two groups. The first group received a brief 5 minute training session on using the system before attempting the tasks, while the second group conducted the experiment with no introduction. This experiment aimed to assess both the system's functionality and its ease of use. The desired result would be for both groups to perform similarly, indicating that the system was intuitive and required minimal training. The metric being measured in this experiment was the length of time taken to complete each task.

Experiment B - Lecturer Tasks In the second experiment, participants were asked to complete four tasks:

1. Log into the provided account
2. Create a group with any name and add the account 'Test User' to it
3. Create a python assignment with any name and assign it to the group you made
4. Find the username of the submission with the shortest execution time for the "Java Test" assignment

Each of these steps was designed around the intended lecturer user experience. Providing similar insightful feedback as mentioned in Experiment A above, but for a different set of features. To clarify, it is not that lecturers are the ones completing this experiment, instead that the feature set being tested is the one that lecturers would use in a real world scenario.

Four participants, none of which had previous experience with the platform or had also participated in Experiment A, were selected for the experiment and also divided into two groups. Unfortunately, time constraints in the project prevented the enrolment of two additional subjects to match Experiment A.

It was noted that this would slightly affect the comparability of results between the two experiments. These groups had a similar educated vs. blind split as the other experiment for the same reasons, and the same metric was also used.

Questionnaire A brief questionnaire was also given to all 10 participants of the evaluation process. While experiments A and B each targeted individual feature sets and aimed to obtain objective data for analysis and comparison, the questionnaire was more subjective in nature.

The aim of this mixture of open-ended and Likert scale questions was to obtain an understanding of user satisfaction, potential developer oversights, and suggestions for future improvements. While the data for many of the responses will not be presented in charts like that of the experiments, trends in user's responses can still be analysed to identify prevalent issues.

The full questionnaire can be found in the appendix of this paper.

6.2.2 Results

Below is a summary of the findings of each of the stages of the user evaluation.

Experiment A

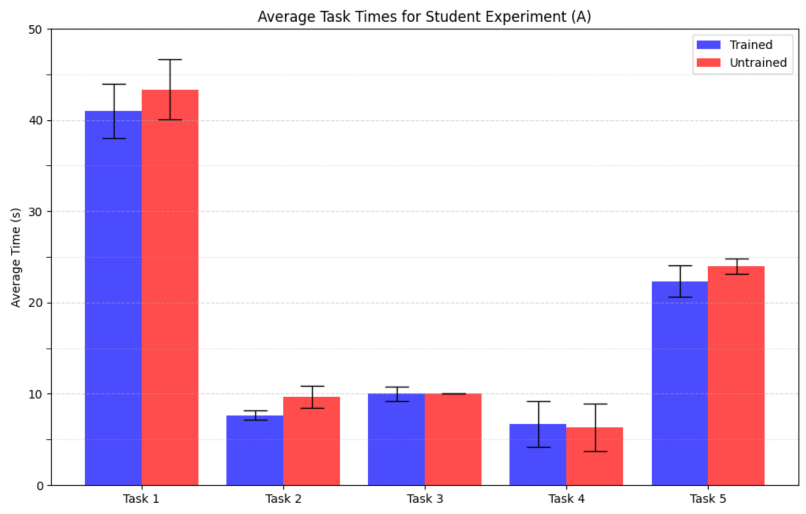


Figure 21: *A bar chart showing the differences in average task execution time for the student-oriented Experiment A. Blue bars represent the portion of the testing group that had prior training with the platform, and red bars represent those who did not. Also visible are error bars, demonstrating the range of values for each task.*

Experiment A was quick to reveal the intuitiveness of student-orientated workflow. This is evidenced by Figure 21, showing the lack of a notable execution time difference between the trained and untrained groups across all five tasks.

Particularly notable is the consistently quick speed of the navigational tasks (2 and 4), this clearly demonstrates the logical and user-friendly site layout performing its function as planned in the design chapter. It emphasises that users rarely mis-navigate or need to spend time considering what the consequences of their clicks may be, as they are confident in the logic of the site's structure.

The longest execution time was for the account creation task. Task 1's high average times can be attributed to the E-Mail confirmation service to verify accounts. Although candidates quickly found the sign up page, the process itself often took upward of 40 seconds as they opened their inboxes to complete their registrations. This process could be sped up significantly by removing this confirmation step, however, this opens the door to spam accounts and related malicious activity. With this consideration taken into account, the longer execution time was deemed acceptable.

Task 5, which required the use of the AI feedback assistant, also took participants considerably longer than tasks 2-4. All candidates initially accessed the 'Feedback and Assistance' drop-down on the submission page. Although this menu contains the AI chat box, it does not technically represent the first step of the feedback process. To initiate the chat, a code file must first be opened. This process necessitates opening the "Code" drop-down menu and selecting a file from the file tree. This perceived misdirection was noted during development, prompting the implementation of a restriction that prevents access to the AI chat until a file is selected, accompanied by an instructional pop-up. Upon encountering this message, all participants successfully navigated the code component, selected the appropriate file, and proceeded to receive their feedback.

To conclude on Experiment A, while the trained group did slightly outperform the untrained group, the gain in trained performance was not significant to the extent that the statement *"training is required to use this platform"* could be made. All tasks were completed with relative swiftness and minimal confusion was experienced throughout, with the slight exception of task 5.

Experiment B

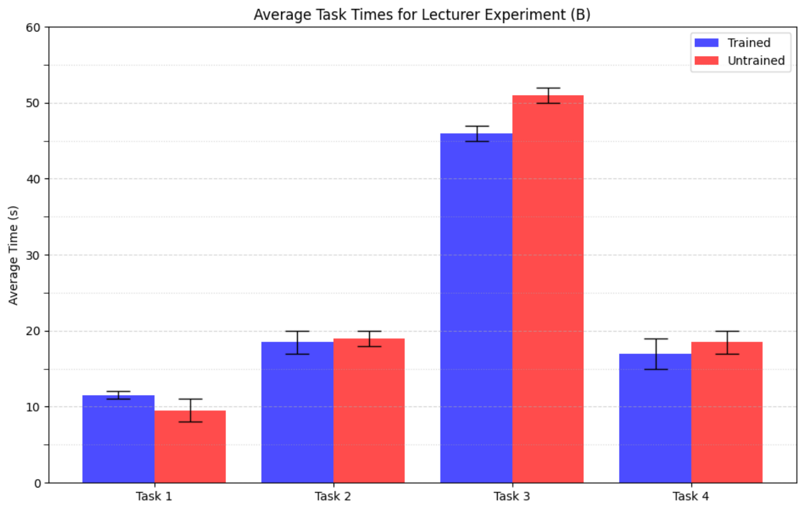


Figure 22: *A bar chart showing the differences in average task execution time for the lecturer-oriented Experiment B. Blue bars represent the portion of the testing group that had prior training with the platform, and red bars represent those who did not. Also visible are error bars, demonstrating the range of values for each task.*

Despite the different feature set, Experiment B reinforced many of the findings of Experiment A. As visible in Figure 22, both trained and untrained groups performed similarly, once again demonstrating the intuitive nature of the platform.

However, it differs from the student-orientated experiment in its much longer average times. This is likely due to the fact that many of the tasks required various forms of user input, as groups and assignments were created. These types of actions are not purely navigational and necessitate the reading and comprehension of various prompts, understandably requiring more time.

The most interesting task of the four was the third, where participants created their own assignments. Not only was this by far the longest task, requiring many steps to complete, but it also had the largest difference between trained and untrained groups. This can most probably be attributed to the preliminary walk-through provided to the trained participants. As a part of the short demonstration, an assignment was created. Having seen this, the subjects needed not read as closely when creating their own, as the untrained group would have had to.

To conclude on Experiment B, while the tasks generally took longer to complete than those in Experiment A, a minimal expertise gained was once again observed through preliminary training. This further corroborated the findings of the first experiment, strengthening the platform's claim to intuitiveness.

Questionnaire

While the questionnaire itself was quite short at only four questions, it did succeed in its role of providing valuable insights into the user experience. Before discussing the shortcomings, it is worth noting that 90% of respondents either agreed or strongly agreed that their experience was seamless and that they would recommend the platform to others. This marked a significant accomplishment, demonstrating that the careful requirement and design considerations had paid off to create an application that users both enjoyed and saw the benefit in.

In terms of user experience complaints, common responses included slow loading times and one respondent noted that the assignment creation process felt slightly *"long-winded"*.

The slow loading times can be attributed to the development build the experiments were performed on. In such an environment, pages often have to be re-compiled and performance enhancement processes such as minification have not yet been applied. If the site were to be fully compiled into a production build, these problems would likely disappear.

While it is true that the assignment creation process has multiple steps, many of these are unavoidable. In order to facilitate automation in the way that *AutoAssign* does, it must first be provided with the right tools. This necessitates a slightly longer set-up process than just an assignment title and a due date. Sub-dividing the creation process further to avoid overwhelming the user could be a potential solution to the respondent's complaint, but it could also be argued that this would only make the process more *"long-winded"*.

When asked about improvements for the platform, the most common notion was that of continuing to facilitate student-teacher interaction, instead of attempting to fully replace it. This is an understandable request, and was also listed as a could have requirement in the form of the lecturer's *"Ability to provide manual comments on student work"*. Given more time in the project, this would have become a priority to implement.

6.3 Requirements

In addition to the basic testing of functionality and empirical feedback of users, another gauge of project success is a comparison against the original requirements.

6.3.1 Design

The MoSCoW requirements in the analysis/requirements chapter were repurposed as a check-list. Through this, the final product of the project can be compared to the original concept, highlighting both shortcomings and where the product has exceeded expectations.

6.3.2 Results

Out of seventeen must have requirements, all seventeen of them were met. In hindsight, the goals for the project were likely slightly on the ambitious side, as the must have requirements category was by far the most overloaded one and closing out all seventeen points was no small feat. Ambition aside, by meeting all must have requirements the project can be classified as a Minimum Viable Product (MVP).

Additionally, five out of five should have requirements were also met. This demonstrates the further steps that were taken during development to extend the system's capabilities to facilitate a more complete and professional application.

However, less successful than the previous two categories were the could have requirements, none of which were met. While this could be interpreted as a failure of sorts, it must be remembered that the purpose of could have requirements is to define desirable features whose absence is still deemed acceptable.

6.4 Summary

To summarise the evaluation process, all aspects of the project were tested. From code reliability in the unit tests, to user behaviour and experience in the user evaluation, to a comparison against original expectations in the requirements.

Although challenges were faced at each stage, with unanticipated unit test complexity, development build limitations, and overly ambitious requirements. This comprehensive analysis into the various aspects of the project has provided the evidence to make the statement that *AutoAssign* as a platform has delivered on its requirements and fulfilled its intended purpose.

7 | Conclusion

Thus far, this paper has delved into the motivations, aims and requirements of *AutoAssign*. This was followed by an outline of the design and a detailed explanation of various aspects of the implementation. After this, a comprehensive evaluation was conducted, determining the success of the project. In this final chapter, the project will be summarised and a short reflection will detail the highs and lows of the project, discussing what would be done differently if it were to begin anew. To finish, beneficial future work will also be explored.

7.1 Summary

In summary, *AutoAssign* is a robust and streamlined solution to the modern nightmare of coding assignments for large student cohorts. The unit test execution architecture is as stripped-back as possible, allowing assignment creators to be flexible in their assignment specifications and testing setup. Also included is a novel competitive aspect, encouraging students to refine their solutions beyond the minimum requirement and promoting good software engineering practices. The product was thoroughly evaluated with unit tests, user feedback, and a requirements comparison. These processes brought to light the intuitive nature of the platform and the acclaimed seamless workflow it facilitates. Additionally, the unit testing process highlighted the solid foundations on which the product is constructed, and all the must- and should have requirements were met.

7.2 Reflection

Overall, the implementation process was the highlight of this project. This was a combination of smart technology choices that had good compatibility and documentation, and a systematic approach to both front- and back-end development, ensuring that one side does not advance too far ahead of the other. In addition, issue tracking and project management as a whole were well handled on the *Trello* board.

What didn't progress as smoothly was the user evaluation process. The coordination required to organise multiple experiments and a questionnaire was underestimated and therefore left too late, resulting in a slightly skewed 6/4 Experiment A/B split. While this oversight did not end up being detrimental, it was a valuable lesson in project and time management.

If I were to start over, I would shift the split between the different categories between the MoSCoW requirements, as the current set were rather ambitious. While it is true that all must- and should have requirements were met, it could be argued that this was at the detriment of other aspects of the project.

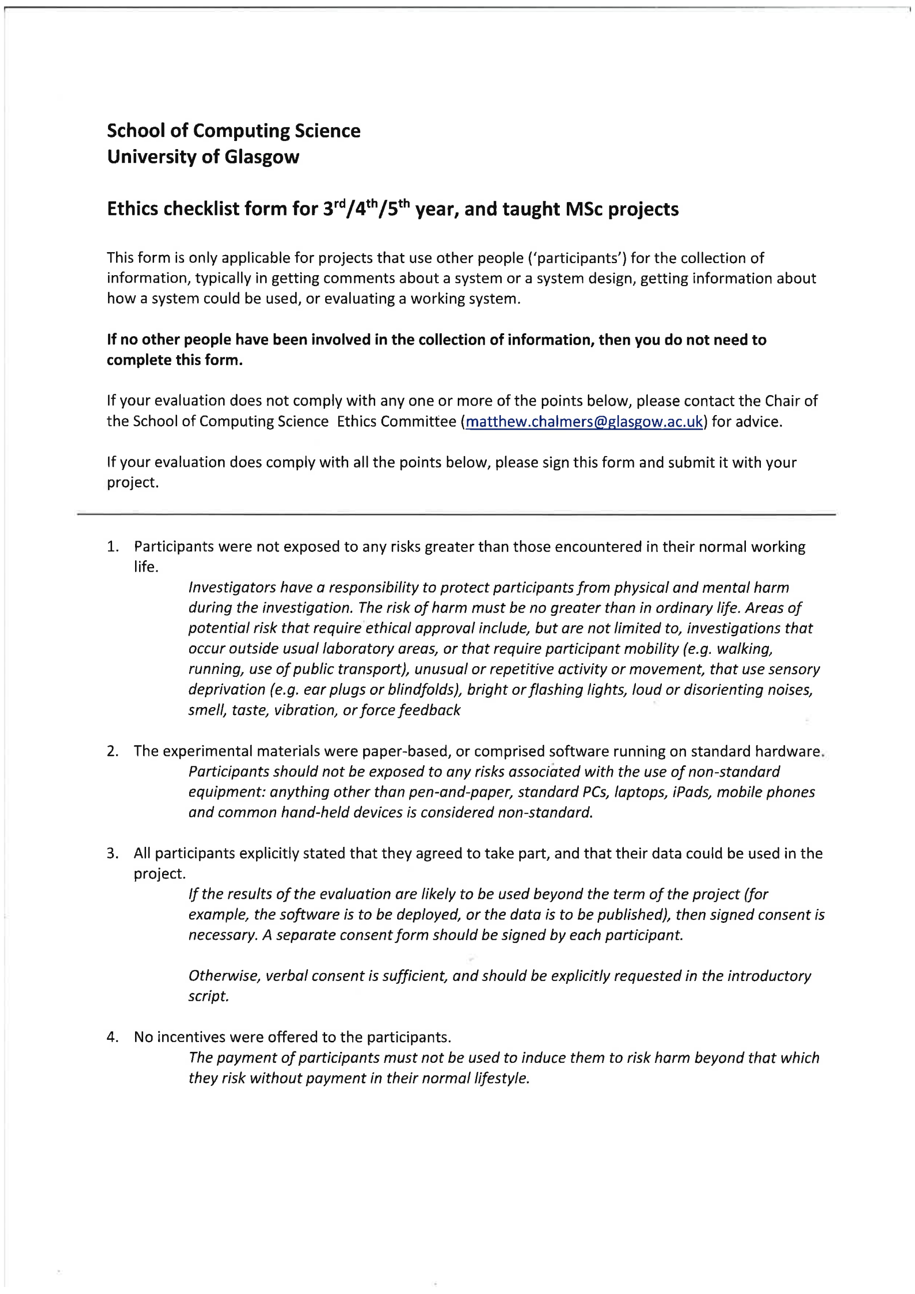
7.3 Future work

Due to the ambitious nature of the project, there were many requirements (particularly in the could have category) that were left unimplemented. It would therefore be a logical set of first steps to continue work on this project by implementing them.

One feature that might take priority over others would be a student-teacher written feedback chat, perhaps with code annotation functionality. There was a shared concern of a lack of human-to-human engagement in the learning process from multiple respondents in the questionnaire. The common consensus was that, while automation is of great use, it should not completely replace the dynamic between those who learn and those who teach.

Other interesting avenues to explore would be the further gamification of the application, incorporating features such as badges, XP points, and achievements for students to pursue. This would add additional incentive to improve submissions, further achieving *AutoAssign*'s goal of learning through competition.

A Ethics Checklist



School of Computing Science
University of Glasgow

Ethics checklist form for 3rd/4th/5th year, and taught MSc projects

This form is only applicable for projects that use other people ('participants') for the collection of information, typically in getting comments about a system or a system design, getting information about how a system could be used, or evaluating a working system.

If no other people have been involved in the collection of information, then you do not need to complete this form.

If your evaluation does not comply with any one or more of the points below, please contact the Chair of the School of Computing Science Ethics Committee (matthew.chalmers@glasgow.ac.uk) for advice.

If your evaluation does comply with all the points below, please sign this form and submit it with your project.

---

1. Participants were not exposed to any risks greater than those encountered in their normal working life.
   Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life. Areas of potential risk that require ethical approval include, but are not limited to, investigations that occur outside usual laboratory areas, or that require participant mobility (e.g. walking, running, use of public transport), unusual or repetitive activity or movement, that use sensory deprivation (e.g. ear plugs or blindfolds), bright or flashing lights, loud or disorienting noises, smell, taste, vibration, or force feedback

2. The experimental materials were paper-based, or comprised software running on standard hardware.
   Participants should not be exposed to any risks associated with the use of non-standard equipment: anything other than pen-and-paper, standard PCs, laptops, iPads, mobile phones and common hand-held devices is considered non-standard.

3. All participants explicitly stated that they agreed to take part, and that their data could be used in the project.
   If the results of the evaluation are likely to be used beyond the term of the project (for example, the software is to be deployed, or the data is to be published), then signed consent is necessary. A separate consent form should be signed by each participant.

   Otherwise, verbal consent is sufficient, and should be explicitly requested in the introductory script.

4. No incentives were offered to the participants.
   The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

5. No information about the evaluation or materials was intentionally withheld from the participants.
   Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

6. No participant was under the age of 16.
   Parental consent is required for participants under the age of 16.

7. No participant has an impairment that may limit their understanding or communication.
   Additional consent is required for participants with impairments.

8. Neither I nor my supervisor is in a position of authority or influence over any of the participants.
   A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.

9. All participants were informed that they could withdraw at any time.
   All participants have the right to withdraw at any time during the investigation. They should be told this in the introductory script.

10. All participants have been informed of my contact details.
    All participants must be able to contact the investigator after the investigation. They should be given the details of both student and module co-ordinator or supervisor as part of the debriefing.

11. The evaluation was discussed with all the participants at the end of the session, and all participants had the opportunity to ask questions.
    The student must provide the participants with sufficient information in the debriefing to enable them to understand the nature of the investigation. In cases where remote participants may withdraw from the experiment early and it is not possible to debrief them, the fact that doing so will result in their not being debriefed should be mentioned in the introductory text.

12. All the data collected from the participants is stored in an anonymous form.
    All participant data (hard-copy and soft-copy) should be stored securely, and in anonymous form.

Project title: AutoAssign – Online Coding Assignment Automation Platform

Student's Name: **Will Murphy**

Student Number: 2766915M

Student's Signature: WILL MURPHY

Supervisor's Signature [illegible]

Date: 17/03/2025

B | User Evaluation Experiment A Handout

AutoAssign User Evaluation Experiment A

To be completed anonymously

If you have any concerns, contact me: Will Murphy, `2766915m@student.gla.ac.uk`

This experiment attempts to mimic a typical student workflow. You will be asked to complete 5 tasks and note down how long (in seconds) it took you to complete each.

After finishing the experiment, you should then proceed to complete this short questionnaire where you will summarise your user experience.

> **Task 1: Create an account**
>
> Sign up and activate an account on the platform.
> Stop the timer once you are logged in on the dashboard.

Important: Let me know once you have completed this task, and tell me your user name or E-Mail. This is required to continue the experiment, as I must add you to an assignment.

> **Task 2: Find and navigate to the 'Evaluation' assignment**
>
> Stop the timer once you are on the assignment page.

Important: Before continuing, download this `src.zip` file. This is required to make a submission.

> **Task 3: Submit the provided zipped source code folder**
>
> Stop the timer once you have received the notification that the unit tests are complete.

> **Task 4: View the unit testing results**
>
> Stop the timer once you visually confirmed the number of unit tests that have passed and failed.

> **Task 5: Use the built-in AI feedback assistant to determine the cause of the failed unit test**
>
> Stop the timer once the AI assistant has helped you figure out what the problem is.

This concludes the experiment, please now complete the questionnaire linked at the top of the document.

Thank you for your cooperation.

C | User Evaluation Experiment B Handout

AutoAssign User Evaluation Experiment B

To be completed anonymously

If you have any concerns, contact me: Will Murphy, `2766915m@student.gla.ac.uk`

This experiment attempts to mimic a typical lecturer/course coordinator workflow. You will be asked to complete 4 tasks and note down how long (in seconds) it took you to complete each.

After finishing the experiment, you should then proceed to complete this short questionnaire where you will summarise your user experience.

Task 1: Log into the provided account

Account details:

- **E-Mail:** williamalexandermurphy1@gmail.com
- **Password:** 123456

Stop the timer once you are logged in on the dashboard.

Task 2: Create a group with any name and add the account 'Test User' to it

Stop the timer once you see the 'Test User' in the group users table.

Important: Before continuing, download this `pysrc.zip` file. This is required to create an assignment.

Task 3: Create a python assignment with any name and assign it to the group you made

Stop the timer once you finish the set-up process and the assignment page refreshes.

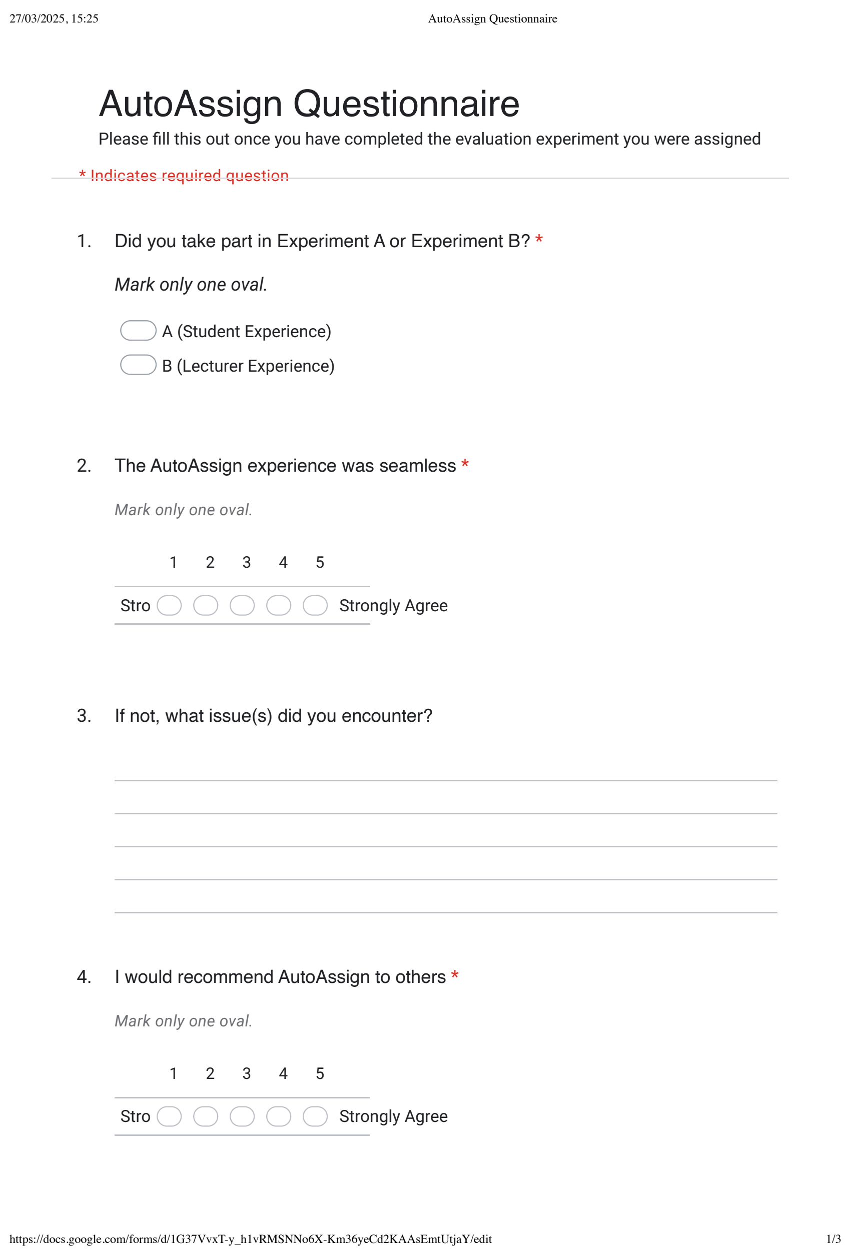
Task 4: Find the username of the submission with the shortest execution time for the "Java Test" assignment

Stop the timer once you visually confirmed the name of the user who is first on the leaderboard for execution time.

This concludes the experiment, please now complete the questionnaire linked at the top of the document.

Thank you for your cooperation.

D | User Evaluation Questionnaire



AutoAssign Questionnaire

Please fill this out once you have completed the evaluation experiment you were assigned

* Indicates required question

1. Did you take part in Experiment A or Experiment B? *

   Mark only one oval.

   ◯ A (Student Experience)
   ◯ B (Lecturer Experience)

2. The AutoAssign experience was seamless *

   Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

3. If not, what issue(s) did you encounter?

4. I would recommend AutoAssign to others *

   Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

5. If not, what improvement(s) would you first like to see?

E Source Code README.md

Also available here: https://github.com/Illogicalll/Automarker/blob/main/src/manual.md

AutoAssign

AutoAssign is a robust and streamlined solution to the modern nightmare of coding assignments for large student cohorts.

Codebase Overview

Within this `/src/` folder there are 7 folders:

- `__tests__`: contains the Jest unit tests for the project
- `app`: contains the pages and site structure
- `components`: contains various re-usable UI components used to construct the site
- `lib`: contains a utility function file
- `public`: contains site assets
- `utils`: contains various useful functions to do with authentication, redirects, and more

Aside from the folders, there are also various configuration files such as `tsconfig.json` or `jest.config.js`. These specify many parameters that are important for the project to run.

Requirements

The following technologies are required to facilitate the automated unit test execution API:

- MacOS/Linux based system
- `node`
- `cmake`
- `gnu-time` (MacOS) / `time` (Linux)
- `openjdk`
- `maven`
- `python3`

These can all be installed with the following commands:

- MacOS:
  - `brew install node cmake gnu-time openjdk maven python3` (requires homebrew)
- Linux:
  - `sudo apt update && sudo apt install -y nodejs build-essential cmake time default-jdk maven python3 python3-pip python3-venv`

Setup Instructions

- Clone from the GitHub repository here (https://github.com/Illogicalll/Automarker)
- Open a terminal window in the cloned repository and navigate to the `/src/` folder
- Run `npm i`
- Create a file called `.env.local`
- Create three environment variables:
  - `NEXT_PUBLIC_SUPABASE_URL`
  - `NEXT_PUBLIC_SUPABASE_ANON_KEY`
  - `NEXT_PUBLIC_HF_API_KEY`
- Contact me at 2766915M@student.gla.ac.uk or contact@w-murphy.com for the values
- Run `npm run dev`
- Navigate to `localhost:3000` (port may be different if `3000` is in use)

Testing

If you wish to execute the unit tests for yourself to ensure the code is functional, do the following:

- In the `/src/` folder find `tsconfig.json`
- Change the value of `"jsx"` from `"preserve"` to `"react-jsx"`
- Save the file
- In your terminal run `npx jest`

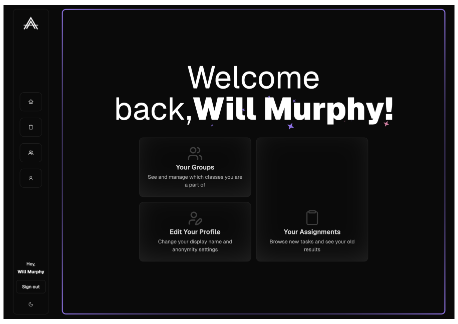
F Source Code manual.md

Also available here: https://github.com/Illogicalll/Automarker/blob/main/src/manual.md

User Manual

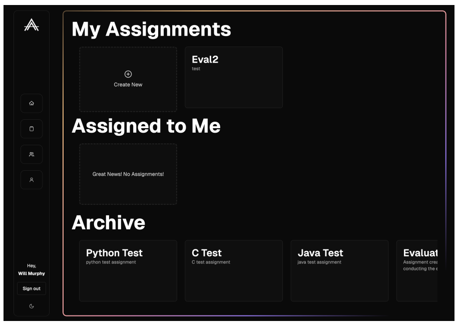
Much of AutoAssign's functionality is quite self-explanatory. Regardless, here is a brief rundown of the core functionality:

Home Page


Welcome back, Will Murphy!
Your Groups
See and manage which classes you are a part of
Edit Your Profile
Change your display name and anonymity settings
Your Assignments
Browse new tasks and see your old results
Hey,
Will Murphy
Sign out

Here you can find useful quicklinks and short descriptions for the different features of the site

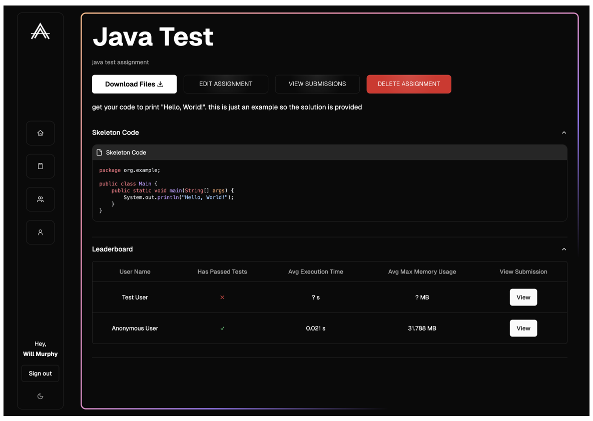
Assignments



Here you can find three lists:

- My Assignments
    - Assignments that you have created
- Assigned to Me
    - Assignments you have been set
- Archive
    - Assignments that you have created or been set that are past their due dates

An Assignment



Java Test

java test assignment

get your code to print "Hello, World!". this is just an example so the solution is provided

Skeleton Code

Skeleton Code

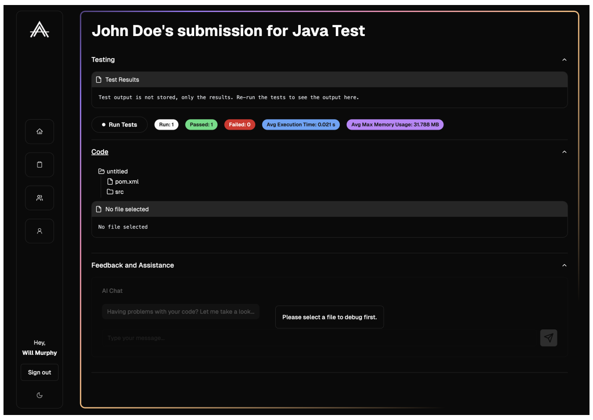
```java
package org.example;

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Leaderboard

User Name	Has Passed Tests	Avg Execution Time	Avg Max Memory Usage	View Submission
Test User	✗	? s	? MB	View
Anonymous User	✓	0.021 s	31.788 MB	View

Here you can find information and useful functionality relating to a particular assignment, from reading and the downloading skeleton code to looking at the leaderboard (if the due date for the assignment has passed or you own the assignment)

A Submission



John Doe's submission for Java Test

Testing

Test Results

```
Test output is not stored, only the results. Re-run the tests to see the output here.
```

Run Tests | Run: 1 | Passed: 1 | Failed: 0 | Avg Execution Time: 0.021 s | Avg Max Memory Usage: 31.788 MB

Code

```
untitled
  pom.xml
  src
```

No file selected

```
No file selected
```

Feedback and Assistance

AI Chat

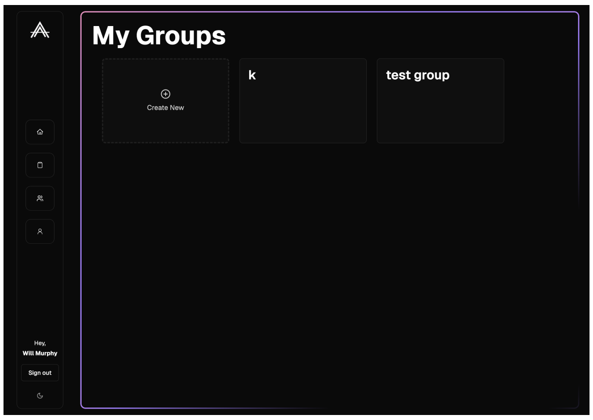
Having problems with your code? Let me take a look...

Please select a file to debug first.

Here you will find three drop-downs:

- Testing
  - Here you can manually execute unit tests and observe output
- Code
  - Here you can browse the files of the uploaded submission
- Feedback and Assistance
  - Here you can prompt the AI assitance for advice

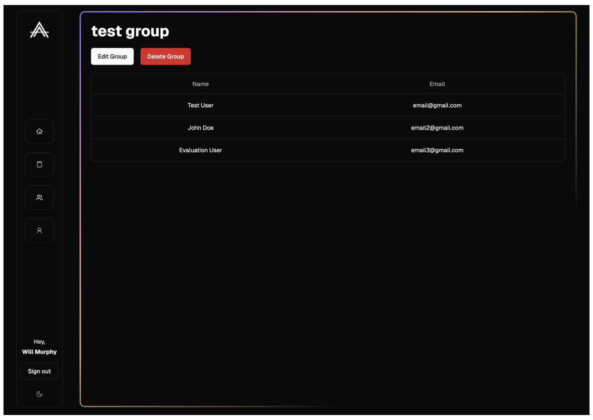
Groups



My Groups

Groups are how assignments are assigned to students. They contain multiple students

A Group

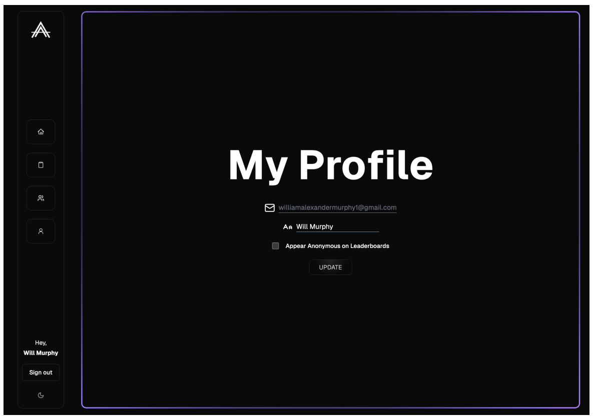


test group

Name	Email
Test User	email@gmail.com
John Doe	email2@gmail.com
Evaluation User	email3@gmail.com

Here you can manage your group, by adding, removing or viewing who is a part of it

My Profile



My Profile

williamalexandermurphy1@gmail.com

Will Murphy

☐ Appear Anonymous on Leaderboards

Here there are some basic options such as changing your display name or hiding it from the leaderboards

Bibliography

[1] "Best Jobs in the UK" Glassdoor, 2022. URL: https://www.glassdoor.co.uk/List/Best-Jobs-in-UK-LST_KQ0,15.htm.

[2] "Digitalisation European Central Bank and its impact on the economy: insights from a survey of large companies", 2018. URL: https://www.ecb.europa.eu/press/economic-bulletin/focus/2018/html/ecb.ebbox201807_04.en.html.

[3] "AI trend drives rise in students wanting to study computing" BBC, 2023. URL: https://www.bbc.co.uk/news/technology-66178247#:~:text=This.%20year's%20application%20data%20showed,almost%2010%25%20compared%20to%202022.

[4] "[Blog] Celebrating 100000 Scholarships for Students Around the World" lilybird @ Codeacademy, 2020. URL: https://discuss.codecademy.com/t/blog-celebrating-100-000-scholarships-for-students-around-the-world/488327#:~:text=learners%20join%20the-,45%20million,-other%20learners%20using.

[5] "Computer science students' perspectives on assessment methods" Student Voice, 2024. URL: https://www.studentvoice.ai/blog/computer-science-students-perspectives-on-assessment-methods/#:~:text=deciphering%20the%20requirements.-,Workload%20and%20Support%20Concerns,-Many%20computer%20science.

[6] "Turnitin Originality" Turnitin. URL: https://www.turnitin.co.uk/products/originality/.

[7] "Turnitin Gradescope" Turnitin. URL: https://www.turnitin.co.uk/products/gradescope/.

[8] "Turnitin Acquires Gradescope" Turnitin, 2018. URL: https://www.turnitin.com/press/turnitin-acquires-gradescope.

[9] "Optimize your hiring with HackerRank" HackerRank. URL: https://www.hackerrank.com/solutions/optimize-hiring/.

[10] "Automate your course Github Classroom and focus on teaching". URL: https://classroom.github.com/.

[11] "CodeAcademy Home Page" CodeAcademy. URL: https://www.codecademy.com/#:~:text=The%20platform-,Hands%2Don%20learning,-AI%2Dassisted%20learning.

[12] Umut Cihan, Vahid Haratian, Arda İçöz, Mert Kaan Gül, Ömercan Devran, Emircan Furkan Bayendur, Baykal Mehmet Uçar, and Eray Tüzün. Automated code review in practice, 2024. URL: https://arxiv.org/abs/2412.18531, arXiv:2412.18531.

[13] Manushree Vijayvergiya, Małgorzata Salawa, Ivan Budiselić, Dan Zheng, Pascal Lamblin, Marko Ivanković, Juanjo Carin, Mateusz Lewko, Jovan Andonov, Goran Petrović, Daniel Tarlow, Petros Maniatis, and René Just. Ai-assisted assessment of coding practices in modern code review. In *Proceedings of the 1st ACM International Conference on AI-Powered Software*, AIware '24, page 85–93. ACM, July 2024. URL: http://dx.doi.org/10.1145/3664646.3665664, doi:10.1145/3664646.3665664.

[14] "HuggingFace Home Page" HuggingFace. URL: https://huggingface.co/.

[15] "Elements of a Prompt" Prompting Guid. URL: https://www.promptingguide.ai/introduction/elements.

[16] "Playground" HuggingFace. URL: https://huggingface.co/playground.

[17] Herman Zvonimir Došilović and Igor Mekterovic. Robust and scalable online code execution system, 09 2020. doi:10.23919/MIPRO48935.2020.9245310.

[18] HuggingFace. URL: https://huggingface.co/docs/api-inference/en/index.

[19] Pietro Schirano. @shadcn/ui - design system | figma community, Jul 2023. URL: https://www.figma.com/community/file/1203061493325953101.

[20] shadcn, Jan 2025. URL: https://ui.shadcn.com/.

[21] "Supabase Homepage" Supabase. URL: https://supabase.com/.

[22] "Next.js Homepage" Next.js. URL: https://nextjs.org/.

[23] "MagicUI Homepage" MagicUI. URL: https://magicui.design/.

[24] Stuart Knightley. URL: https://stuk.github.io/jszip/.