TOM SWAINSTON

GCSE ADVENT OF CODE

A-Level Computer Science Project

Table of Contents

Analysis

The problem

There is a substantial need for developers and many young people lack the skills to code. Previously, OCR GCSE Computer Science contained a programming project as a part of the GCSE award, however, the assessment of this component was poorly planned by JCQ resulting in students being able to “cheat” leading to its removal from the qualification in 2017. As a result, fewer younger people learn how to code, which is an excellent way to develop computational thinking and problem-solving skills. Finally, knowing how to code is a very useful life skill and can be applied in many careers.

How the problem be solved through computational methods

The use of computational methods to solve this issue is very suitable as the final product will be a primarily coding-based game. As it is code-based, lots of analysis will be needed.

Abstraction

The development of the solution can be accelerated through abstraction. An example of abstraction in my solution is the database user information. Users' information will only store the required information used by the program, such as their progression level through the Calendar and their username. Furthermore, I will be abstracting time to allow students to make their way through the challenges at their own pace. This makes the learning more optimal for the student and also means that no issues are caused if a student is busy for a day or would like to have a break as they will not “miss” a challenge.

Decomposition

My solution can be broken down into multiple different features that I can individually tackle and eventually combine to create a flowing program, examples of these problems are as so:

- Logging into accounts
- Creating new accounts
- Retrieving challenges from a database
- Accepting an answer input from a student
- Marking students answers and giving appropriate feedback
- Presenting a leaderboard to the student
- Updating leaderboards to make them accurate as much as possible

Suitability

Currently, there is a yearly coding challenge known as the “Advent of Code”, its principal being daily new coding challenges that progressively get harder each day of December, easy to start and almost impossible by the end. However, I aim to recreate this idea, starting even more basic and including the required coding knowledge for the OCR GCSE Computer Science exam, such as loops and 1D arrays.

Stakeholders

Stakeholder #1 - The Students

The primary users of the solution will be students who have recently started their GCSEs and who are now trying to learn how to code. The solution will be appropriate for the students because it will introduce new coding ideas to students who are new to coding yet also challenge those who are competent at coding. Providing them with instant feedback will keep the students engaged and possibly even create competition between students, creating motivation to continue and learn.

Furthermore, each challenge will carry a key concept of coding that the students will need for their GCSE qualification. By the end of all the challenges, they will understand all they need to know to do well in the programming component.

Stakeholder #2 - The teachers

My solution will also hugely benefit the teachers of the students; it will save them precious time that they could spend teaching other, more challenging to grasp, topics. The solution will provide instant feedback to the teacher informing them of the progression of each student and the working time spent on each challenge. Teachers will also be able to see a leaderboard of students who have completed the most challenges and other statistics such as the cumulative time spent on the application meaning they can reward students for their effort.

Stakeholder #3 - The exam board

The exam board could be affected by my solution because with students gaining access to higher marks in the programming-related part of the exam it could well prompt the return of a programming-related project for GCSE students, although this would need lots of planning and a total rewrite of the previous programming project as it had many flaws.

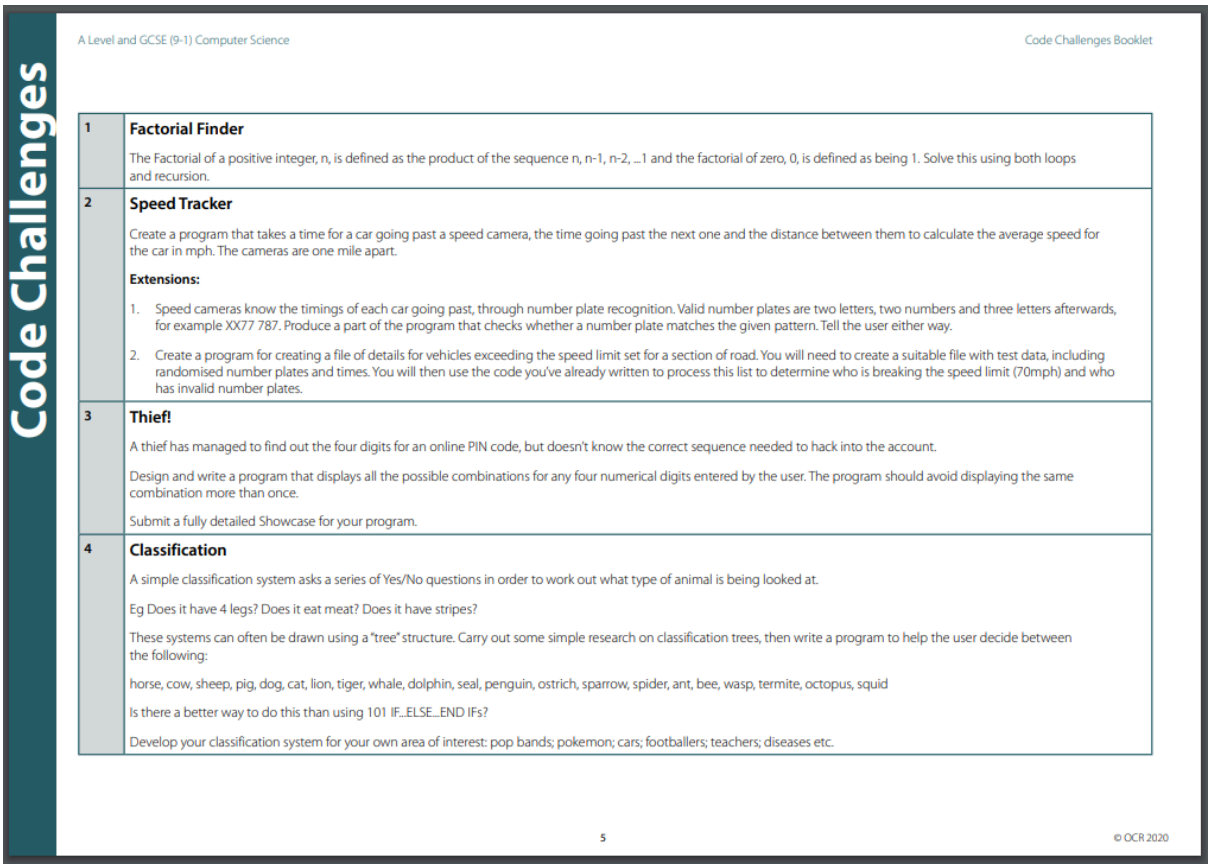
Stakeholder #4 - Future Employers

Future employers will be affected by my solution because whilst the students are working through the challenges they are subconsciously developing key skills that

employers look for such as: problem-solving, built by solving the harder problems; easy teachability, the students will be teaching themselves through exposing themselves to new teaching methods making them more teachable; teamwork, students will be able to work with their peers to help each other solve challenges; abstract thinking is another key skill that is used in the wider world outside of programming that is built from programming.

Similar systems

Example #1



Code Challenges

A Level and GCSE (9-1) Computer Science — Code Challenges Booklet

1	**Factorial Finder** The Factorial of a positive integer, n, is defined as the product of the sequence n, n-1, n-2, ...1 and the factorial of zero, 0, is defined as being 1. Solve this using both loops and recursion.
2	**Speed Tracker** Create a program that takes a time for a car going past a speed camera, the time going past the next one and the distance between them to calculate the average speed for the car in mph. The cameras are one mile apart. **Extensions:** 1. Speed cameras know the timings of each car going past, through number plate recognition. Valid number plates are two letters, two numbers and three letters afterwards, for example XX77 787. Produce a part of the program that checks whether a number plate matches the given pattern. Tell the user either way. 2. Create a program for creating a file of details for vehicles exceeding the speed limit set for a section of road. You will need to create a suitable file with test data, including randomised number plates and times. You will then use the code you've already written to process this list to determine who is breaking the speed limit (70mph) and who has invalid number plates.
3	**Thief!** A thief has managed to find out the four digits for an online PIN code, but doesn't know the correct sequence needed to hack into the account. Design and write a program that displays all the possible combinations for any four numerical digits entered by the user. The program should avoid displaying the same combination more than once. Submit a fully detailed Showcase for your program.
4	**Classification** A simple classification system asks a series of Yes/No questions in order to work out what type of animal is being looked at. Eg Does it have 4 legs? Does it eat meat? Does it have stripes? These systems can often be drawn using a "tree" structure. Carry out some simple research on classification trees, then write a program to help the user decide between the following: horse, cow, sheep, pig, dog, cat, lion, tiger, whale, dolphin, seal, penguin, ostrich, sparrow, spider, ant, bee, wasp, termite, octopus, squid Is there a better way to do this than using 101 IF...ELSE...END IFs? Develop your classification system for your own area of interest: pop bands; pokemon; cars; footballers; teachers; diseases etc.

5 — © OCR 2020

Description

After a look around online for similar websites, I found a booklet created by OCR that has the same intentions as my solution which is to improve the coding level of GCSE students. The booklet is a 30-page document stuffed with different coding challenges varying in difficulty and some of them have extra extensions to the challenges.

Link: https://www.ocr.org.uk/images/260930-coding-challenges-booklet.pdf

Pros

- The booklet was created by OCR who made the specification for the GCSE which should mean that the challenges are relevant and would help students build their knowledge
- The booklet can be downloaded meaning that students do not need access to the internet to be able to use the booklet to its full potential assuming that they have a way of coding offline
- There are 80 examples meaning there is a wide range of different challenges that both GCSE and A-Level students can use to help themselves to improve.

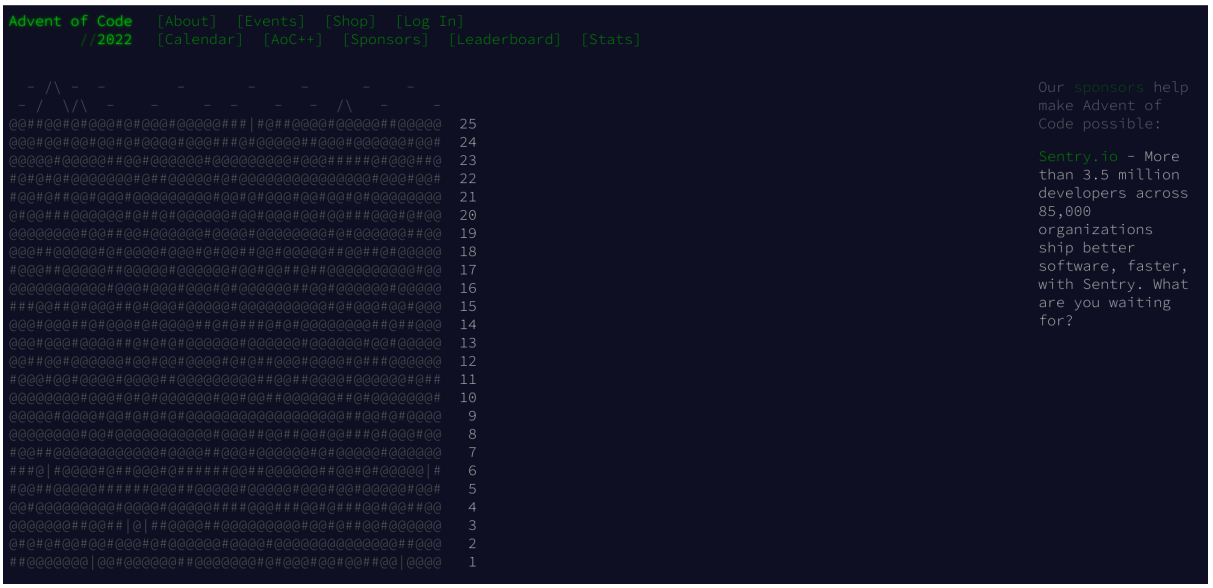
Cons

- There is no interaction with the booklet and there are only examples meaning that students could easily get bored
- It's less of a teaching tool and more of a practising tool as there is no feedback
- It does not label which challenges are easier than others so students who have never coded before may struggle with particular challenges whereas others may be doing challenges which are too easy for their ability

Evaluation

Overall, this is a very useful tool for students who just need to practise coding and the challenges are very effective. However, the booklet cannot provide any live feedback or leaderboard system for students, meaning that teachers may be unable to track students' progress easily.

Example #2



Description

The principle of my project originated from this website, Advent Of Code. The website's idea is that each day of December there is a new, more challenging, programming task to complete each day and you can compete with other members of a class, or internationally if you'd like. The website is a very basic and easily navigable website which I intend to mimic.

Link: https://adventofcode.com/

Pros

- The website has a user system where it allows you to create an account and stores your data and progress against that data, a principle I plan to include
- It contains a leaderboard system which can compare you to other players that you choose, such as classmates or global leaderboards

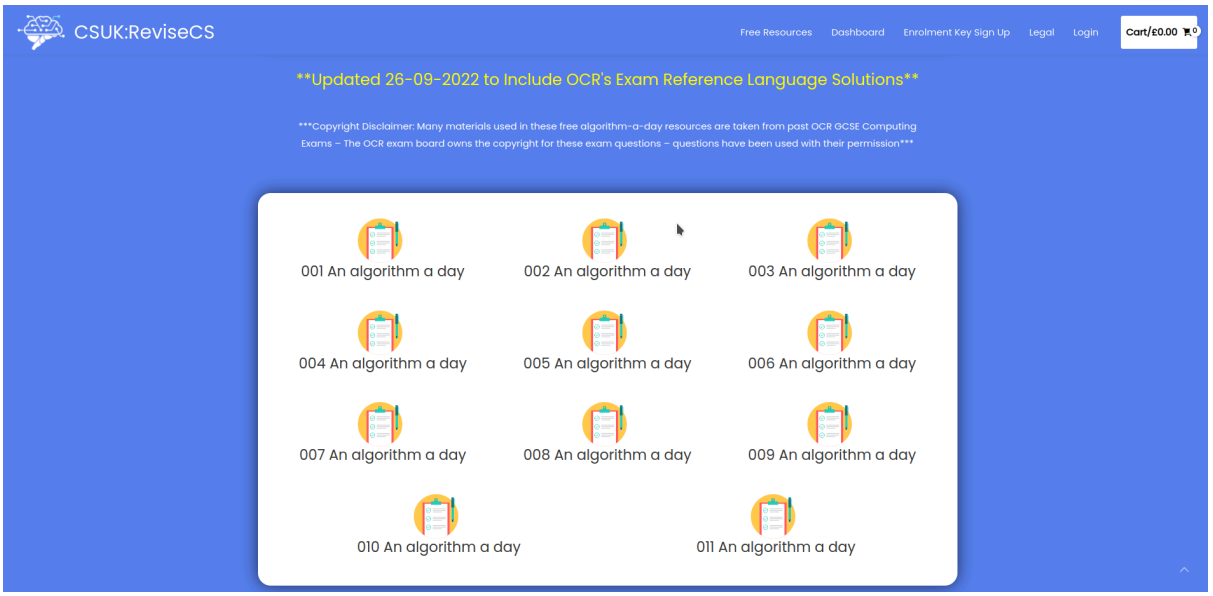
Cons

- The daily tasks can become very, very challenging and most GCSE students would not be able to complete the challenges
- The challenges don't contain GCSE-specific ideas meaning that some of them may not be beneficial to the student
- The basic idea is to challenge the users and not to teach them which is what I aim to do along the way of them completing the challenge

Evaluation

The website does have many key features that I wish to implement into my solution such as its leaderboard and user account system, however, there are still many key features that aren't implemented into Advent Of Code that I would like to implement, such as the ability to “release” tasks to students. This feature would allow teachers to set the challenges at a pace that matches their class progression as well as being able to let students who are competent in coding jump to harder tasks. Furthermore, there isn’t a way for teachers being able to track how long a student has spent on a challenge or on the applications as a whole which would be a useful feature to check if students are using pre-made solutions or on the flip side awarding students who are trying hard on a challenge.

Example #3



Description

This resource, “An Algorithm A Day” provides GCSE students with challenges that they can use in the leadup to their final exams. Each daily challenge includes a downloadable PDF document, a question which requires them to write an algorithm which carries out a given task, and an example algorithm that carries out the given task (an example answer).

Link: https://revisecs.computerscienceuk.com/algorithm-a-day/

Pros

- Tailored to specifically suit OCR GCSE students
- Although I did not look into it, there is an option for a “Teacher Sign-up” which is a feature that I would like to add to my solution
- There are example answers to help guide students to improve

Cons

- There aren’t many challenges to choose from which means students will run out of practice quickly
- There is no interactive feedback and the website never “checks” your answer, just gives you an example
- The example answers don’t contain a set of criteria which you would expect they would as most GCSE questions are marked using a set of given criteria

Evaluation

This example is the closest a website gets to my desired solution. It tailors for GCSE students, has example answers and has (not tested) teacher logins. The only downside to this website is the general lack of questions that it has. Although the website is tailored explicitly for GCSE students, the questions do not look like they have been taken from past papers and therefore may not replicate the difficulty or reality of what they may face in an exam. As well as this, a student can easily be dishonest with themselves and scroll to see the answer.

How will my solution be more beneficial than other systems?

As I mentioned above I aim to make my solution as beneficial to the teachers as it is to the students. Teachers will be able to track students' progress to allow them to plan and alter upcoming lessons to target the areas of programming that need attention. I will also allow teachers to look at the time students have spent on the solutions as a whole and also individual time spent on each challenge which could help teachers find topics where students are not stuck but more challenged.

The solution will be interactive and specifically tailored to GCSE students. Many other systems have interactive systems but do not offer specific areas or specifications of a course. On the other hand, some systems tailored specifically to GCSE students however fail to provide feedback to the user which could lead to bad practice and consistent incorrect mistakes leading to extra learning required to reverse the drilled-in incorrect practice.

There will be a competitive aspect of the system which will not only give the students motivation to complete the challenges and beat their classmates but also gives the teacher a simple way of viewing students who deserve positive recognition but also students who could help other students who may be struggling on certain challenges. A downside to this system is that it will most likely be based on a point system where points are gained upon completion of challenges which doesn’t congratulate students who are finding it challenging but still persevering, however, this is where tracked time on the soliton will become beneficial.

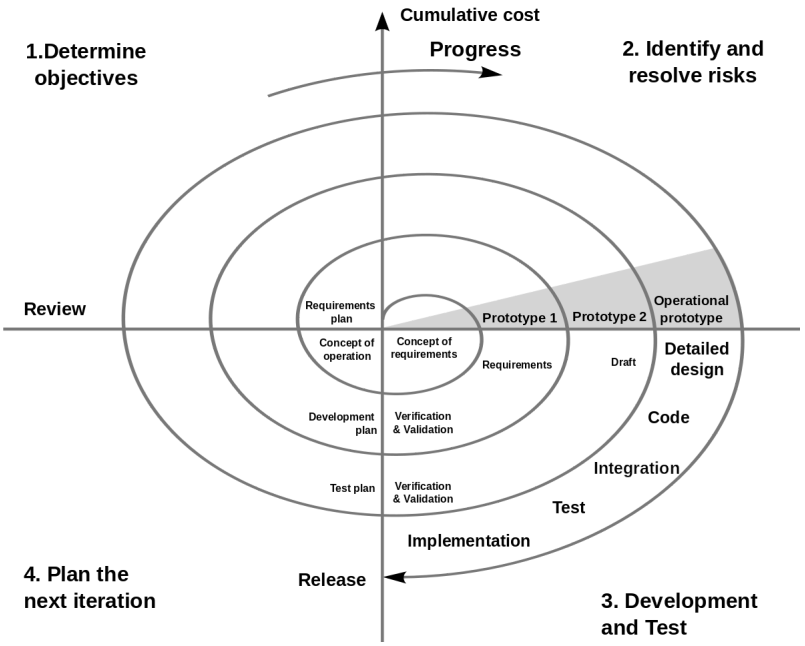
Unlike Example #2, my solution will take a user's code, instead of data, run the inputted code, run test data through the code and see if the outputs are correct. This way it means that the user can easily input external data through a separate file which can be time-consuming and sometimes quite confusing to a new user. Furthermore, with code being inputted there is a much lower chance of the user completing the challenge without writing code and instead just working out the data themselves.

An issue with the example solutions is that students are given identical questions meaning identical answers can be copied and pasted from other people. As this solution aims to build personal development skills this would not be beneficial for the students. I am to rule out this possibility by randomly generating variables for each question. For example, if a challenge asks to calculate the price of a taxi ride, I will randomly generate the cost per mile and base fee.

My approach to creating a successful solution

My solution will be developed using the spiral methodology which includes multiple prototypes and a repeating loop of designing a new prototype from the flaws of previous prototypes, analysing the issues creating a solution and releasing a new prototype until a stable version is reached.

A diagram from Wikipedia shows how the spiral methodology works.



https://en.wikipedia.org/wiki/Spiral_model

Benefits of this method

The benefits of this method are that the continual development of the solution allows you to find bugs and issues with the most recent prototype and implement a solution into the next prototype release. Furthermore, if a prototype causes issues or has unfixable flaws you can revert to the most recent prototype instead of having broken code

The methodology of releasing prototypes gives the customers/stakeholders an idea of the final product that they will be receiving, allowing them to suggest changes and due to the continual development of the solution, changes can be made without hugely altering the progression of development.

Drawbacks of this method

There is a possibility that time may become an issue with this methodology as it is hard to predict how long it takes for a prototype to be released and how many prototypes will need to be released until the final product, especially if large changes are needed. As well as this, there are multiple stages which can be quite complex and protocols must be followed accurately for the method to work fully.

Limitations to creating a successful solution

My solution will be a small-scale system which will not be designed to be hosted and used by thousands of people. The aim would be that an absolute maximum number of concurrent users would sit around 40, the average intake of GCSE Computer Science students at Hurst. It will be designed to fit around that number of people on the leaderboard page, as well as other pages where all students are being displayed, the teacher portal for example, and so with more students than that, user interface errors and pages will be displayed incorrectly.

I do not plan to host the solution permanently but instead, host it locally on my machine whilst developing. If the solution were to be used by the college it would need to be hosted on a domain by a server capable of handling the number of students doing computer science. As well as this the web pages would need to be updated to cater for students in previous years or the database would need to be cleared periodically.

The content of the website will also only be suited for OCR GCSE computer science solutions. So students who are doing other exam boards will be able to use my solution but it may not be as beneficial as other tools.

My solution aims to enhance the coding ability of students, which in reality is very difficult to make consistent as all students learn in different ways. My solution may work very well for one student and not work at all for another. It is hard to accurately conclude that my solution will help *all* students that use it.

Application requirements

Requirements	Justification

Input
1. Sign up/Log in
  ○ First & Last name
  ○ (Hurstpierpoint college) E-mail address
  ○ Password & repeated password
  ○ Forgotten password button which invokes a password reset
2. Password change button after confirming the current (active) password
3. Ability to control notifications such as automated weekly emails
4. Input box for challenge answers that is scrollable to view all code
	1. This is required to enable students to log into their own accounts. This feature means that students are able to start the challenges, log out and come back later to complete them. It also means that teachers are able to differentiate who has done which challenges
	2. This would be needed in case a student forgets their password
	3. The student should be able to personalise the notifications they receive from the website
	4. A scrollable feature would be needed in the case that the student is coding within the website. Without this feature they would not be able to view all of their code

Output ● Alert the user if an inputted login credential is incorrect and prompt them to edit and re-enter ● Show the user the current leaderboard as well as the number of challenges they have completed ● Emails, if they are subscribed, to show them their weekly update ● Show the result of the unit tests run on their inputted code, including the data inputted and outputted by their program	1. The student should be made aware if the credentials they have entered are not correct 2. The student should not find it difficult to find out how far through the challenges they are 3. If an email is provided, this will be the primary method of communication 4. This is required to let the student know whether the code they have entered passes all the unit tests that have been run on it. This means they can know whether their code is functional
Storage	

• MySQL database holding 3 tables which will store data such as:
  ○ User information
  ○ Passwords
  ○ Progression
  ○ Challenges and challenge answers
  ○ Student attempt at challenges
  ○ Time students spend on each challenge
	1. A database would be required to store students' data safely and securely. Without the data that has been give it will not be possible to store the students progress or store the students accounts

Development requirements

Requirements	Justification
Processing 1. Handle database connection issues, when the database may be down or incorrect database credentials inputted 2. Handle incorrect credentials being inputted into input boxes, including malicious inputs such as SQL Injection Attack 3. Safely and securely write to and receive data from the databases 4. Hash passwords for first-time password inputs and log-in attempts to see if they match stored hashes 5. Be able to generate random, yet feasible variables for challenges to increase individuality 6. Be able to generate leaderboards with student data	1. The project cannot crash in the case of a database failure 2. The database should be as secure as possible to prevent data breaches of potentially private data 3. Connections to the database should be secure to avoid interceptions 4. In the case of a database leak, this prevents passwords being leaked 5. This makes it harder for the students to copy each other 6. This can create a sense of competitiveness, perhaps motivating the students
Developer 1. A Linux server with at least 10GB of storage, enough to handle processing, hosting and databases 2. A PC powerful (16GB RAM, 512GB SSD) enough to locally host test databases and web pages	1. There must be sufficient storage to hold the data about the students as well as the challenges and their attempts. 2. A powerful PC may be required to develop as many applications will need to be open concurrently, such as MySQLWorkbench, IntelliJ, Chrome, Firefox etc.
Student 1. Access to the internet with internet speeds of at least 5Mb/s, more than enough for the page to run smoothly	1. A sufficient internet connection will be required from the student to be able to communicate with the database and receive data, such as the feedback for the code they have submitted. 2. Students may prefer to code

2. Access to an IDE if they would prefer to write code in there and copy it in	within an IDE and then paste the written code into the website. The text box in the website will not have common IDE coding assist features, such as syntax highlighting and autocomplete.

Measurable success criteria

Criteria to be met with justification	How it will be measured

A website that can be used in the most common internet browsers, devices and operating systems without any problems or decreased speed.
 	
This would be required as not all students will be using the exact same software that I am testing the program on, such as chrome.
	• Open and explore the website on different internet browsers such as:
	  ○ Chrome
	  ○ Firefox
	  ○ Internet Explorer
	  ○ Safari
	• Open the website on different devices, such as:
	  ○ Surface Pro
	  ○ Mobile Phone
	  ○ iPad
	  ○ Laptop
	  ○ Personal Computer
	• Open the website on the following operating systems
	  ○ Linux (Unix)
	  ○ Linux (Chrome)
	  ○ iOS
	  ○ macOS
	  ○ Windows 10/11

A solution that assists students in gaining the most amount of marks in the coding aspect of their course and final exam. The whole aim of the project is to boost students' coding skills, so this is essential.	• Analyse end-of-year exam results for students who used this application and students who didn't • Let the students sit a coding exam before and after using this solution
Securely hold the students' progress and login information in a database that is secure from breaches and leaks. The database may hold personal or sensitive information and therefore must be secure.	• Attempt the most common methods of attack on the solution • Hire white box and black box testers to attempt to penetrate my levels of security

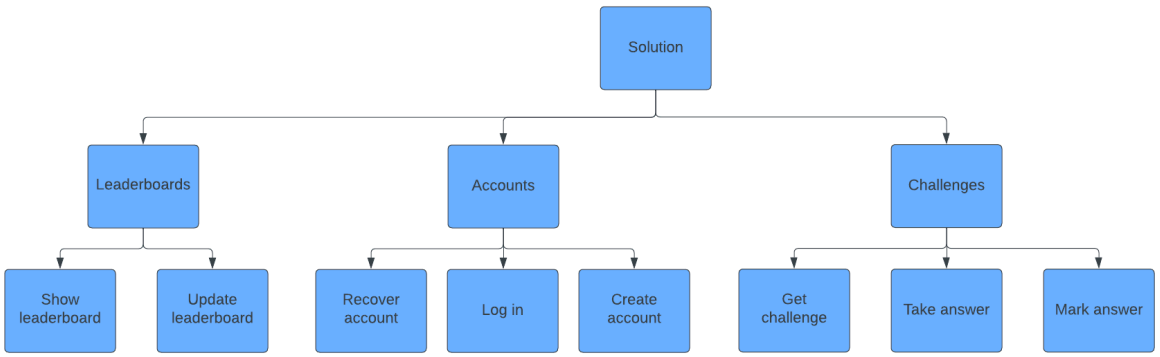
A secure teacher portal that allows teachers to view certain student's progress and manage their account. This could assist the teacher in tracking students progress with the challenges and perhaps even use it to set homework. It could also be used to reset students' passwords in the likely case that they forget it.	● Using a teacher account: ○ Attempt to initiate a password reset on a student ○ Attempt to select a student and see if you can retrieve their challenge data ○ Attempt to delete a student’s account and ensure that a 2FA prompt is received

Design

Design Overview

My project will mainly focus on building a server with a front-end website supported by a Java backend and MySQL database. The website will be written in HTML, JavaScript, and CSS. This website will present all of the user-interactable features, including a log-in system that will be stored within the MySQL database. These assist in the security of the users' progress as well as a way of identifying a user's progress for the leaderboard. A benefit of using a Java backend is that sensitive data does not need to be sent to the user's browser at any time meaning security can be maintained throughout the project and use of the solution.

Decomposition diagram



The above flow decomposition diagram shows how the whole problem can be broken down into smaller, easier-to-handle and solve problems. From this decomposition diagram, it is easy to look at the main tasks at hand and decide which ones to tackle. Because of this, it makes planning the development process of the project easier as I can see what the main parts of the project are going to be and get a sense of how long each section will take. Creating the decomposition diagram also allows me to create algorithms, such as the ones that are below, and increase the general robustness of my program as each algorithm will have been created before development, allowing time for testing.

Decomposing the problem down and down again allows me to create solutions that can be put together to satisfy each level in the diagram above and then keep moving up until the entire problem has been solved. Each problem listed above will require a database call. This defines in detail the structure of the solution to be developed.

Variable dictionary

Below is a table which contains the main variables of significations that will be implemented into the program. This identifies and justifies the key variables that are used in this solution.

Variable name	Type	Justification
uuid	UUID	An unchangeable UUID is assigned to a user when an account is created. It will be Version 4 UUID generated through `java.util.UUID.randomUUID()`
query	String	This variable is used multiple times throughout the program and stores the SQL query that is going to be sent to the MySQL server to execute. It is a variable to ensure that it is separate from the execution code and can be seen/edited quickly.
emailAddress	string	An email that optional weekly updates, progress and reminders are sent to
hashedPassword	string	This hash will be a PBDKF2 one-way hash stored in the database. Each time a password is entered it is hashed and compared to the stored hash
DATASOURCE	DataSource	This variable is of Datasource type. It is within the main class and is required to form a modular program where database access is

		separate from initialisation.
code	String	To ease the flow of the program, I have made the code from the student's attempt, which is stored in the database, a local variable to be used in the compiler class. This saves the program from repeatedly calling the database
user	String	Again, to assist a modular approach, it is best to have user as a variable that can be accessed and altered from anywhere in the file
attemptsStore	AttemptsStore	Java, being an object oriented language, requires attributes for an object. This will be an attribute for the ChallengePageForm class allowing for it only to be called once.
simpleJavaFileObject	JavaFileObject	Possibly one of the most convenient variables, this JavaFileObject will allow me to compile a student's code without the need of creating an extra file that becomes garbage after execution.

Data dictionary

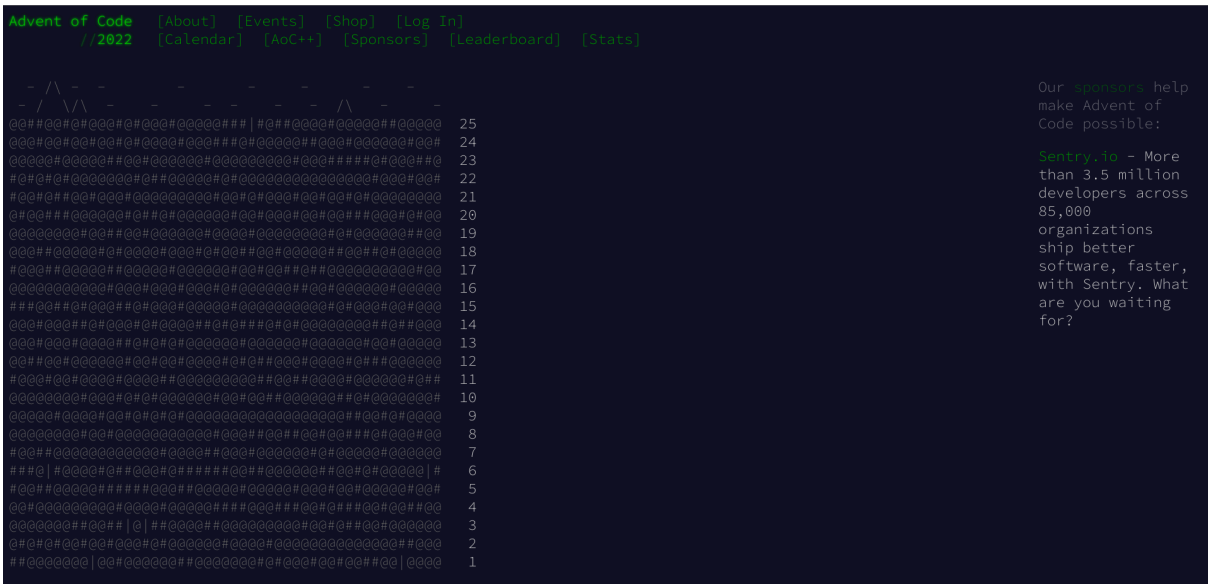
The following items are the key data items that will be stored within my MySQL database. Each data item will have a name, type and justification for its use.

Name	Database	Type	Justification
email	users	CHAR(128)	This is the email of the student. It is not used currently to communicate with the student but instead used as a unique identification feature.
password	users	TEXT	This is the string that the student uses to identify themselves allowing them to log into the system.
progress	users	INTEGER	This number will be the ID of the challenge that the student needs to complete next.
id	challenges	TEXT	Each challenge will have an ID that the system will use to identify and get a challenge.
title	challenges	TEXT	This will be the title of the challenge that is on the challenge page to the student.
instructions	challenges	TEXT	This will be an in-depth set of instructions on

			what is required from the student's code, including how it will be tested
solutionTemplate	challenges	TEXT	This will be the code that the student is initially presented with that they can then edit and add their own code to
email	attempts	CHAR(128)	This will be email of the student whose attempt is listed
id	attempts	INTEGER	This is the id of the challenge that has been attempted by the student
attempt	attempts	TEXT	This will be the block text of code that the student has inputted as an attempt to solve to challenge

User interface

The primary method of user interaction will be through a website serving HTML, Javascript and CSS. This addresses usability, ensuring that students are able to navigate the website. I plan to take a minimalistic approach to the website design, avoiding excessive animations and confusing pop-ups. My website will be similar to the current Advent Of Code website, as seen below, with respect to its colour scheme and fonts.

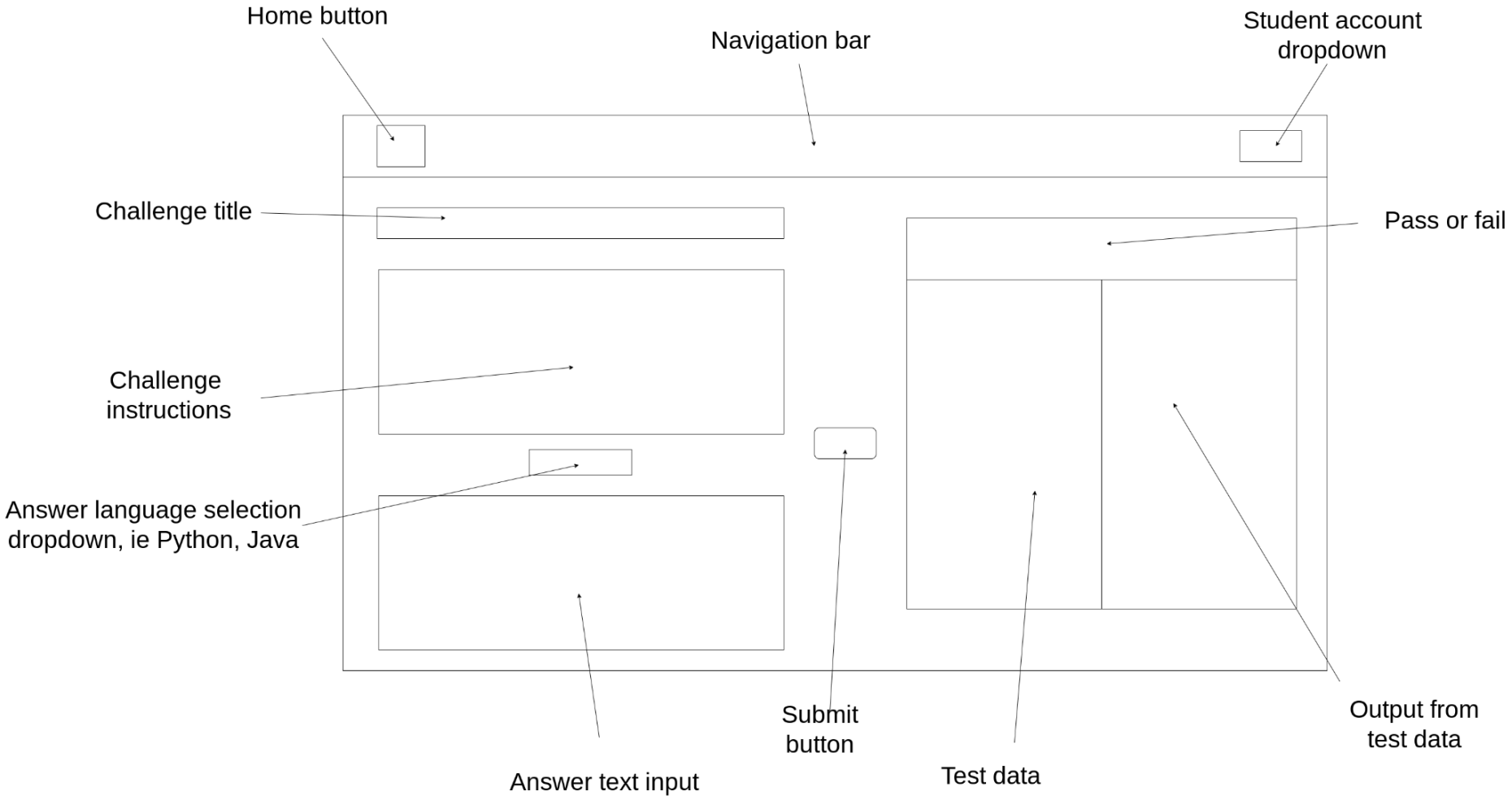


I aim to have a navigation bar that appears on every page on the site, this ensures that users cannot get “lost” on the website and are always able to redirect themselves to the home screen as well as being able to log out in one click. I don’t want the website to have complicated animations or confusing colour schemes simply for the reason that it does not need them. The website should be minimal and easy to use. The student doesn’t really care about how fancy the website looks because they are only there to submit code.

My user interface will be heavily reliant on my backend which could occasionally lead to issues due to the way that the Apache Wicket framework notifies developers of errors. As it does not have a console errors are thrown through the website, showing the raw error on the web browser. This could be catastrophic depending on what the error being displayed is. At best, it will interrupt the student’s session and possibly not save their data, and on the worst end, it could take down the host as a whole and leak answers to the student, although the latter should theoretically be impossible as they will be stored on a separate database.

Wireframes

Challenge page

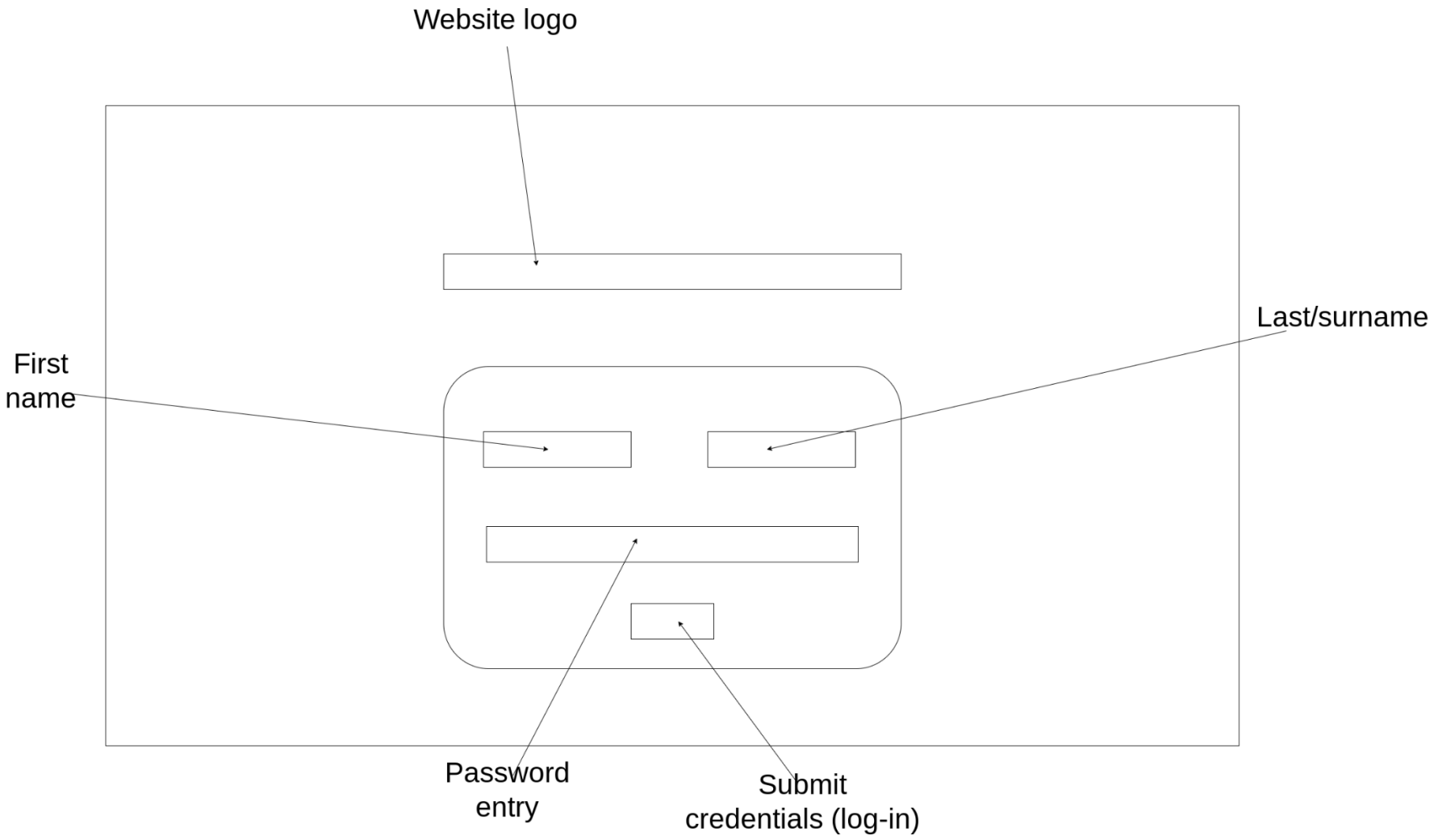


Clockwise justification

The following list justifies the usability features to be included in the solution.

- Navigation bar
    - Ensure the user can easily return to the homepage and access essential account details with ease
- Student account dropdown
    - This is where the students will be able to log in and out and potentially change their account settings
- Pass or fail
    - Allows the user to quickly see whether their code has outputted the correct data. This box will be colour coded, green (passed), red (failed)
- Output from test data
    - The data that was outputted from the test data is given here to show the student if their code is accurate
- Test data
    - This will be the data that is given to the students' program to test whether it meets the given criteria or not
- Submit button
    - Pretty self-explanatory. The button that is pressed to compile and submit their code to my program
- Answer text input
    - This is where the student will input their code for my program to compile and run tests on
- Answer language selection
    - The student may be able to select their preferred language to code in here.
- Challenge instruction
    - A brief yet sufficient explanation of the criteria that the code that the student produces must meet
- Challenge title
    - This title will be the "name" of the challenge. Having it separate allows the student to communicate with the teacher if they are stuck on the challenge
- Home button
    - A button that takes the student to my project homepage

Sign-in page

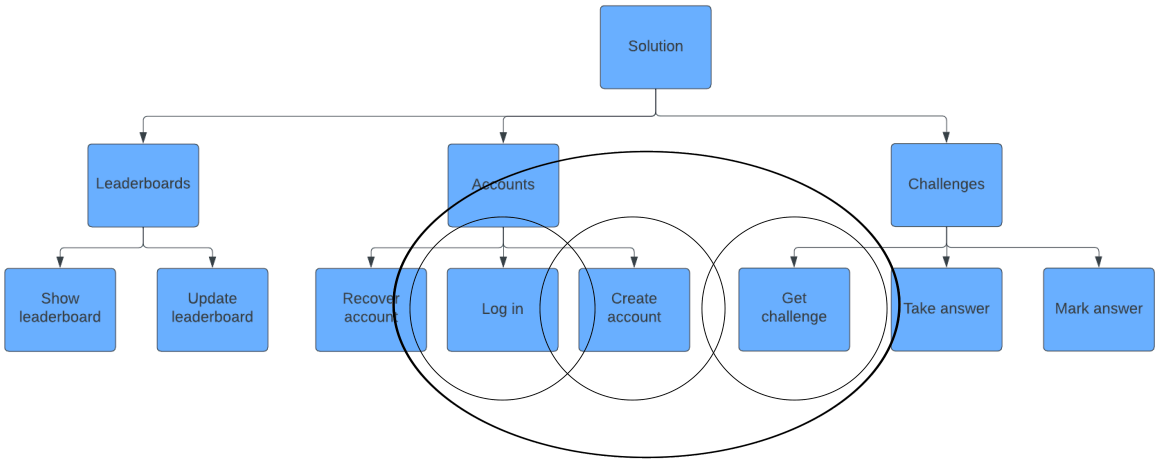

Website logo
First name
Last/surname
Password entry
Submit credentials (log-in)

Clockwise justification

- Website logo
    - This logo will help the student identify that they are in the correct place to sign in
- Surname
    - This will assist with security and identifying students who are on the leaderboard
- Submit
    - This will be a different coloured button to ensure they don’t miss it, but it's required to sign the user in
- Password entry
    - The text box that the user input their password in. It will preferably be blanked to ensure another layer of security
- First name
    - Similar to above, however, this name can also be used to address the student in personalised messages if need be

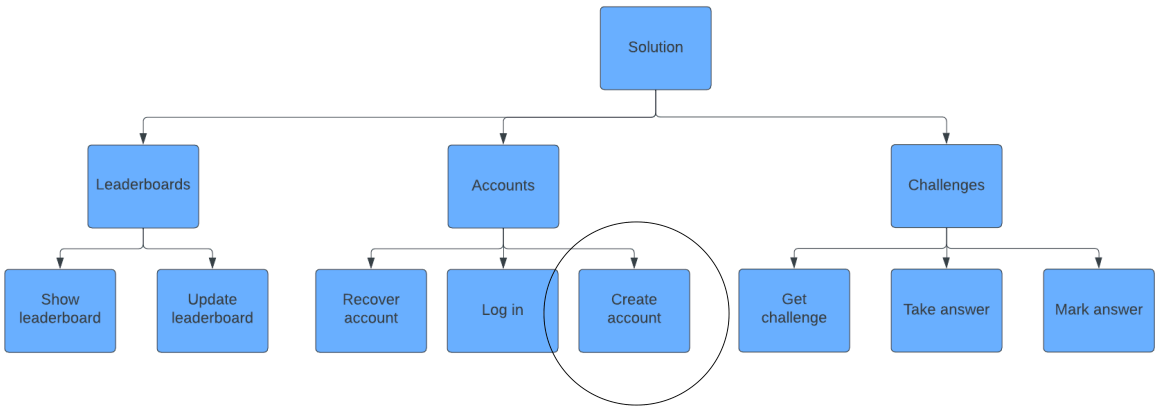
Algorithms

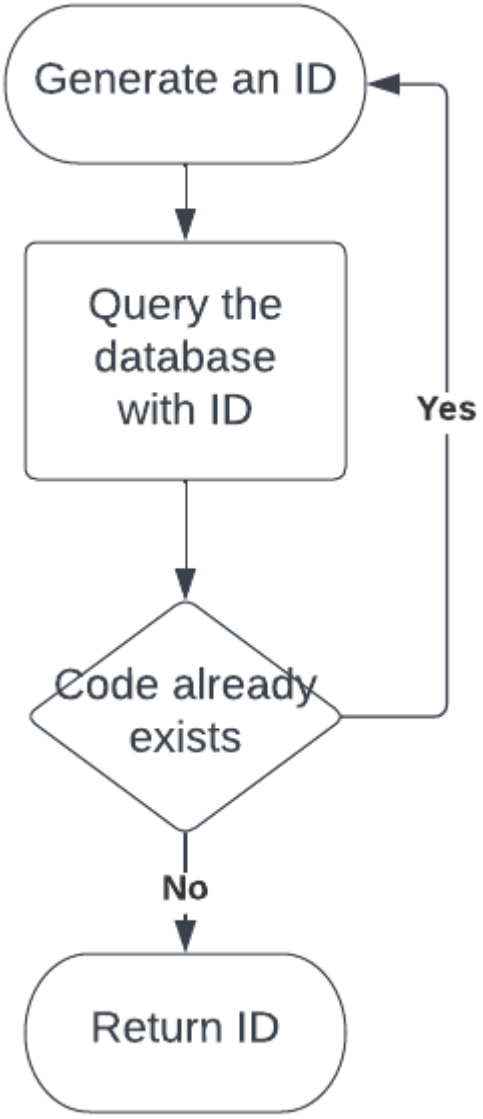
Below are a few algorithms that are key to the final working solution. All of the algorithms will fit together to form large algorithms. For example, the UUID generating, sign and sign-up algorithms will all fit together to form the account flow algorithm, which will incorporate many different algorithms. This is the case for all algorithms that are being produced. The reason for this algorithmic breakdown approach is to ease the need for on-the-go algorithm generation and increase the general robustness and effectiveness of the program, as well as reduce in-code writing times.



Generating a UUID

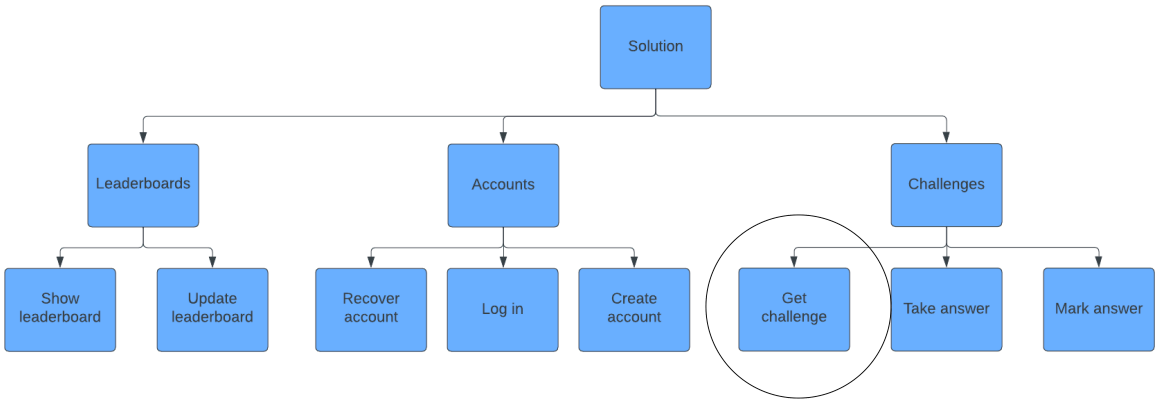
This algorithm is required just to ensure that a UUID generated by Java's UUID randomisers does not already exist on the database. Although there is little to no chance of this happening if it were to happen the chances would be it would break lots of functionality therefore I have ensured it cannot happen.

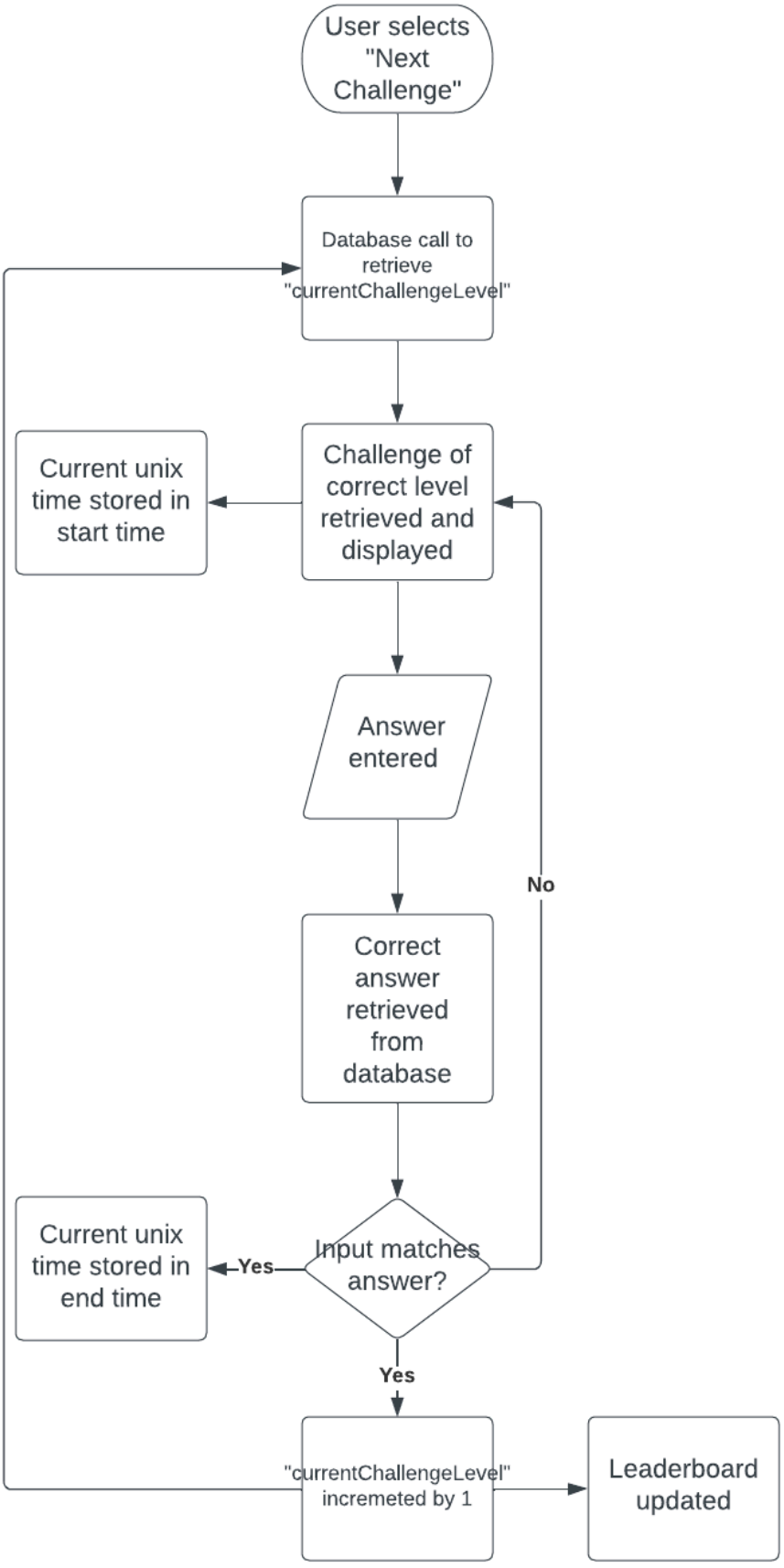



Generate an ID
Query the database with ID
Code already exists
Yes
No
Return ID

A student selecting a new challenge

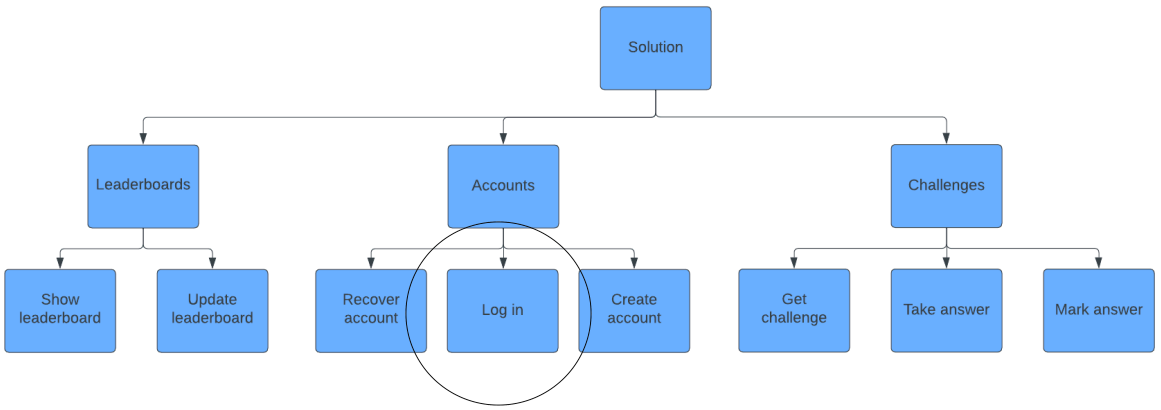
This algorithm will be implemented to present the user with the correct challenge when they press a button to trigger a challenge to appear. At the start and end of each challenge, the current time will be stored in the database, this data will be used to calculate the time taken for a student to complete a challenge for use on their account statistics and leaderboards.

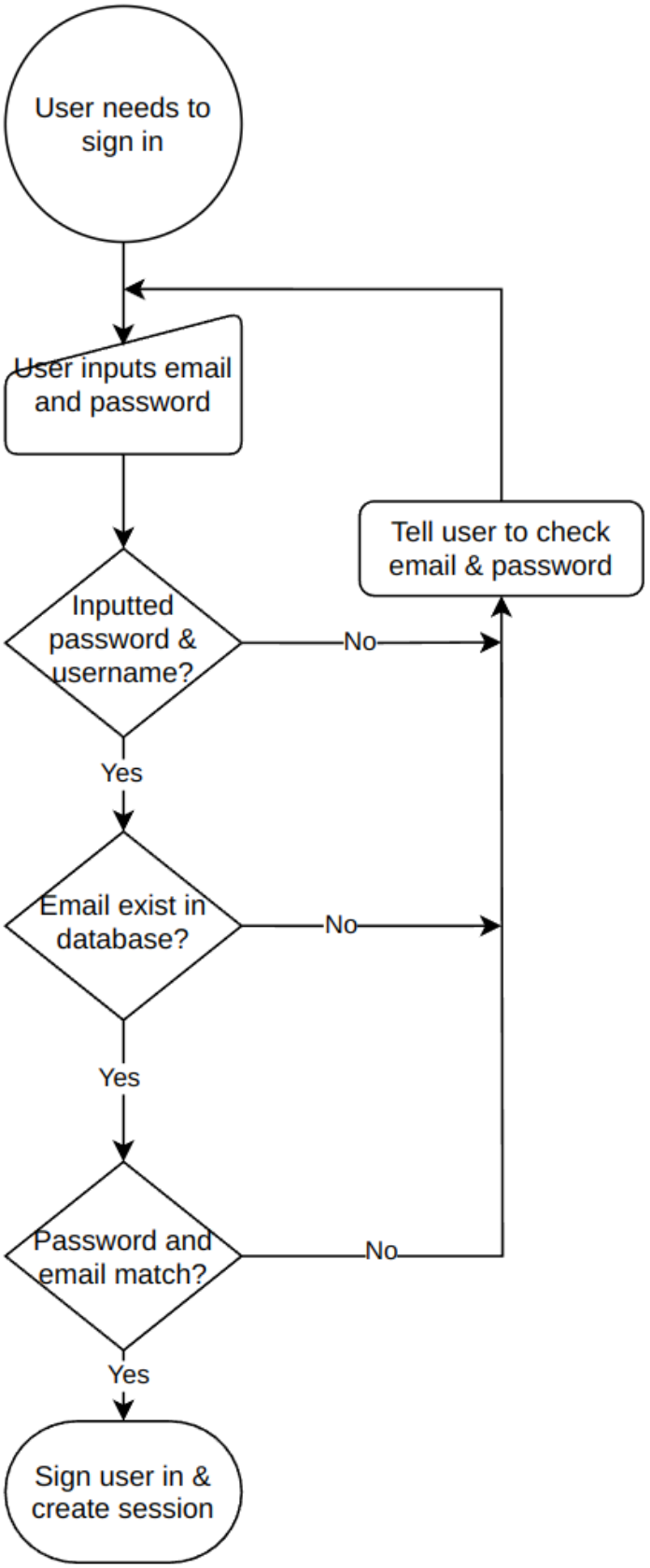



User selects "Next Challenge"
Database call to retrieve "currentChallengeLevel"
Current unix time stored in start time
Challenge of correct level retrieved and displayed
Answer entered
Correct answer retrieved from database
Current unix time stored in end time
Yes
Input matches answer?
No
Yes
"currentChallengeLevel" incremeted by 1
Leaderboard updated

User signing in

This algorithm is a key algorithm and is accessed by every student and user of the website, every time they attempt to access challenges with an expired or new session. As a security-related algorithm, it is vital that this algorithm is robust and has no flaws. Each section of this algorithm has been tried and tested to ensure maximum security. Potential vulnerabilities within this algorithm include the fact that it is calling an external database which may lead to security weaknesses, especially if the database has not been set up correctly, or if common hacking methods have not been addressed, such as SQL Injection.

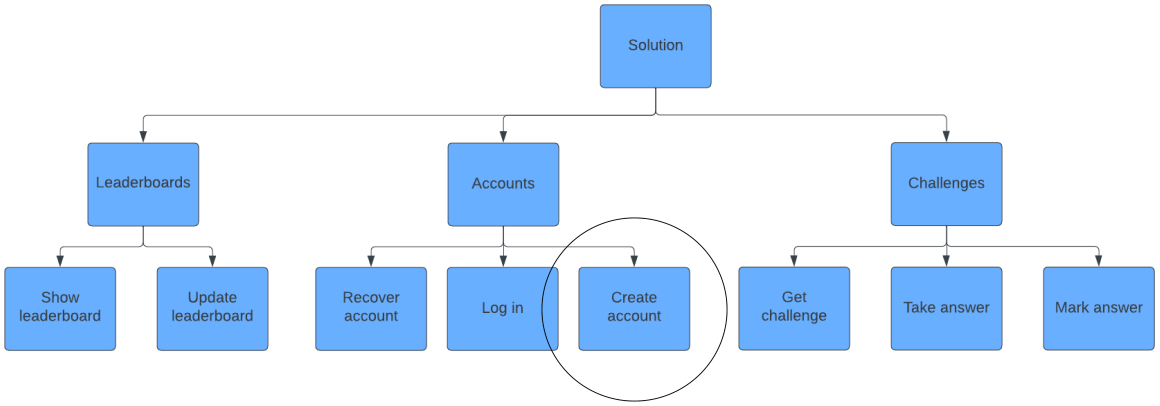


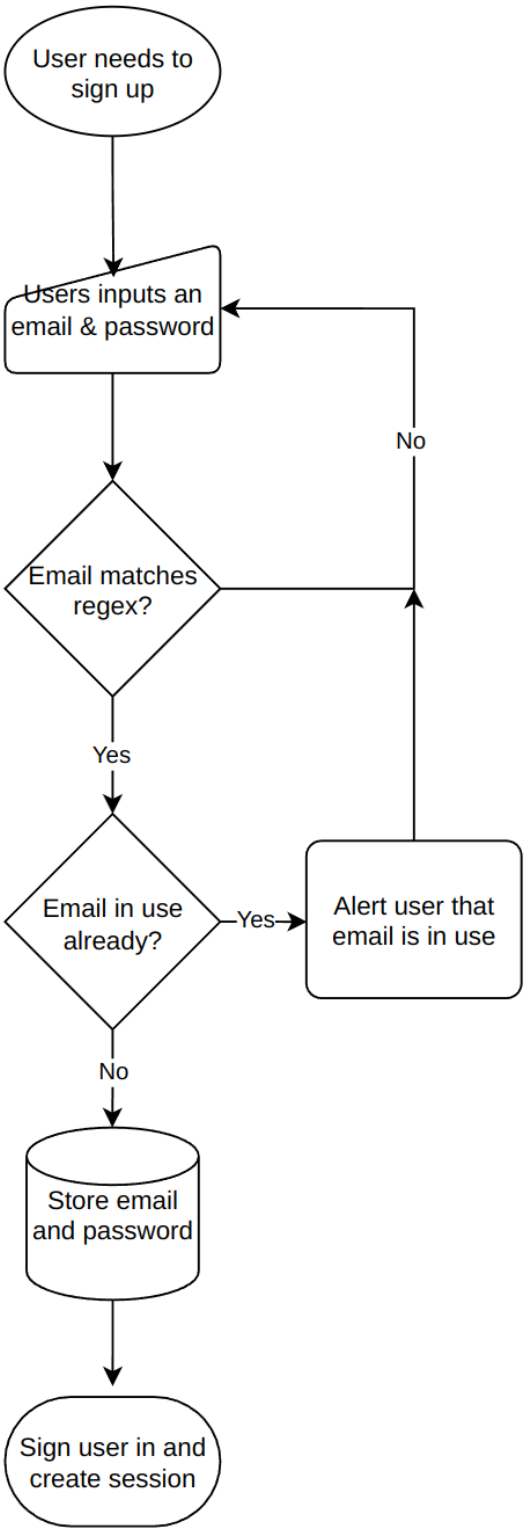

User needs to sign in
User inputs email and password
Inputted password & username?
No
Tell user to check email & password
Yes
Email exist in database?
No
Yes
Password and email match?
No
Yes
Sign user in & create session

User sign-up

This algorithm is similar to the login algorithm, however, requires a few more checks than the login algorithm. Due to the nature of my database, duplicate accounts could cause some serious problems in the project, and there is no way to currently remove duplicate accounts, so it is down to this algorithm to ensure that duplicate accounts cannot be generated.

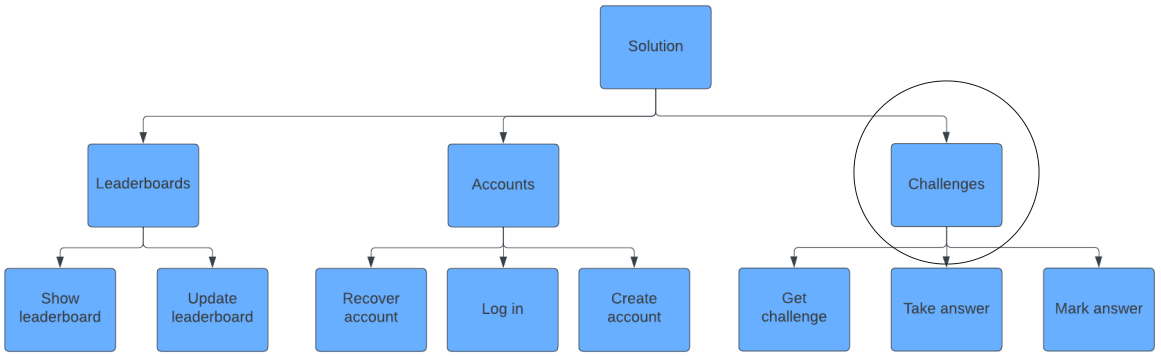
This algorithm requires validation as students are inputting an email address to use as the primary key for the user's table. The email address they are using could also be used for future communications so it is key that it is correct and the algorithm must ensure that the email address exists before adding the account to the database.

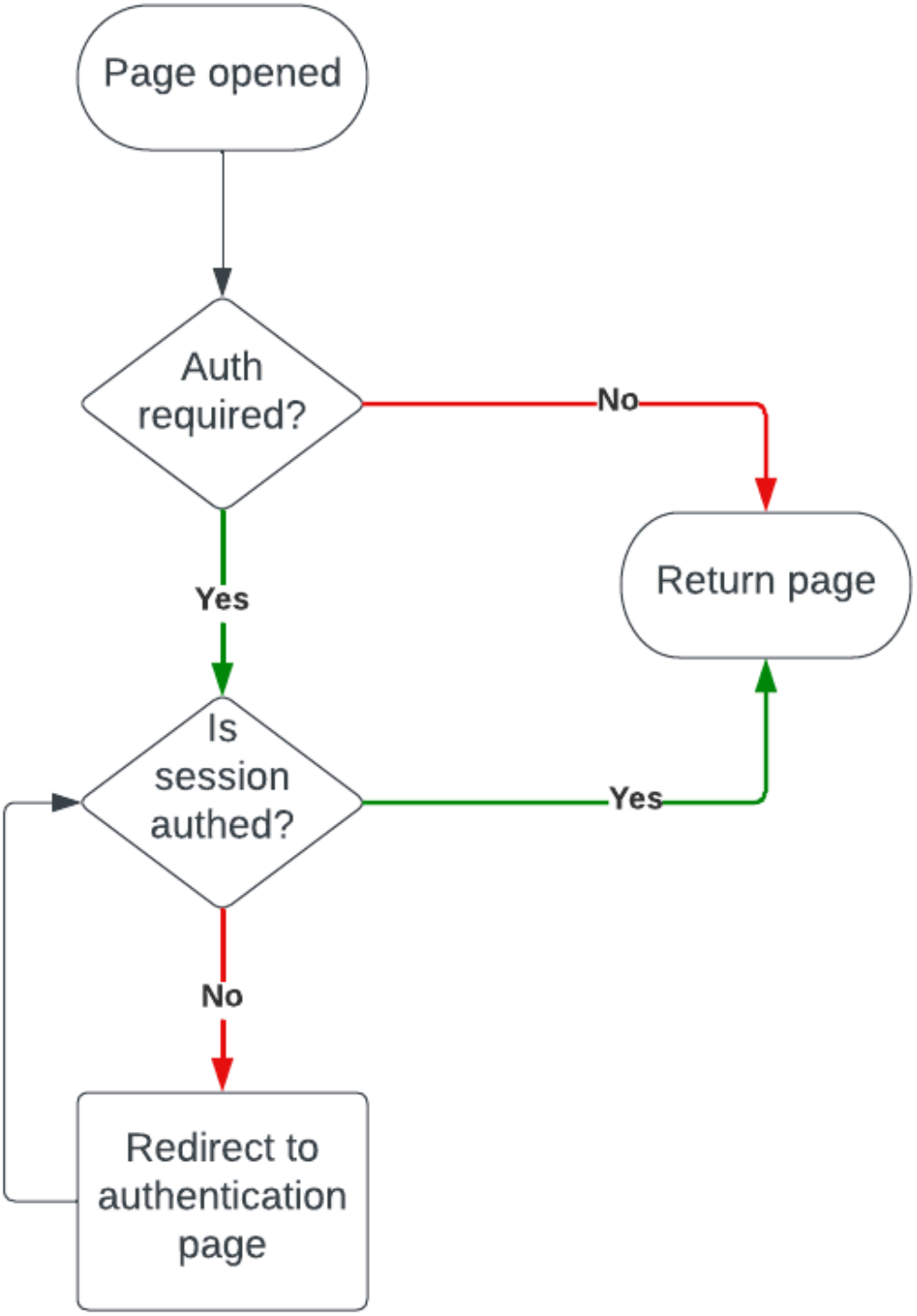



User needs to sign up
Users inputs an email & password
Email matches regex?
No
Yes
Email in use already?
Yes
Alert user that email is in use
No
Store email and password
Sign user in and create session

Authorisation strategy

A slightly simpler algorithm compared to the rest, this authorisation strategy algorithm links in with both the sign-in and log-in algorithms as they are only used when the authorisation deems it necessary. The point of this algorithm is to ensure that a user has an authenticated session on pages where it is required, for example, the challenge page. Without this, it would be difficult to track students' progress as it would not be possible to tell who is submitting the code.



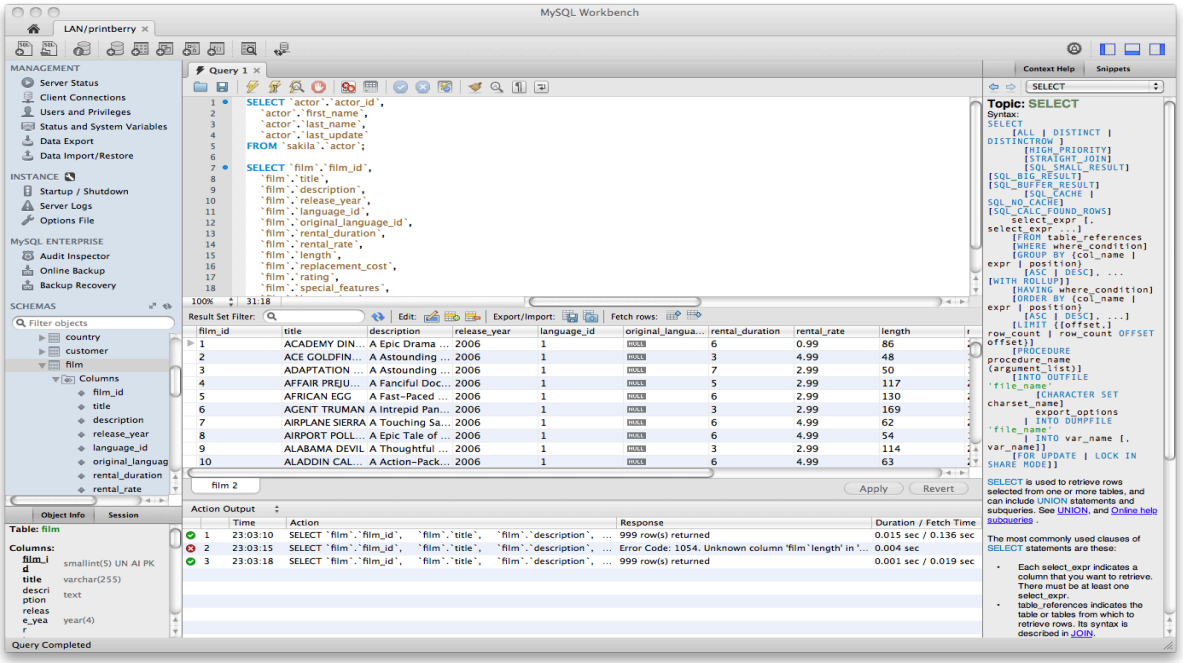

Page opened
Auth required?
No
Return page
Yes
Is session authed?
Yes
No
Redirect to authentication page

Databases

I plan to use a MySQL database to hold the data required for my project to run. I have previous experience setting up and hosting databases using MySQL and some knowledge using SQL. The main tool I will be using to access the database will be MySQL Workbench. This is an official tool released from MySQL and can be found here.

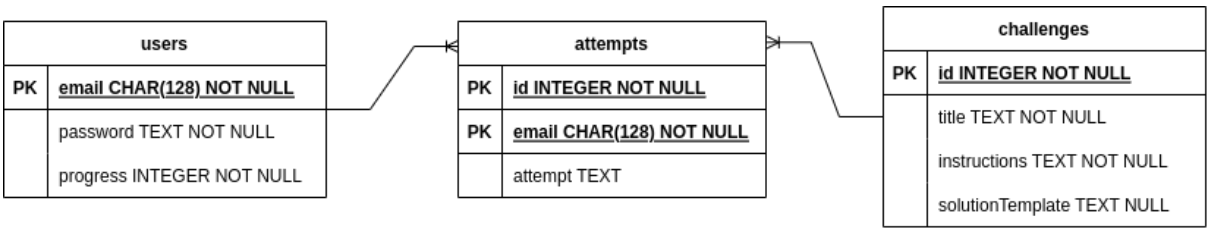
https://www.mysql.com/products/workbench/

The workbench is compatible with localhost database servers, externally hosted servers and SSH servers. This is what it looks like when using it:



The design of my database system will be simple, and follows rules for Third Normalised Form. There will be three tables, one for users, one for challenges and another for attempts. The users table will be where all the users log-in info will be stored including their username, password and current level (progress) that the student is on. The challenges table will be a read-only table and contain the information for every challenge, including the title, the challenge instructions and the challenge ID. The final table will be the attempts table and it will store the latest attempt that a student has completed, along with their email address and the ID of the challenge (these two form a composite key). See ERD below.

Entity Relationship Diagram



Validation

Need for validation

Validation for inputs is required in many parts of my solution. Different types of validation will be required for different parts of my solution. All inputs that make contact with the database will need validation to ensure there are no compromising security flaws that allow users to run SQL instructions through input boxes. Each layer of my solution will validate data passed to it by higher levels, and return from lower levels.

Examples of where validation will be required

- Sign Up - validation will be required in the email input box to ensure the given email exists. This could be done through a simple character check to check if the string contains an "'@" symbol. Although this method checks that the input is an email address, it does not check to see whether it is valid. A common way of ensuring an email is valid is to send the given email a "verification" email with a one-time use code or even a link that verifies the email is valid.
- User inputs - all data provided from the user to the backend must be checked carefully. For example, empty strings, and huge strings such as a string size above 10kB, which is the maximum that would be expected for the user to enter.
- Log-in system - when logging in, users will be asked for their first and last names, I will be validating that by ensuring there are no numbers, spaces or special characters in the input as first names do not contain characters apart from letters and occasionally a "-" if they are double-barreled
- Database system - the database system will validate all data before it is committed to the database to ensure there are no empty strings or incorrect variable values being written, such as negative numbers where only positive numbers should be added.

Alpha test plan

To avoid unwanted errors occurring that may be hard to fix down the line, after each stage of the development I am to test my solution with the following data. The test data I will use will consist of valid and invalid data. Some of this data will be required to test the items listed for validation in the previous section. This ensures that I am testing that my solution is working how it should be and preventing data that could compromise the effectiveness or security of the system.

Plan for debugging errors

Initially, I will debug the code by using standard Java logging techniques, such as the Java logger to output key values as the program executes, as well as write unit tests to ensure the code is working as intended. If the solution to the error is obvious I will attempt to fix it myself. If I still cannot find a solution to the error I will search on stackoverflow.com or w3schools for a solution.

Alpha test data

For the alpha testing data I will be using known good data to ensure that each level of the system is working correctly. Each of the inputs in the following alpha test categories should allow the forward flow of the system and not cause any errors.

I plan to leave the robustness testing by which I use inputs that shouldn't work to the beta test.

Sign in/up system

There will be one preconfigured student account that I will use whilst developing the system

Login	foo
Password	bar

User inputs

For the text box in the challenge page, there will be one class that I copy and paste into the box and log that it is received by the backend web server.

```
public String upperCaseConverter(String upperCaseString) {
  return upperCaseString.toLowerCase();
}
```

Database

I will use MySQL workbench or MySQL to ensure that the data was successfully written to the database. I will do this by checking the tables in the database and comparing it to the data that was inputted.

Beta test plan

Unlike my Alpha test plan, my Beta test plan will not be run periodically but instead, be a brutal set of data designed to test every part of my solution once it is complete. It aims to test every feature and find any issues or vulnerabilities before it is distributed as a release to the customer. I have created the set of data ensuring that it tests all validation within my program and it aims to find errors and security vulnerabilities

For beta testing, we are showing that the system is fully functional. This will involve testing by means of unit tests:

- Adding new users
- Taking more than one question
- Successfully compiling and reporting the status back to the user
- Progression onto the next challenge

I will separate my beta tests into two generic categories, robustness tests and functionality tests. Robustness tests will be highlighted blue, and functionality tests will be highlighted green.

Beta test plan data

The following data will be used to test whether new users can be added whilst concurrently testing all the levels of validation within the system.

Data	Where it will be used	Expected output	Justification
Test123!	String inputs	Invalid first/last name	This string contains, characters, ints and other characters which will test my input validation to ensure that first and last names can only contain letters and no

			other characters
Test, @, test@, @gmail.com, test@com	Email inputs	Invalid email address	The string contains similar formats to what a real email address would be, however, none of them can be valid and it is therefore testing my email regex validation
“null”	Any input boxes	Stored in database as a string	Any inputs into the database should not result in a null being inputted into the database as this would result in errors, i.e if they try to log in again
Qwerty OR 1=1	Any input boxes	Does not allow SQL injection	This is a commonly used string to test against SQL Injection. If inputs are unsanitized it will read the two as separate queries and as 1 does equal 1 it will allow false entry
“Test email@gmail.com”	Email inputs	Invalid email address	An email should not contain a space.
tom.swainston@hppc.co.uk	Email inputs	Valid email	This email is a valid email address and should therefore be allowed by the system

tom.swainston@hppc.co.uk	Password input	Valid password	Although not recommended, this password should be allowed as it contains secure characters

Challenge input box

Following on from the Alpha test, the same data will be used, however, this data will be compiled and the student will be notified whether it was successful or not.

Having successfully signed in, the first challenge displayed on the challenge page will accept the following data values.

This challenge requires the student to take some supplied uppercase text and return it as lowercase. For example, given the text “TEST”, the student’s code should return “test”.

An example of code that will work is as follows.

```
public String upperCaseConverter(String upperCaseString) {
  return upperCaseString.toLowercase();
}
```

Once this data has been inputted and submitted into the text box it should report back to the user two things. The first should be whether the student’s code has successfully compiled, which in this case it will, and whether it has passed the unit tests that the code undergoes to check it passes the challenge.

If the student's code successfully compiles and passes all the tests, the program should move on and display the next challenge to the student. It should also update the database with the code that they submitted that worked, as well as increment the integer that tracks their progress.

Further tests that I will run on the challenge input box are seen below.

Challenge input box data

Test	Expected result	Justification
Remove package	The system will identify the fact that there is no package and return the error to the student.	Changing the package name for the method will invalidate the class when it is being accessed from the class within the backend system.
Remove or change the class access modifier	The system will not crash and return the error to the student.	Removing or changing the access modifier from public to private will make the class unreadable once it has compiled.
Remove or change the method access modifier	The system will not be able to compile and return the error to the student	Removing or changing the access modifier from public to private will mean it no longer implements the test interface
Add invalid imports	The system will not import any libraries and return the error to the student	If the students attempts to add invalid imports from external sources they should be prevented as malicious attacks could happen if any imports were allowed
Change the method name	The system will not crash and return the error to the student.	Changing method name for the subroutine will mean that when the code attempts to run tests or call the students code it will not be able to find the block of code
Change the parameter type	The system will not crash and return the error to the student.	If the parameter type were to be changed, it would mean that when the tests are run on the class and arguments are given they will be of the wrong type
Completely empty the text	The system will not crash	Emptying the input box

input box	and return the error to the student.	will test to check whether the compiler will attempt to compile an empty string
Enter random text	The system will not crash and return the error to the student.	Without any Java, the compiler will fail to compile random string and characters
Attempt System.exit()	The back-end system will not shut down.	As the students code will be running locally, if this were to be executed it could potentially shut the entire back-end server off
Change the method to return the wrong type	The system will not crash and return the error to the student.	As the tests will be calling the students method it will be expecting a return type that it is compatible with

Development

Todo-update code

Prototypes

The following sections show prototypes for each stage of the iterative development process.

Prototype #1 - User interface systems

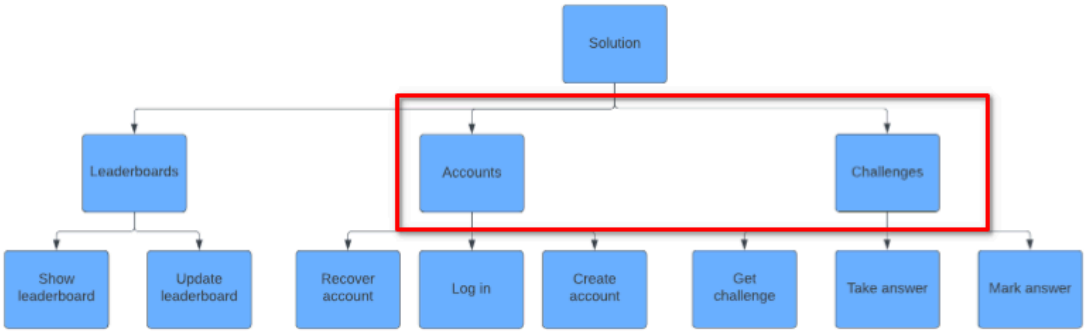
Prototype number one I will focus on the user interface of the project which consists wholly of webpages. I decided it would be easiest to develop the user interface first as it would allow me to have a foundation for testing future code.

I will follow each wireframe carefully and produce an HTML file for each page with a single CSS file being used by each webpage. I hadn't initially planned on doing this but after a while, I realised that the stylesheets for each web page were almost identical. By using only one stylesheet it allows for much more consistency throughout the website as well as saving me time as I did not need to produce three separate stylesheets.

I started with the homepage, the challenge page and finally the sign-in and sign-up pages which in total took me around 3 days to complete. I assigned a day to each page which allowed me ample time to set up my environment, code the page, check that it matched the wireframes as closely as possible, then finally refine the page.

Once I had each page completed, and compared to all the wireframes, I decided it would be a good time for some valuable stakeholder feedback, as after all, this is what the students will be seeing for the whole time using my solution.

Decomposition analysis



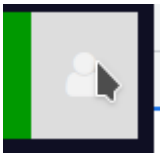
It is difficult to completely pinpoint which section of the decomposition I am tackling within this prototype. This is because what I am tackling within this prototype is a part of many sections on the decomposition diagram. One may think that this could hint that my decomposition diagram could be incorrect, but I think if the diagram were to decompose the project into “front-end” and “back-end” etc it would simply be too vague making it difficult to follow.

I think the most appropriate section to highlight that is being tackled within this prototype is the accounts challenges sub-decomposition. This is because the front-end section of the project will be holding most of the accounts and challenges part that the student will be seeing. The following is evidence of the iterative development process relating to the decomposition of the problem.

Navigation bar

The first part of the page that I created was the navigation bar. I carefully used the wireframe given for the homepage to create the correct content, including the home button, the sign-out button and the settings button. My navigation bar will be displayed on all pages of my website. The buttons on the navigation bar will stay the same, apart from the sign-in/sign-out button which will change accordingly. The result was the following

This is what an icon looks like when the mouse is hovered over it.



Stakeholder feedback

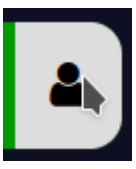
To validate the key elements of my solution, I sought feedback from my stakeholders once the first few items had been developed. Upon discussion with one of my stakeholders, they said that they liked the basic theme making it very user-friendly, but did not like the sharp edges nor the white on white when hovering.

To change these two features, I implemented `border-radius` under the navbar and `color` under the `navbar a:hover` in my style.css file, as seen below

```
7   .navbar {
8       overflow: hidden;
9       background-color: #009900;
10      border-radius: 15px;
11  }
```

```
22  .navbar a:hover {
23      color: black;
24      background-color: white;
```

To produce the following.



Homepage

Having decided that my project be based on Advent Of Code, I decided to borrow their colour scheme and font, by inspecting their website and looking into their CSS files. I set the background of all my pages to #0f0f23 and my font to '"Source Code Pro", monospace'. I set these both within my style.css file meaning that all pages would have the same background, font and text colour.

Following the Christmas theme, I also set the page icon to a Christmas tree through the icon attribute in HTML.

As this page is the first page that a student is met with when the website is accessed, I have added a basic descriptive paragraph of what this project is about. This block of text has been sectioned off under the class "main", this allows me to centre, and adjust if I want, all the text at once. The written text was mostly to test out that my font sizing was correct, which it was as I used header tags. The final line will be a link to either the sign-in page or the challenge page depending on whether the student has signed in yet.

Examples of code and the first homepage are as follows.

```
body {
    background: #0f0f23;
    font-family: "Source Code Pro", monospace;
    color: white;
}
```

```
<link rel="icon" href="xmas_tree.png">
```

```
<div class="main">
    <h1>GCSE Advent Of Code</h1>
    <h3>An A Level project developed by Tom Swainston</h3>
    <p>Within this website are a set of mixed difficulty coding challenges that
        cover all aspects of the coding side
        of your OCR GCSE computer science qualification. Although there are no
        specific tests or projects to
        examine your coding skills, there are questions that may appear on your
        paper that require basic coding
        knowledge to be able to answer and gain full marks.</p> <br/>
    <p><a wicket:id="challengePageLink">Go to the challenges</a></p>
</div>
```


GCSE AOC!
http://localhost:8080/?0
Incognito
Other bookmarks
GCSE Advent Of Code
An A Level project developed by Tom Swainston
Within this website are a set of mixed difficulty coding challenges that cover all aspects of the coding side of your OCR GCSE computer science qualification. Although there are no speccific tests or projects to examine your coding skills, there are questions that may appear on your paper that require basic coding knowledge to be able to answer and gain full marks.
Go to the challenges

Sign-up and sign-in page

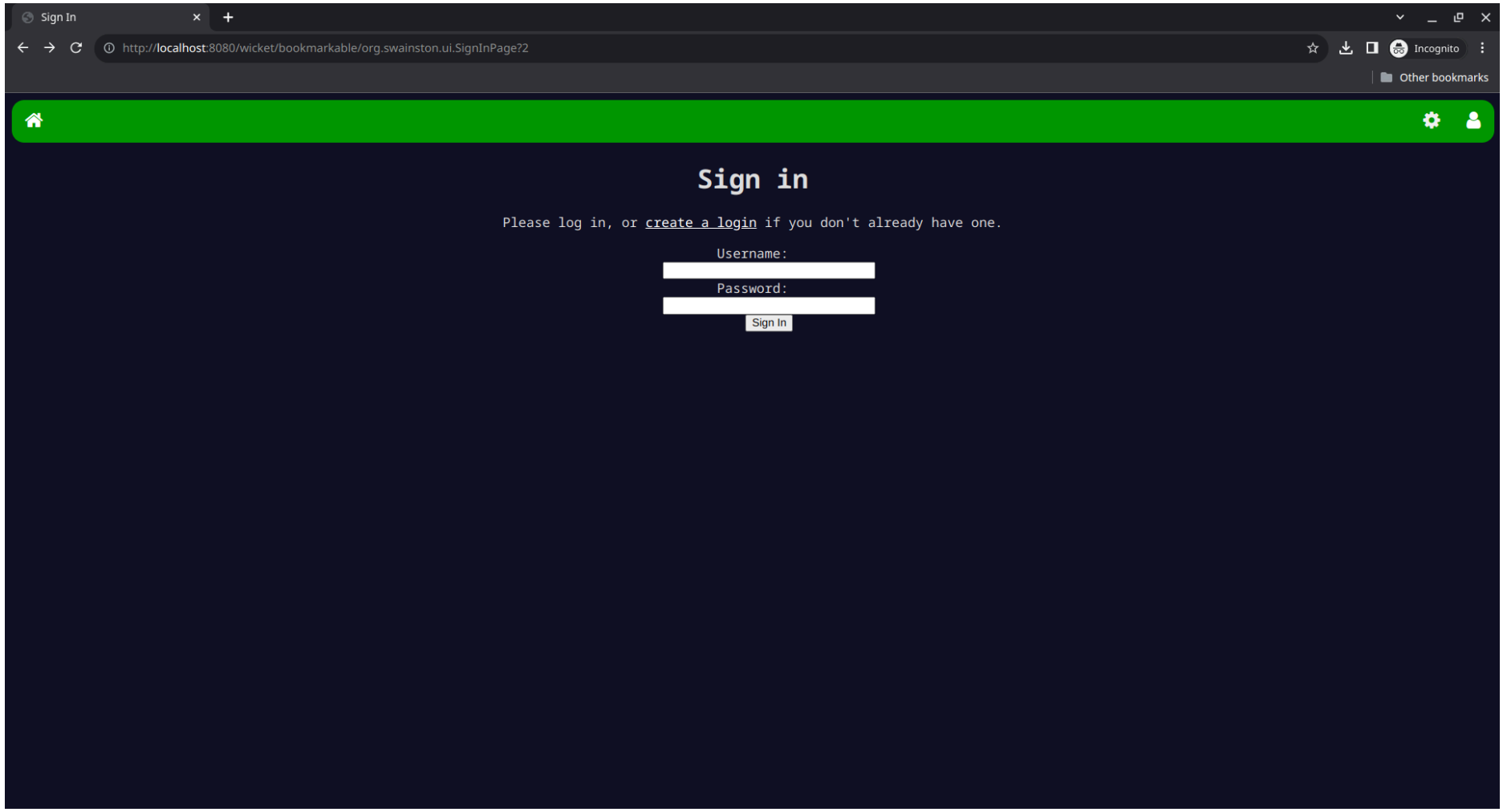
I decided not to include separate sections for sign-up and sign-in as they are highly similar and there are only small differences between each page.

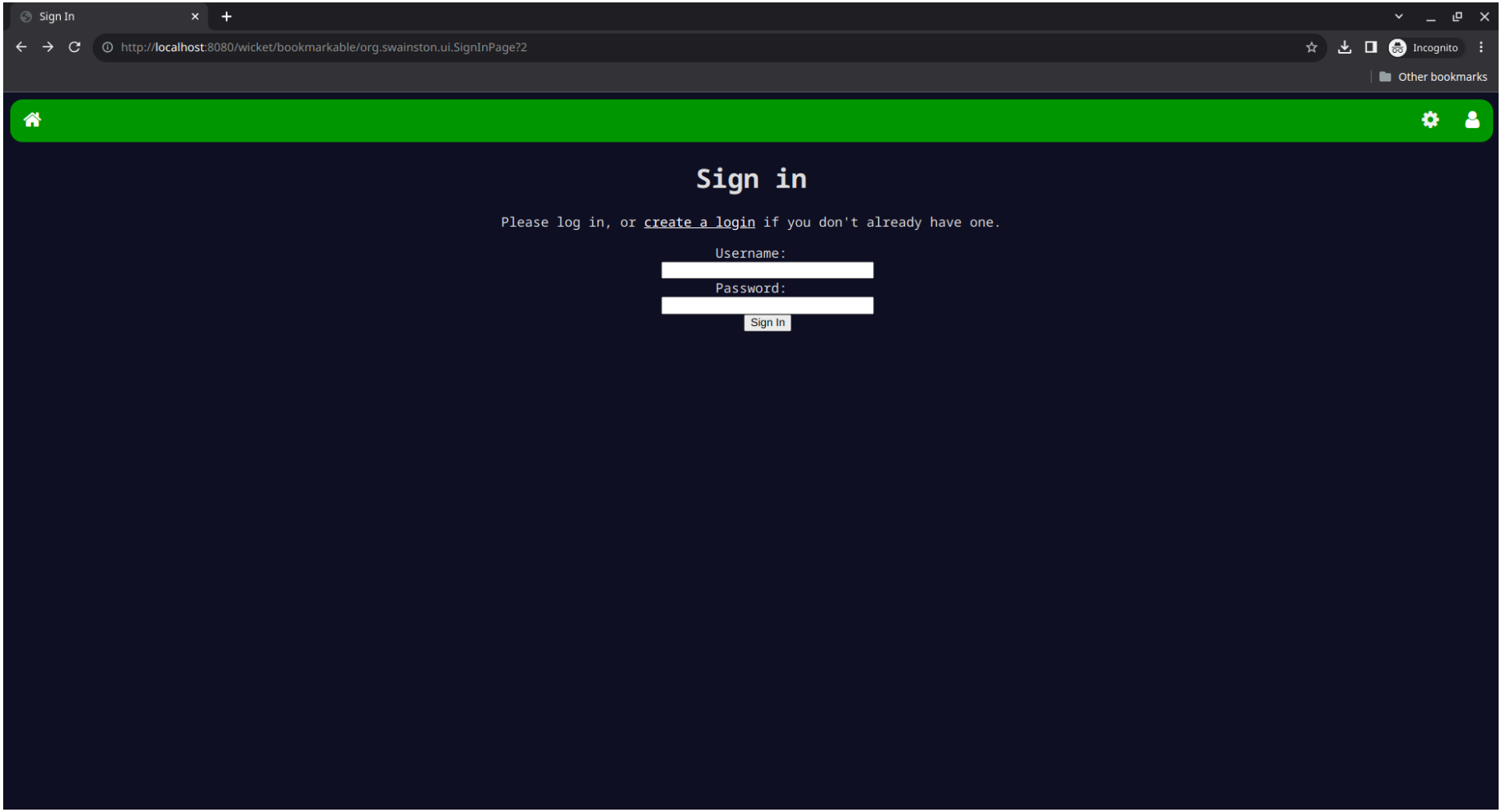
A lot of the content within the page was influenced by examples of the Apache Wicket website, which contained examples of the most common pages. These examples can be seen here:

https://examples7x.wicket.apache.org/authentication1/wicket/bookmarkable/org.apache.wicket.examples.authentication1.SignIn?1

Both the sign-up and sign-in pages contained a title and a small description of the page, although most students would have this part explained or pick up on what this meant pretty quickly. The pages also contain an email and a password input box which currently do not contain any CSS as there is a small chance that further boxes will be required.

The student will first be presented with the sign-in page in any instance, even if the student doesn't have a login as my system has no way of knowing if they have an account. To combat this my sign-in page contains a link that will redirect the user to the sign-up page to create the account


Sign In
http://localhost:8080/wicket/bookmarkable/org.swainston.ui.SignInPage?2
Incognito
Other bookmarks
Sign in
Please log in, or create a login if you don't already have one.
Username:
Password:
Sign In


Sign In
http://localhost:8080/wicket/bookmarkable/org.swainston.ui.SignInPage?2
Incognito
Other bookmarks
Sign in
Please log in, or create a login if you don't already have one.
Username:
Password:
Sign In

Challenge page

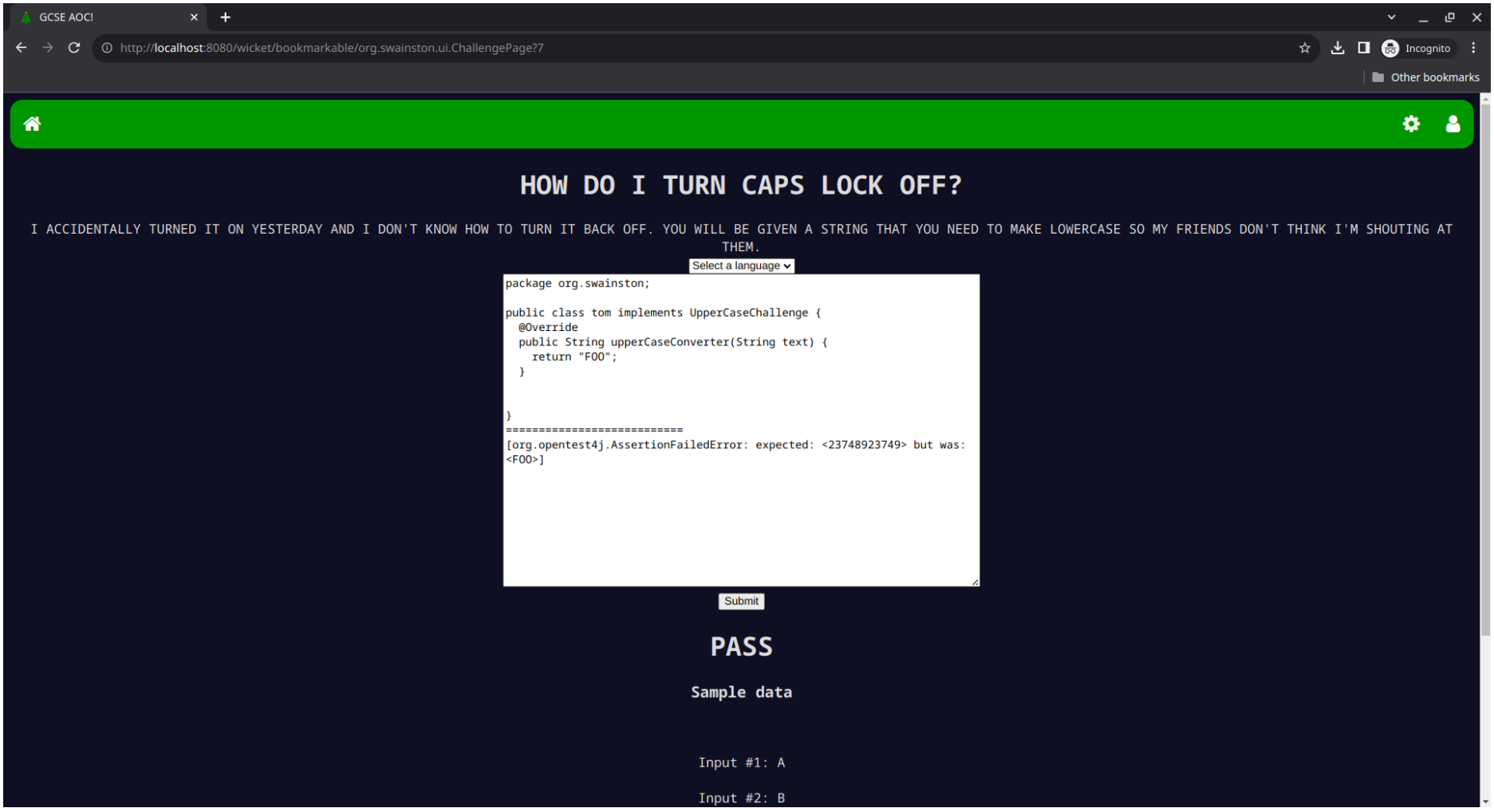
The challenge page initially followed the wireframe given during design, however it then veered slightly off the plan due to the way that the future development of the backend started coming through.

The page consists of the most items on the website and these include, an h1 heading, which will be the title of the challenge that the student is attempting, an h2 description, which describes the details of the challenge and what is generally expected from the student, a dropdown menu that allows the student to select which language they would like to code their answer in, a basic HTML input box that has some pre-written or outlined code for them to fill in and complete, a basic HTML submit button, then the feedback section of their code. The feedback section contains a pass or fail, which is again a h1 heading, then the inputs that have been used to test their code, the outcome of their code and what is expected of the code.

The hardest part of this page was creating the different sections of the page that allowed me to give it the box-like feel that it initially was wireframed out as. After some research and tutorials from https://w3schools.com, I decided that container classes would be the best approach.

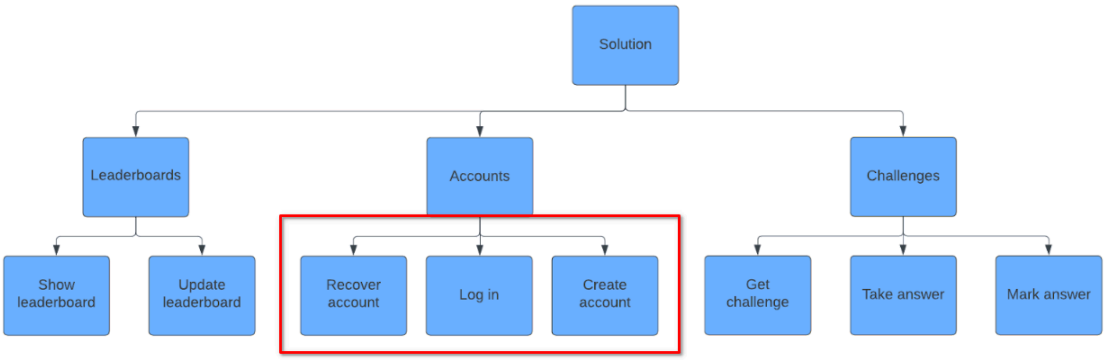
I divided each of my pages up using the tag and then was able to move each container about meaning there was less to code and less confusing maths required to create the desired web page according to the wireframe.

An issue that arised when coding this webpage was the fact that it would require the page to be updated uniquely for each student on each attempt. Although this wouldn’t


GCSE AOC!
http://localhost:8080/wicket/bookmarkable/org.swainston.ui.ChallengePage?7
Incognito
Other bookmarks
HOW DO I TURN CAPS LOCK OFF?
I ACCIDENTALLY TURNED IT ON YESTERDAY AND I DON'T KNOW HOW TO TURN IT BACK OFF. YOU WILL BE GIVEN A STRING THAT YOU NEED TO MAKE LOWERCASE SO MY FRIENDS DON'T THINK I'M SHOUTING AT THEM.
Select a language
package org.swainston;
public class tom implements UpperCaseChallenge {
@Override
public String upperCaseConverter(String text) {
return "FOO";
}
}
==========
[org.opentest4j.AssertionFailedError: expected: <23748923749> but was: <FOO>]
Submit
PASS
Sample data
Input #1: A
Input #2: B

Prototype #2 - Backend systems

This prototype is the development of the backend system, excluding the database and the game engine.



The first part of this prototype will be installing the backend systems required to host and develop the project. This should not take too long although I must be able to fully understand what is happening for the system to be online, especially as a large portion of code is written by Apache Wicket developers and imported when the system is installed. To be able to effectively debug and alter settings a knowledge of what is going on is key.

The second part will be the scripting side of the web pages, excluding the challenge page.This is because this page requires a lot of research and more code than the rest of the system. The prototype also includes sessions and a temporary in memory credential store that will be used prior to the implementation of a database. Without this, the system would be difficult to test and build effectively.

The final part of this prototype will be the beta test and feedback, where I use the data selected in my design stage to test the effectiveness and robustness of my system. I will reach out to my stakeholders again and ensure scrutiny on the prototype to ensure that the stakeholders will be happy with the final product

Installation of Apache Wicket

The installation of Apache Wicket was a very quick and simple process with the help of their quickstart page which can be found at:

https://wicket.apache.org/start/quickstart.html

The quickstart page presented a UI which asked for a Group and artefact ID, Wicket Version and server to deploy on. I was then given the following maven command that I pasted into IntelliJ console and Wicket was pretty much all set up. The following command also installed Jetty, the server I used for local testing.

mvn archetype:generate -DarchetypeGroupId=org.apache.wicket -DarchetypeArtifactId=wicket-archetype-quickstart -DarchetypeVersion=9.14.0 -DgroupId=org.swainston -DartifactId=GCSEAdventOfCode -DarchetypeRepository=https://repository.apache.org/ -DinteractiveMode=false

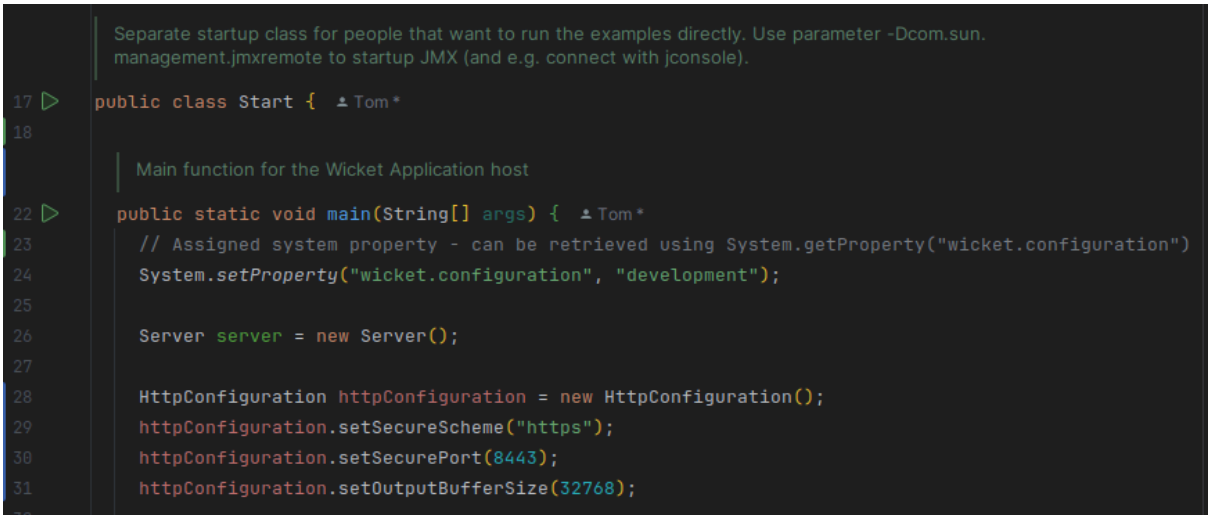
Hosting

As previously mentioned, Jetty was installed with Wicket using the command. Running the applications was simple, and all that's needed is the Start.java file, found under testing, to be run. The webserver would then open port 8080 with the web server which was accessible to me through localhost:8080.

This way of hosting allowed me to edit code within IntelliJ and then re-run the Start.java file allowing me to edit "live". Please note that a large portion of the code within the Start.java file is not written by me.

The Start.java file is outside of the scope of my IntelliJ project and falls under the test package. This means that if my code is shipped, it can be hosted in a way that the client prefers and they are not fixed to my hosting configuration.

In this file, the server object is created, as well as the HTTP configuration, which is assigned the schema, secure port and output buffer size.



The server and HTTP configuration are then parsed into a ServerConnector. This is where the port of the server is set, in my case I left it as the default web server port (8080), and also the idle timeout. This connector is then assigned to the server previously created.

```
33      ServerConnector connector = new ServerConnector(server, new HttpConnectionFactory(httpConfiguration));
34      // Sets the port for the server to run on
35      connector.setPort(8080);
36      connector.setIdleTimeout(1000 * 60 * 60);
37
38      server.addConnector(connector);
39
```

The path to the web.xml, which contains information about my project, is then set to the server

```
39
40          WebAppContext webAppContext = new WebAppContext();
41          webAppContext.setServer(server);
42          webAppContext.setContextPath("/");
43          webAppContext.setWar("src/main/webapp");
44
```

Finally, the server is then started and joined.

```
52      try {
53          server.start();
54          server.join();
55      } catch (Exception e) {
56          e.printStackTrace();
57          System.exit( status: 100);
58      }
59    }
60  }
```

Main class

This class was a partially preloaded file that was created when I executed the maven command to create the workspace. It is a very basic class which extends the WebApplication which is the main class that I will be using to run my project.

Within the class, the homepage for the website is set. This is done by returning the class that is named the same as my HTML code for the web page, in my case “HomePage.class”

```
Application object for your web application. If you want to run this application without deploying, run
the Start class.
30  public class WicketApplication extends WebApplication {  18 usages  Tom *
      See Also: org.apache.wicket.Application.getHomePage()
34    @Override  1 usage  Tom
35    public Class<? extends WebPage> getHomePage() {
36      return HomePage.class;
37    }
```

The web server is then initialised and the Content Security Policy settings are also set here, including a whitelist for external sources that the web server can access. More on this is discussed in error #2.

```
63      super.init();
64
65      // Required to allow custom fonts to be displayed and not be blocked by Content Security Policy
66      getCspSettings().blocking()
67          .add(CSPDirective.STYLE_SRC, CSPDirectiveSrcValue.SELF)
68          // Google fonts
69          .add(CSPDirective.STYLE_SRC, ...values: "https://fonts.googleapis.com/css")
70          // FontAwesome fonts stylesheet
71          .add(CSPDirective.STYLE_SRC,
72                  ...values: "https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css")
73          // FontAwesome custom font
74          .add(CSPDirective.FONT_SRC,
75                  ...values: "https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/fonts/fontawesome-webfont.woff2")
76          .add(CSPDirective.FONT_SRC,
77                  ...values: "https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/fonts/fontawesome-webfont.woff")
78          .add(CSPDirective.FONT_SRC,
79                  ...values: "https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/fonts/fontawesome-webfont.ttf");
```

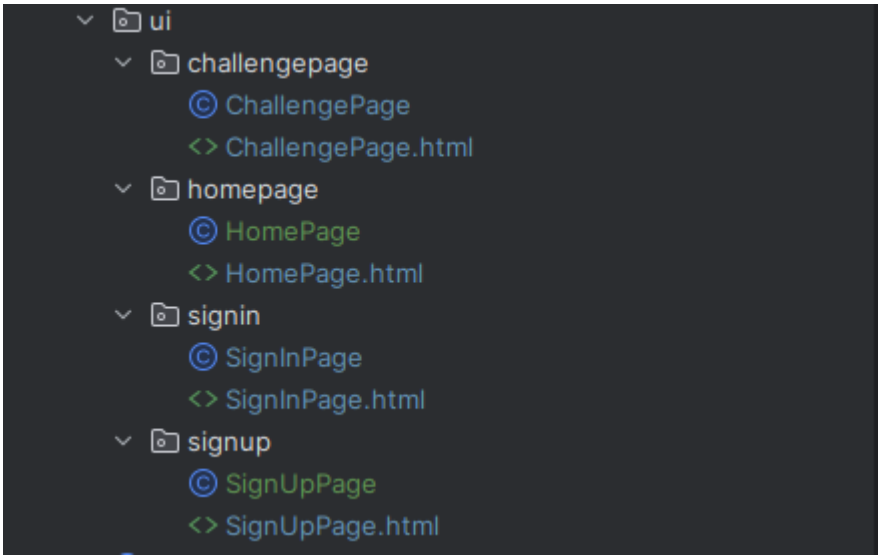
As this class is being accessed and run every time a new webpage is open, it must be the home of the authorization strategy, which is the method that runs to ensure a session is correctly authenticated if an auth-protected page is being opened, and whether a page can be accessed without authorization.

```java
// Register the authorization strategy
getSecuritySettings().setAuthorizationStrategy(
    new IAuthorizationStrategy.AllowAllAuthorizationStrategy() {  Tom *
      @Override  no usages  Tom *
      public <T extends IRequestableComponent> boolean isInstantiationAuthorized(
          Class<T> componentClass) {
        // Check if the page requires auth, if it does it will implement marker class, and checks
        // if the session is signed in
        if (AuthenticatedWebPage.class.isAssignableFrom(componentClass)) {
          if (((SignInSession) Session.get()).isSignedIn()) {
            return true;
          }
          // Session not signed in and page requires auth. Send them to auth, then redirect
          // to original page
          // continueToOriginalDestination when logged in.
          throw new RestartResponseAtInterceptPageException(SignInPage.class);
        }
        // Page we're being directed to does not require authentication.
        return true;
      }
    });
```

Transferring Prototype #1 into my environment

After some research, it did not seem there was a clear way to access external HTML files and use them within an Apache Wicket project. Therefore, I followed the way suggested on their website which is to move the HTML files into my project and to create a script for the page a Java class with the same name as the HTML file.

To make my coding environment more organised and less chaotic, I decided to create a “ui” package that contains a package for each webpage to store each HTML file along with its corresponding Java class. Each package follows the standard Java package naming convention of all lowercase to avoid confusion of packages with classes or interfaces.



HomePage class

This class contains the code required for the scripting aspect of the homepage. This class will be the first page that a user is greeted with, therefore the session does not need to be authenticated before access to the webpage hence why I have not implemented AuthenticatedWebPage. WebPage has been extended so that Wicket understands what type of class this is, and how to link it to the HTML page.

```
    As this page does not implement AuthenticatedWebPage it does not require auth to access
11  public class HomePage extends WebPage {  12 usages  Tom *
12    private static final long serialVersionUID = 1L;  no usages
13
        Method containing the scripts required to run the home page
        Params: parameters – page parameters required for execution
19    public HomePage(final PageParameters parameters) {  no usages  new *
20      super(parameters);
21
```

As my homepage only consists of introductory text and a link that can be followed, there is little code required in this class. The only actions that can be performed on this page lead to the sign-in page, one way is by following the hyperlink in the middle of the page, and the other is by utilising the sign-in button on the nav bar.

```
22      // Response page for the hyperlink "Go to challenges"
23      add(new Link<Void>( id: "challengePageLink") {  new *
24        public void onClick() {  no usages  new *
25          setResponsePage(new ChallengePage());
26        }
27      });
28
29      // Response page for the navbar sign in button
30      add(new Link<Void>( id: "signIn") {  new *
31        public void onClick() {  no usages  new *
32          setResponsePage(new ChallengePage());
33        }
34      });
35    }
```

To access elements within the HTML page from the Java class, Wicket IDs must be given to the elements, which are separate from the HTML ID tags.

```
19      <a wicket:id="signIn" id="signout"><i class="fa fa-user"></i></a>
37          knowledge to be able to answer and gain full marks.</p> <br/>
38      <p><a wicket:id="challengePageLink">Go to the challenges</a></p>
39  </div>
```

Testing the Authorization Strategy

Before I continue, I think it would be best to produce evidence of validation authorisation strategy as the challenge page will need to be authentication protected, which is the next page I will be developing.

I plan to test it by initially not implementing AuthenticatedWebPage on the homepage and attempting to load the page. I expect it to load without redirecting me to a sign-in page. I will then implement AuthenticatedWebPage and then attempt to load the page. It should redirect me to the sign-in page.

Without implementing AuthenticatedWebPage

```
    As this page does not implement AuthenticatedWeb
12  public class HomePage extends WebPage {  14
13      private static final long serialVersionU
```

Page was accessed without a redirect to the signin page. Test passed.

With implementing AuthenticatedWebPage

```
12  public class HomePage extends WebPage implements AuthenticatedWebPage {
```

The page did redirect me to the sign-in page. Test passed.

The next part of the system that needs to be tested requires me to have credentials that I can log into the system with. For this, I must create a credential store interface and a temporary in-memory credential store whilst the database does not exist.

Credentials store interface

As I only intend to create a temporary in memory credential store then once the database is complete, migrate into a database credential store, it seems best suited to create a credential store interface to ease the migration across.

This interface should only require 3 basic methods: add, add a new email and password into the system; exists, to check if a pair of credentials already exist in the store; and finally validate, to check whether a pair of credentials exists in the store and validate a login.

Here is the coded version of the planned design from above.

```
1   package org.swainston;
2
    2 implementations
3   public interface CredentialsStore {  4 usages  Tom
      2 implementations
4     boolean add(String email, String password);  Tom
5
      2 implementations
6     boolean exists(String email);  4 usages  Tom
7
      2 implementations
8     boolean validate(String email, String password);  1 usage  Tom
9   }
```

In memory credential store

The in memory credential store will be using Java hashmaps, storing the email as the key and the value as the password. This will therefore be a hashmap, with the type string and string. This is created like so.

```
8   private final Map<String, String> credentialMap = new HashMap<>();  4 usages
```

As this map is only required within this class, it can be made private meaning that it can only be accessed and edited within this class.

All methods within this class have been marked with the annotation “Override”. This annotation is being used and this class is implementing a interface and this prevents error from occurring, although it is not essential

The first method in this store is the add function, which adds credentials to the map, ensuring they do not already exist in the store.

```
10      @Override  Tom *
11      public boolean add(String email, String password) {
12          if (!exists(email)) {
13              credentialMap.put(email, password);
14              return true;
15          } else {
16              return false;
17          }
18      }
```

Although it would have been effective to add validation in this module, it makes more sense to add it to the module that is taking the input straight from the text boxes. This decreases processing power and time, as well as removes the need for me to code the system twice, for each credential store.

The next function, checks whether a pair of credentials already exist within the store. This function is only used within the add function to ensure that duplicate users cannot be created.

```
20      @Override  4 usages  Tom *
21      public boolean exists(String email) {
22          return credentialMap.containsKey(email);
23      }
24
```

Note that this method has four uses, contradicting my above statement. The reasoning for 2 of these usages is because maps are cleared every time the system is restarted, which is very regular in my case, and it would not be ideal to have to create an account every time. To combat this I created the following block of code.

```
17          if (!credentialsStore.exists( email: "foo")) {
18              credentialsStore.add( email: "foo",  password: "bar");
19          }
```

This is run when the system is started up, giving me an account that I can just log into.

The final function is validate. Its job is to check whether a pair of credentials exists within the map, and if they do return true.

```
25      @Override  1 usage  Tom *
26      public boolean validate(String email, String password) {
27          return (credentialMap.containsKey(email) && credentialMap.get(email).equals(password));
28      }
29  }
```

Sign-in page

Logically, I decided that this would be the most appropriate class to code next. This page is required as a foundation for the challenge class, which will require authentication before accessing. This page is also required to be functional to allow me to test the authorization strategy.

The first part of this page was trivial, and almost a copy and paste from previous classes, due to my repeating navigation bar at the top of the page. As we are on the sign in page, the only functional button that I required on the navigation bar was the redirect to the homepage.

```
41          // Navigation bar home button
42          add(new Link<Void>( id: "go-home") {  Tom *
43              public void onClick() { setResponsePage(HomePage.class); }
46          });
```

The difficulty with this page is that it requires me to take inputs from the HTML page, which seemed like a trivial task, but turned out to be slightly challenging.

The solution to this was to use a form, but even this was not a simple task.

I created a subclass, named “SignInForm” which is private, static and final. As I am creating a form, the class extends Apache Wicket “Form” of type void.

```
    Class required to create the form that takes the users credentials
60  private static final class SignInForm extends Form<Void> {  1 usage
61      private static final String USERNAME = "username";  3 usages
62      private static final String PASSWORD = "password";  3 usages
```

The two strings are set constants, required later.

This class requires a constructor, which takes the parameters ID, which is the wicket:id of the HTML component. This is then parsed by the super class. Three new components are then added to the script, two being the username and password and the last being the button.

```
    Method that creates and handles the data given within the form
    Params: id – the wicket:id of the HTML component
71  public SignInForm(final String id) {  1 usage  Tom
72      super(id);
73
74      add(new TextField<>(USERNAME, new PropertyModel<String>(properties, USERNAME)));
75      add(new PasswordTextField(PASSWORD, new PropertyModel<>(properties, PASSWORD)));
76      add(new Button( id: "signInFormSubmit"));
77  }
```

The next block of code is executed when the button is pressed, meaning that the text boxes have been filled with the user's information. The session of the users is fetched and assigned to the variable “signInSession” which is not set as a type, thanks to Java 10.

The program then attempts to create a session using the details provided, and if the session is successful, the original destination is fetched and the user is sent to there, otherwise the session sign in will fail and the user will be prompted to check their inputs.

NB: This is the first instance where sessions have been mentioned. They have not been covered yet but are next.

```
79      @Override  no usages  Tom *
80      public void onSubmit() {
81
82          // Get session info for the user
83          var signInSession = getMySession();
84
85          // Sign the user in and proceed to the original page that was the destination before
86          // flow was interrupted by this login action.
87          if (signInSession.signIn(getUsername(), getPassword())) {
88              continueToOriginalDestination();
89          } else {
90              // Get the error message from the properties file associated with the Component
91              String loginError = getString( key: "loginError",  model: null,  defaultValue: "Sign in details not recognised.");
92
93              // Register the error message with the feedback panel
94              error(loginError);
95          }
96      }
```

Sessions

The main functions of this class is to verify that a user's credentials exist, sign a user out and create new accounts. This class extends AuthenticatedWebSession.

The first variable in this class is of type CredentialsStore and does not change, therefore final. This variable is where the desired credential store type is assigned. As the database has not been created yet I am using the InMemoryCredentialStore.

```
7
        Used for verifying credentials for sessions
11      public final class SignInSession extends AuthenticatedWebSession {  11 usages  Tom
12
13        // Credential store is set to desired store
14        private final CredentialsStore credentialsStore = new InMemoryCredentialsStore();
15
16        private String username;  5 usages
17
```

The variable username is simply the username of the user being handled. It is a global variable as it is used between multiple methods simultaneously.

This class does contain a constructor, and its main function is to parse the request object to the super class, however, I have used this constructor to house my code that creates an account that I can use for testing. I used “foo” for the username, and “bar” for the password.

```
17
            Constructing method that is run when the class is called
            Params: request – the request object (as per apache javadoc)
22        public SignInSession(Request request) {  1 usage  Tom
23          super(request);
24          // Credentials that always exist to allow for testing
25          if (!credentialsStore.exists( email: "foo")) {
26            credentialsStore.add( email: "foo",  password: "bar");
27          }
28        }
```

It may seem counterintuitive to house this code within the constructor of the sessions class, however, I have used it here as the main class creates a new session every time the server is accessed, as per this function within the WicketApplication class.

```
45      @Override  no usages  Tom
46      public Session newSession(Request request, Response response) {
47          return new SignInSession(request);
48      }
```

As we know, a constructor is a block of code that is executed when the class is called, and in this case the class is called, and given a request meaning that my code to create the temporary account will be executed.

The next method is the sign out method. This method is substantially simplified due to the fact I am using Apache Wicket sessions meaning that all actual server sessions are handled by Apache Wicket itself.

```
        Trivial sign out method
33      @Override  2 usages  Tom
34      public void signOut() {
35          super.signOut();
36      }
37
```

As you can see, signOut() is a method from the superclass AuthenticatedWebSession; a small peek into that class shows that the method invalidates the session.

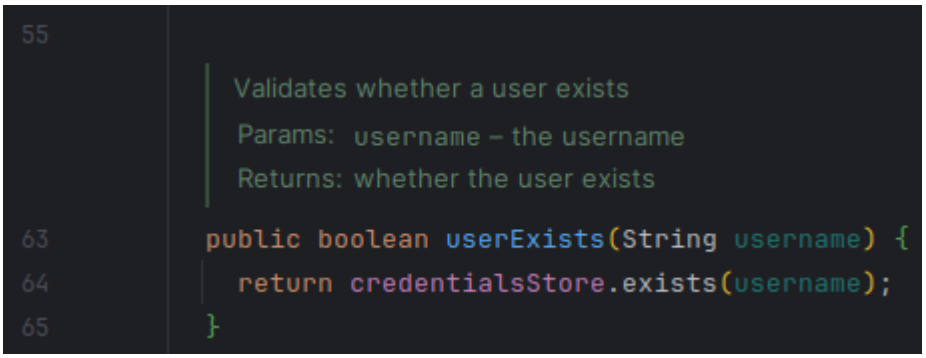
```
        1 override
45      public void signOut() {  2 usages
46          this.invalidate();
47      }
```

The authentication method is the next method. This method is a boolean that returns true if a user's credentials can be authenticated and returns false if they cannot. The parameters for the method are the username and password.

```
37
        Authenticates the given username and password.
        Params: username – the username
                password – the password
        Returns: true if the user was authenticated
46      @Override  Tom *
47      public boolean authenticate(final String username, final String password) {
48        if (credentialsStore.validate(username, password)) {
49          this.username = username;
50        } else {
51          this.username = null;
52        }
53        return this.username != null;
54      }
```

As I have two variables with the same name, username, I have used “this” to ensure I am changing the correct variable.

The userExists method is simple, and just checks whether a username exists within the credential store.



```
55
        Validates whether a user exists
        Params: username – the username
        Returns: whether the user exists
63      public boolean userExists(String username) {
64        return credentialsStore.exists(username);
65      }
```

The next method in this class that contains logic is the sign up method. This method utilises the userExists method to ensure that duplicate users are not being created. It then uses the method add from the interface class to add it to the credential store.

```
    Creates a new user in the credential store
    Params: username – the validated username
            password – the password
    Returns: whether the sign-up was a success
75  public boolean signUp(final String username, final String password) {
76      if (!userExists(username)) {
77          return credentialsStore.add(username, password);
78      } else {
79          return false;
80      }
81  }
```

The final method is a simple getter and returns username.

```
    Getter method for the username attribute
    Returns: the username
87  public String getUser() {   Tom
88      return username;
89  }
```

Sign-up

This class contains the script for the sign up page. As it is a web page, it extends Apache Wicket WebPage class but as this class does not require a login to view, it does not extend AuthorizedWebPage which would make the page require an account to view.

```
16      import java.util.regex.Pattern;
17
            Contains the script for the sign-up HTML page.
            Does not implement org.swainston.AuthenticatedWebPage as this page does not require
            authentication to be viewed.
            See Also: WebPage
26      public class SignUpPage extends WebPage {  2 usages   Tom *
27
```

This class contains a constructor which is parsed to the superclass. This is also where the HTML components are initialised, however, these are all the navigation bar components which have already been shown in previous classes, so I will not show them again.

Similar to the sign in class, this class contains a private static final SignUpForm subclass which extends Apache Wicket Form. It also contains a value map that is populated by constants defined first.

```
            The subclass that houses the code to handle the form used for the sign-up
            See Also: Form,
                      ValueMap
60      private static final class SignUpForm extends Form<Void> {  1 usage
61
62          // Constants required to populate value map
63          private static final String USERNAME = "username";  3 usages
64          private static final String PASSWORD = "password";  3 usages
65
66          private final ValueMap properties = new ValueMap();  4 usages
```

As these variables and the subclass itself are only used within the SignIn class, they can all be private.

This subclass does contain a constructor, and similar to its main class it is used to parse the Form ID to the super class, Form, as well as initialising the components of the form, which are the two text boxes and the submit button.

```
    Constructing method that initialises the components of the form
    Params: id – form id
72  public SignUpForm(final String id) {  1 usage  Tom *
73      super(id);
74
75      //Username text field
76      add(new TextField<>(USERNAME, new PropertyModel<String>(properties, USERNAME)));
77
78      // Password text field
79      add(new PasswordTextField(PASSWORD, new PropertyModel<>(properties, PASSWORD)));
80
81      // Sign up button
82      add(new Button( id: "signUpFormSubmit"));
83  }
```

The next method is the onSubmit method, which is executed when the button is pressed. This method is where the validation for the username and password is. It also, of course, is where the session and account are created for the user.

```
    Submit method that is executed when the sign-up button is pressed
    Handles all validation required for the user accounts
91      @Override  no usages  Tom *
92      public void onSubmit() {
```

Email validation

As discussed previously, I am using regex to validate the user's email and password. Java contains a regex class within java.util which makes using regex very straightforward. The first step is to create a variable of type Pattern using Pattern.compile and the regex statement chosen.

```
93
94      // Validation for the email
95      Pattern emailPattern = Pattern.compile(
96          regex: "^(?=.{1,64}@)[A-Za-z0-9_-]+(\\.[A-Za-z0-9_-]+)*@[^-][A-Za-z0-9-]+(\\.[A-Za-z0-9-]+)*(\\.[A-Za-z]{2,})$");
97
98      // Validation for the password using regex
99      Pattern passwordPattern = Pattern.compile( regex: "^[a-zA-Z0-9_]+$");
```

The next step is to use the pattern matcher. This is a method that is executed from the pattern parsing the string, and then producing a boolean by using the “matches” method.

```
100
101         String username = getUsername();
102
103         if (!emailPattern.matcher(username).matches()) {
104             // Get the error message from the properties file associated with the Component
105             String loginErrorInvalidUsername = getString( key: "loginErrorInvalidUsername",  model: null,
106                     defaultValue: "Email address invalid.");
107             error(loginErrorInvalidUsername);
108             return;
109         }
```

An identical logic is used for the password, the only difference being the message being sent to the user if the pattern does not match

```
111         String password = getPassword();
112
113         if (!passwordPattern.matcher(password).matches()) {
114           // Get the error message from the properties file associated with the Component
115           String invalidPassword = getString( key: "loginErrorInvalidPassword",  model: null,
116                 defaultValue: "Only letters, number, underscore in password");
117           error(invalidPassword);
118           return; // break to ensure that no unnecessary session are created
119         }
```

Sign-up sessions

The sign-up session is very similar to the sign-in session code, however, the difference is that the code is nested within an if statement which checks the condition of the session sign up with the given credentials.

```
121           // Sign up, and return to the original destination page.
122           SignInSession signInSession = getMySession();
123
124           // Check if the sign-up is successful
125           if (signInSession.signUp(username, password)) {
```

If the sign up is successful, the user's session is signed in with the new credentials and they are redirected to the original destination.

```
127             // Sign the user in
128             if (signInSession.signIn(username, password)) {
129               continueToOriginalDestination();
130             } else {
```

Otherwise, the user is given a standard message of failed to log in, although this case should not logically occur, it is best to have it covered in case it does happen.

If the sign up fails, the reason for the for the fail is investigated, and if it is found that the account already exists, the error message is set as so

```
138            // Get the error message from the properties file associated with the Component
139            String errorMessage;
140            if (signInSession.userExists(getUsername())) {
141              errorMessage = getString( key: "loginError",  model: null,
142                   defaultValue: "Unable to sign you up - that user name already exists");
```

But if this is not the case, a more general error is set.

```
    } else {
      errorMessage = getString( key: "loginError",  model: null,  defaultValue: "Unable to sign you up");
    }
    // Register the error message with the feedback panel
    error(errorMessage);
```

Finally, the error is produced to the user.

The final 3 methods are getters but not for attributes of the object.

The first is the getPassword method but returns a string, the password from the properties valuemap

```
        Returns: password from properties ValueMap
155     private String getPassword() {  1 usage   Tom
156       return properties.getString(PASSWORD);
157     }
158
```

The second is the getUsername, which similarly, returns a string, the username from the properties map

```
        Returns: username from properties ValueMap
163     private String getUsername() {  2 usages   Tom
164       return properties.getString(USERNAME);
165     }
```

The third is getMySession which returns a SignInSession object. This method uses getSession to get the session from Apache Wicket then casts it to my class, then returns it.

Stakeholder feedback

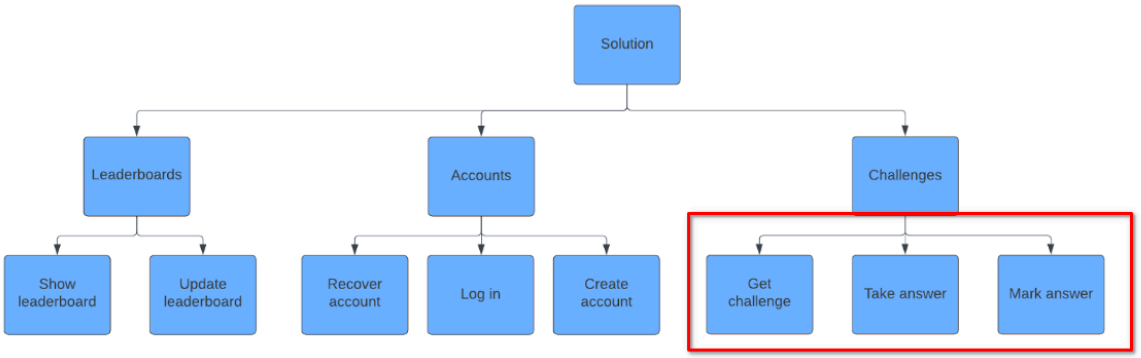
Following this prototype, I approached both the students and the teachers to gain some feedback on the most recent updates to the system. These were the general comments from both.

Students - the students were a fan of the general simplicity of the login system and how there was no fluff or extra unnecessary code. They also liked the idea behind how Authenticated Web Pages are identified as needing authentication

Teachers - the teachers liked the emphasis of validation on the email for the system as it means that if the teacher wanted to track progress of a student or see what their attempts look like, they are able to easily differentiate between students and see who contributed which code. The teachers were also very fond of the use of sessions within the website.

Prototype #3 - The game engine

This is the most complex and time consuming prototype. It is essential that this prototype is not rushed and minimal bugs should make it through production as this is the main system that the solution depends on. Below you can see the section of the decomposition that should be achieved by the end of this prototype.



ChallengePage class

This page contains the script required for the interactive part and some processing for the challenge page.

```
This class handles the challenge page and the submitted in the challenge page.
public class ChallengePage extends WebPage implements AuthenticatedWebPage {
```

As this page should only be accessed by students, it is essential that it implements AuthenticatedWebPage which means that only students who are logged in are able to access and view the page.

Not only does this ensure that security of the page is maintained, but it also prevents database and other back-end failures as many processes rely on having a valid session.

```
22      private final ChallengeStore challengeStore;  2 usages
23      private final AttemptsStore attemptsStore;  3 usages
24
        Constructs the challenge page and layout.
28      public ChallengePage() {  2 usages  Tom
29          challengeStore = new MySQLChallengeStore();
30          attemptsStore = new MySQLAttemptsStore();
31          layout();
32      }
```

This next block of code constructs the webpage, initialising both the ChallengeStore and AttemptsStore, which are essential variables required for the challenge page. The constructor also calls the “layout” method. This method contains code that could be placed inside the constructor, however, to ensure easier debugging and maintain a modular program, I created a private method.

```
        Holds all the code required to run the page, process the students' code and return feedback to the
        student
37      private void layout() {  1 usage  Tom *
38          String user = getMySession().getUser();
39
40          // Optional variables are used to ensure that no null variables are able to leak into code
41          Optional<Challenge> nextChallenge = challengeStore.getNextChallenge(user);
```

As this method is strictly used in the ChallengePage class, it is made private meaning that it cannot be accessed or called from another class, as this could cause problems.

The use of optional variables allows me to ensure that no null variables can be parsed into my code which could throw some unwanted errors.

```
44          // Checks that the optional is not empty
45          if (nextChallenge.isEmpty()) {
46              // Theoretically unreachable
47              throw new AssertionFailedError("Unable to find next challenge for user.");
48          }
```

A simple check to see whether the optional variable returned is “present”, basically whether a value has been assigned or not.

```
50        // Get the challenge on the basis that it is not empty
51        Challenge challenge = nextChallenge.get();
52        Optional<Attempt> attempt = attemptsStore.getAttempt(user, challenge.getId());
53        if (attempt.isEmpty()) {
54            MySQLAttemptsStore.loadDefault(user, challenge);
55        }
```

Given that the challenge is present, any previous attempts are attempted to be loaded. If there are none then the default template is loaded from the database.

The next section of code is the generic loading components of the page and navbar, and therefore doesn't require explaining for the third time…

```
57        add(new Label( id: "title", new PropertyModel<>(challenge,  expression: "title")));
58        add(new Label( id: "instructions", new PropertyModel<>(challenge,  expression: "description")));
59        add(new Link<Void>( id: "go-home") {  Tom *
60            @Override  no usages  Tom
61            public void onClick() {
62                setResponsePage(HomePage.class);
63            }
64        });
65        add(new Link<Void>( id: "sign-out") {  Tom *
66            @Override  no usages  Tom
67            public void onClick() {
68                getMySession().signOut();
69                setResponsePage(HomePage.class);
70            }
71        });
```

```
77        private SignInSession getMySession() {  2 usages  Tom
78            return (SignInSession) getSession();
79        }
```

Again, to maintain a modular project, the getSession method is extracted into a method.

```
72
73            add(new ChallengePageForm(user, challenge, attemptsStore));
74        }
```

This is perhaps the most important line within this whole class, what this does is initialise the textbox within the challenge page and allow me to read from, compile and execute code within.

ChallengePageForm class

The ChallengePageForm class contains the code that calls the compiler and runs and executes the test classes on the students' code. As this page uses Apache forms, this class extends Form.

```
    Students' code is compiled and tests called on it within this class
16  public class ChallengePageForm extends Form<Attempt> {  1 usage
17
18    private final Challenge challenge;  2 usages
19    private final String user;  2 usages
20    private final AttemptsStore attemptsStore;  2 usages
```

The attributes required for the ChallengePageForm object are defined here.

```
    ChallengePageForm object
    Params: user – the student
            challenge – the challenge the student is attempting
            attemptsStore – the attempt store that holds the student's attempts
28  public ChallengePageForm(String user, Challenge challenge, AttemptsStore attemptsStore) {
29    super( id: "form-submit-code");
30
31    this.challenge = challenge;
32    this.user = user;
33    this.attemptsStore = attemptsStore;
34
35    Attempt modelObject = attemptsStore.getAttempt(user, challenge.getId()).get();
36    setModel(new Model<>(modelObject));
37    add(new TextArea<>( id: "textarea-code",
38        new PropertyModel<Attempt>(modelObject,  expression: "attempt")));
39  }
```

This is the constructor for the class. The superclass, form, is called with the string “form-submit-code”, this is how the textbox has its ID set for future referencing.

Each of the attributes is initialised with the arguments given when calling the class.

A model object is then defined using the students username and the ID of the challenge given.

The final method within the constructor is the add method, this method adds the text box on the web page containing the given attempt.

The next method is the onSubmit method.

```
    The function that is run when the student presses the submit button
@Override  no usages  Tom *
protected void onSubmit() {

    TextArea<Attempt> textArea = (TextArea<Attempt>) get("textarea-code");
    IModel<Attempt> textAreaModel = textArea.getModel();
```

This method is a protected, void method. This means that it can only be accessed from classes within the same package as it, and that it does not return anything.

The TextArea<Attempt> is an Apache Wicket form type with the generic of the attempt class. This variable fetches the text area that contains the student's code.

The IModel<Attempt> fetches the model from the TextArea allowing me to eventually retrieve the code.

```
    Object attemptObject = textAreaModel.getObject();
    String studentCode = (String) attemptObject;
    Attempt attempt = new Attempt();
    attempt.setAttempt(studentCode);
```

The Object attemptObject fetches the Object contained within the IModel of the text area.

A new attempt object is then created, using new Attempt();

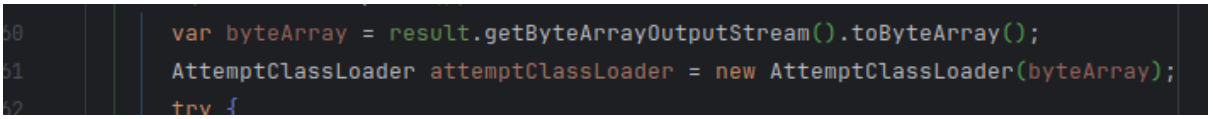
This Object is casted to the type String so that it can be parsed as an argument to setAttempt, a method within the attempt object.

```
    final Compiler.Result result = Compiler.checkCompiles(attempt);

    if (result.isCompiles()) {
```

This part is quite self-explanatory. A new object of type Result from my Compiler class is created through the user's attempt.

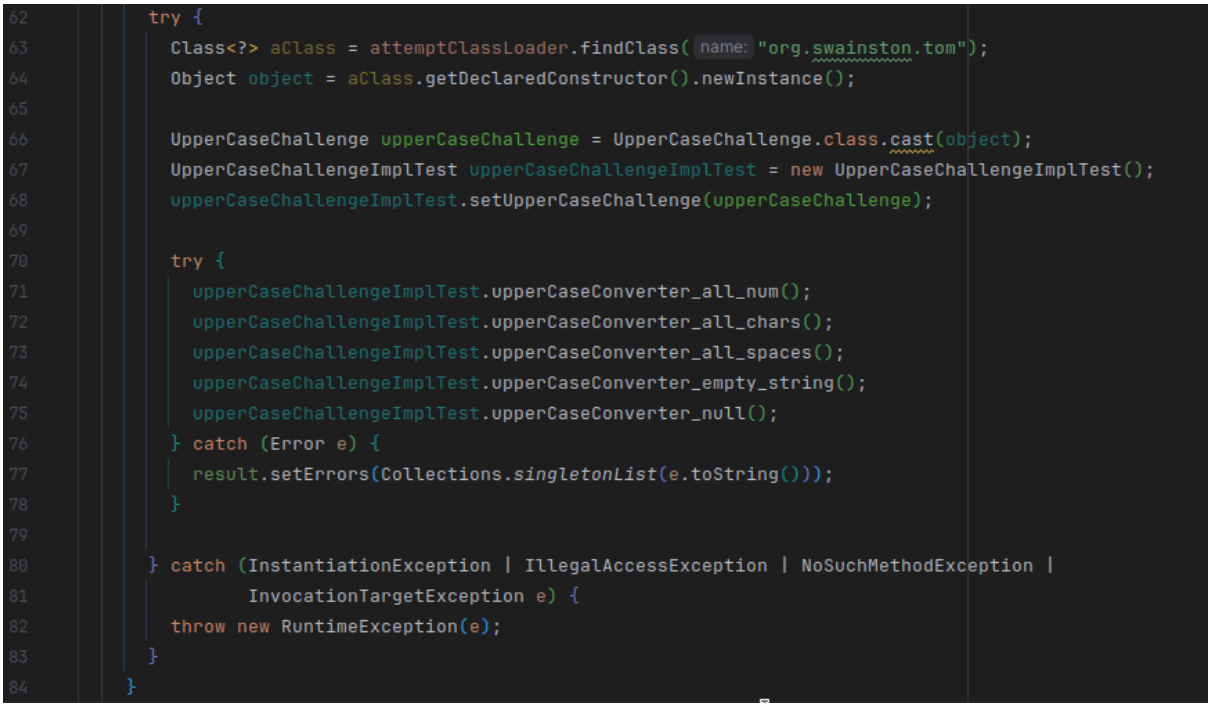
This object's attributes can be accessed through getters and setters and isCompiles returns a boolean as to whether the students code compiles or not.



```
var byteArray = result.getByteArrayOutputStream().toByteArray();
AttemptClassLoader attemptClassLoader = new AttemptClassLoader(byteArray);
try {
```

If the student's code does compile, then the output stream from the result is fetched and converted to a byte array. This byte array is then parsed to create a new AttemptClassLoader object.

The next set of code is the tests. For this example, I have only included the tests for one specific question, however, the rest are simple to implement with a switch statement.



```
try {
    Class<?> aClass = attemptClassLoader.findClass( name: "org.swainston.tom");
    Object object = aClass.getDeclaredConstructor().newInstance();

    UpperCaseChallenge upperCaseChallenge = UpperCaseChallenge.class.cast(object);
    UpperCaseChallengeImplTest upperCaseChallengeImplTest = new UpperCaseChallengeImplTest();
    upperCaseChallengeImplTest.setUpperCaseChallenge(upperCaseChallenge);

    try {
        upperCaseChallengeImplTest.upperCaseConverter_all_num();
        upperCaseChallengeImplTest.upperCaseConverter_all_chars();
        upperCaseChallengeImplTest.upperCaseConverter_all_spaces();
        upperCaseChallengeImplTest.upperCaseConverter_empty_string();
        upperCaseChallengeImplTest.upperCaseConverter_null();
    } catch (Error e) {
        result.setErrors(Collections.singletonList(e.toString()));
    }

} catch (InstantiationException | IllegalAccessException | NoSuchMethodException |
         InvocationTargetException e) {
    throw new RuntimeException(e);
}
}
```

The first variable is of type class and it is formed by retrieving the class from the AttemptClass loader. This is the compiled class with the students code within.

An object is then created from the class which is then casted to the type upperCaseChallenge. This type allows the test cases to be run on the students code.

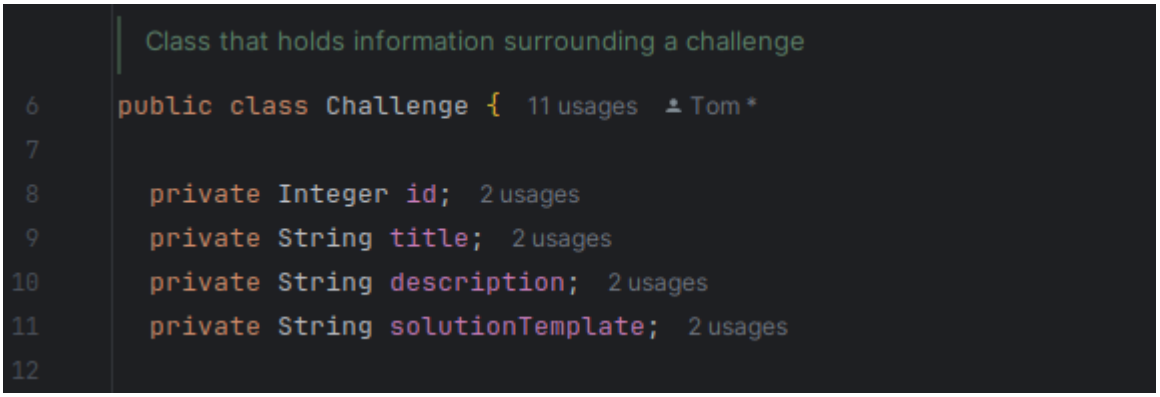
The tests are run! The way the tests are formatted will be shown a few pages down.

Note that there is a nested try catch clause. The inner try catch clause catches errors within the student code and returns sets the error on the result object. The outer

catch class catches errors that occur while attempting to create the object that is used to run the tests on.

Challenge class

This class holds the information to do with a certain challenge. There are four attributes of this class.



The first three are quite self explanatory, the last one might be a little vague, however, it's the text that is shown to the student that they are able to edit and type their answer into. This gives the student a head start and makes it less confusing.

The rest of this class is getters and setters for the four attributes.


Fetches solution template for a challenge
Returns: set template for the solution defined in the database
24 public String getSolutionTemplate() { 1 usage Tom
25 return solutionTemplate;
26 }
27
Sets the template for a certain solution
Params: solutionTemplate – the template for the solution
33 public void setSolutionTemplate(String solutionTemplate) {...}
36
Returns: id of a challenge
40 public Integer getId() { return id; }
43
Sets the id of a challenge
Params: id – the challenge id
49 public void setId(Integer id) { this.id = id; }
52
Returns: title of the challenge
56 public String getTitle() { return title; }
59
Sets the title of a challenge
Params: title – the new title
65 public void setTitle(String title) { this.title = title; }
68
Returns: the description of a challenge
72 public String getDescription() { return description; }
75
Sets the description of a challenge
Params: description – the new description
81 public void setDescription(String description) { this.description = description; }
84 }

Attempt class

The attempt class is the object where the information surrounding a user's attempt is loaded.

```
5    public class Attempt implements Serializable {  15 usages  Tom
6
7      private int id;  2 usages
8      private String email;  2 usages
9      private String attempt;  2 usages
10
```

This class contains three attributes: id, email and attempt. The ID is a generated ID of the user's attempt and the email is the unique identifier of the user, and the attempt is the String form of the user's attempt.

The rest of this class consists of intelliJ generated getters and setters.

```
11     public int getId() { return id; }
14
15     public void setId(int id) { this.id = id; }
18
19     public String getEmail() { return email; }
22
23     public void setEmail(String email) { this.email = email; }
26
27     public String getAttempt() { return attempt; }
30
31     public void setAttempt(String attempt) { this.attempt = attempt; }
34   }
```

Test class (UpperCaseConverter challenge)

This class houses the tests that I will be running on the UpperCaseConverter challenge. This is the first test class that I wrote and all of the test classes will be almost identical minus the data being executed for the test, so this class is a blueprint for the rest.

JUnit 5

For the tests I have used JUnit version 5. A link to their website can be found here:

https://junit.org/junit5/

I imported the JUnit 5 dependency via my pom.xml as I am using Maven to build my project. I added it to my pom as so:

```
79          <!--  JUNIT DEPENDENCY FOR TESTING -->
80
81          <dependency>
82              <groupId>org.junit.jupiter</groupId>
83              <artifactId>junit-jupiter-api</artifactId>
84              <version>5.7.2</version>
85      <!--            <scope>test</scope>-->
86          </dependency>
```

After adding this to my pom, I ran the following command:

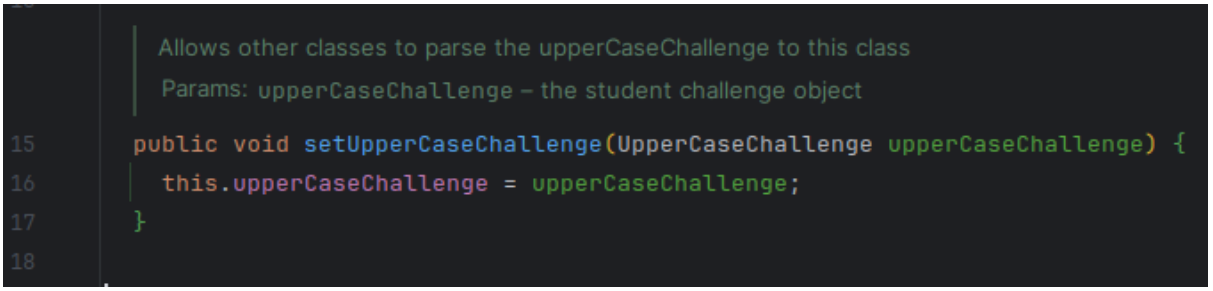
“mvn clean -U”

The maven subcommand “clean” cleans the maven workspace, to help save some storage but mainly to remove garbage files, and the argument “-U” checks for updates in repositories for the dependencies as well as working effectively to check whether there have been any new dependencies added to the pom. Information regarding maven commands can be found on their official website:

https://maven.apache.org/run.html

The test class

This is a public class that does not extend any superclasses, as this class is independent of Apache Wicket and the database there are no requirements for superclasses.



The first variable that is declared is the upperCaseChallenge which is a new instance of the implemented class.

The method setUpperCaseChallenge allows other classes to give the instance of the UpperClassChallenge and it can then be set as a variable previously declared above.

Onto the tests themselves, these tests are quite similar, so I will only run through the two different types of tests in this instance.

The first thing to note is that all tests are annotated by “@Test”. This tells JUnit that the public void method defined below can be run as a test case. The first variable, “test” is the test data that is parsed into the students code. The second variable “expected” is the expected and correct outcome of the program, and the final variable “actual” is the data that is returned when the students program executes with the previously defined test data.

The final line is where the test passes or fails. Assertions.assertEquals checks to see whether two variables are the same. If they are the same, then it sets the test as passed, and if they are not the same it sets the test as failed.

The next type of test is shown below.

```
    Tests to see whether given a null value, the code throws a NullPointerException
42  @Test  Tom
43  public void upperCaseConverter_null() {
44      Executable executable = () -> upperCaseChallenge.upperCaseConverter( text: null);
45      Assertions.assertThrows(NullPointerException.class, executable);
46  }
```

This test differs slightly from the test above as it does not check for outputs given from the students code, but instead whether an error is thrown. In the instructions for this challenge, it says to throw an error if the inputted value is null, so if the program does, it passes the test otherwise it fails.

The rest of the challenges are very similar to the first example I went through, however, they are shown below, unannotated.

```
    Tests whether given numbers, it does not alter the string at all
34  @Test  Tom
35  public void upperCaseConverter_all_num() {
36      String test = "23748923749";
37      String expected = "23748923749";
38      String actual = upperCaseChallenge.upperCaseConverter(test);
39      Assertions.assertEquals(expected, actual);
40  }
```

```
    Tests whether given an empty string, it returns an empty string again
62  @Test  Tom
63  public void upperCaseConverter_empty_string() {
64      String test = "";
65      String expected = "";
66      String actual = upperCaseChallenge.upperCaseConverter(test);
67      Assertions.assertEquals(expected, actual);
68  }
```

```
    Tests whether to see a string of spaces is kept the same and not changed
54  @Test  Tom
55  public void upperCaseConverter_all_spaces() {
56      String test = "    ";
57      String expected = "    ";
58      String actual = upperCaseChallenge.upperCaseConverter(test);
59      Assertions.assertEquals(expected, actual);
60  }
```

UpperCaseChallenge interface

A short and sweet, yet useful interface. This public interface, UpperCaseChallenge is required to define the UpperCaseConverter that is used for the students' attempt.

```
3   public interface UpperCaseChallenge {  6 usages  Tom *
        Method that converts a string from uppercase to all lowercase
        Params:  text – string that requires lowercasing
        Returns: lowercase string
    1 implementation
9       String upperCaseConverter(String text);  5 usages  Tom
10
11  }
```

Attempt class

This class is a class which holds information surrounding a student's attempt. This class does not extend any classes but does implement Serializable. This is an Apache Wicket protocol, as without this the class throws many errors to the console.

```
    Class which holds information about a students given attempt
8   public class Attempt implements Serializable {  15 usages  Tom
9
10      private int id;  2 usages
11      private String email;  2 usages
12      private String attempt;  2 usages
```

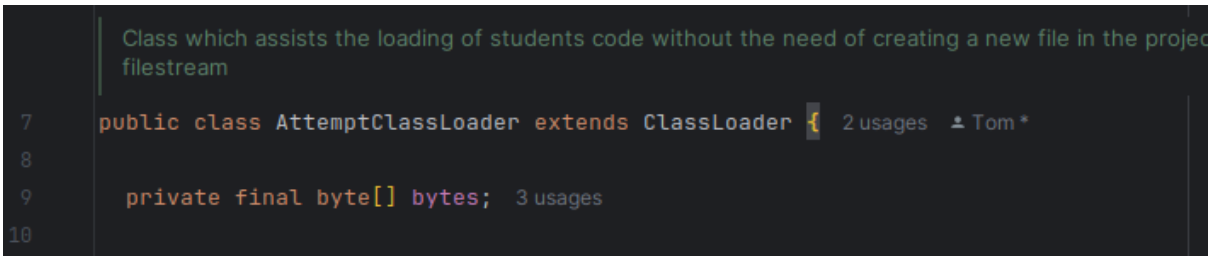
The variables within this class are all related to the attempt from the student, for example, the attempt id, the student's email and the attempt itself. These are all private variables as there is no requirement for them to be accessed outside of this class thanks to the getters and setters, shown below.

```
14      public int getId() { return id; }
17
18      public void setId(int id) { this.id = id; }
21
22      public String getEmail() { return email; }
25
26      public void setEmail(String email) { this.email = email; }
29
30      public String getAttempt() { return attempt; }
33
34      public void setAttempt(String attempt) { this.attempt = attempt; }
37  }
```

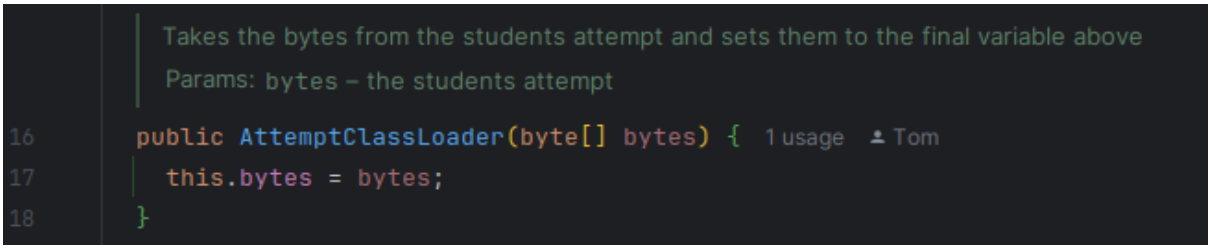
Each attribute has a getter and a setter which allows the variable to be set, and accessed from other classes thus making the actual attribute itself not needing to be public or protected.

AttemptClassLoader

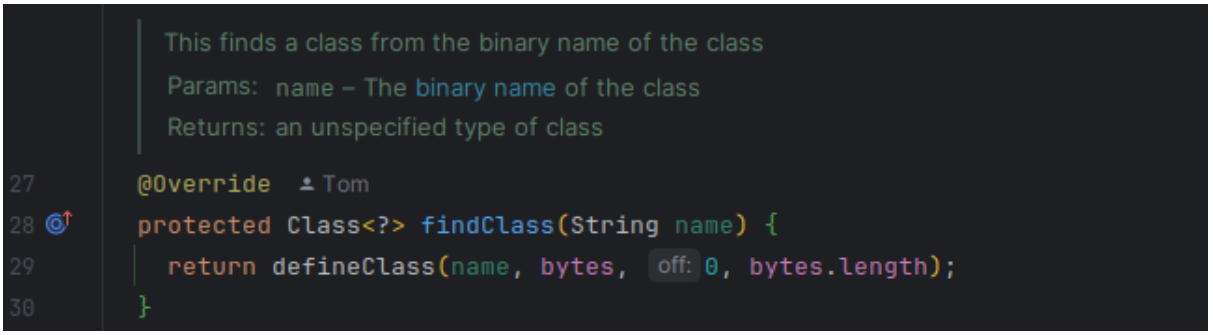
This valuable class is a public class that extends the java.lang ClassLoader. The aim of this class is to dynamically load Java classes into the Java Virtual Machine. Without this class it would be difficult to load and run the students code without creating excess files which would require tracking and deleting after use. Extra code which I could do without!



The first variable, bytes, is defined as a private final constant. It is of type byte array. This means that the scope of this constant is this class only, and that it is final meaning it cannot be changed once it is defined.



This first method is a setter for the byte array defined above. It takes in a byte array from another class and sets the constant bytes to the given byte array.



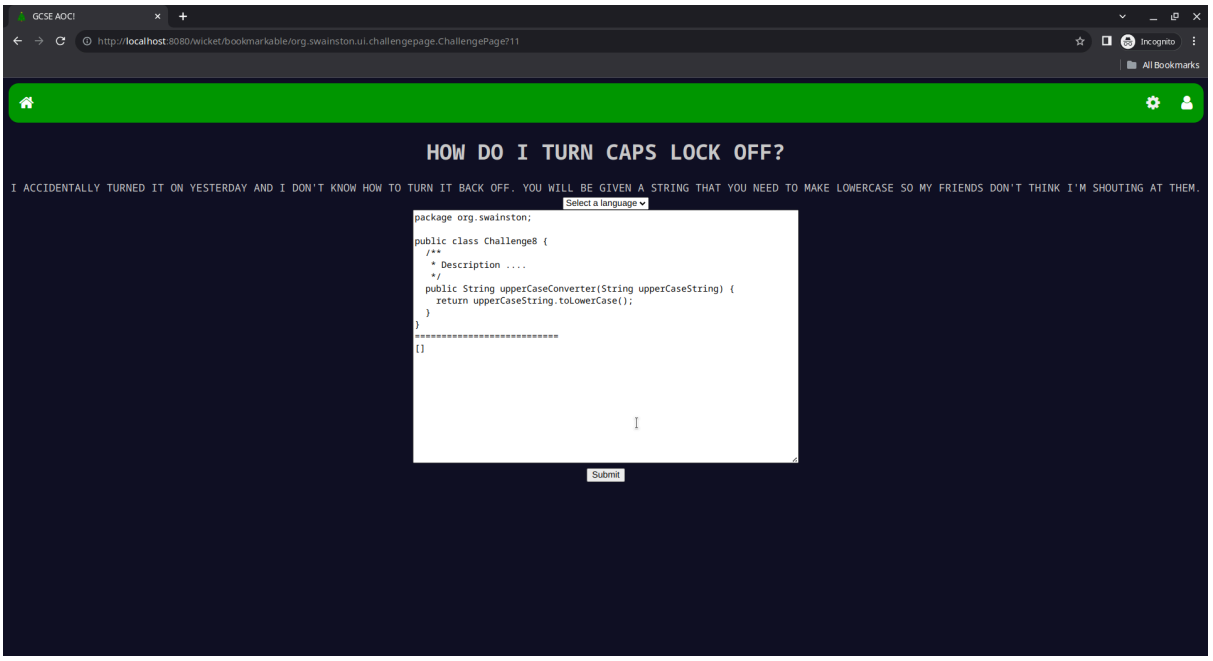
This method, findClass, returns an unspecified type of Class from the binary name of a class. It returns through defineClass which is a method from the superclass ClassLoader.

Alpha test

To prove evidence of testing at each stage, I will refer to the Alpha test section of my design section to see what needs testing.

> ### User inputs
>
> For the text box in the challenge page, there will be one class that I copy and paste into the box and log that it is received by the backend webserver.
>
> ```
> public String upperCaseConverter(String upperCaseString) {
>   return upperCaseString.toLowerCase();
> }
> ```

I pasted the following alpha test data into the challenge page input box and this was the response



The code compiled successfully with no errors, therefore passing this alpha test.

Stakeholder feedback

I spoke briefly to a teacher following this prototype and they were happy with the progress that was being made to the system and commented on the simplistic layout of the challenge page and the way that the feedback was given to the student. He said that in the future, were the system to be updated, it would be a nice feature for the errors and other tests to be shown in a separate box, or on a different part of the page.

Prototype #4 - Databases

Setting up the MySQL database

Using instructions from the MySQL website (https://www.mysql.com/) I installed the latest version (8.0.26) of MySQL on my machine.

Justifying the use of 1NF

After some research and online research, with help from certain articles, including the one listed below, I decided that this project was not sufficiently large enough to justify the use of 2/3NF, and so I have decided to opt for 1NF. If this project were to increase in size, it would make much more sense to ensure that it was at least 3NF, perhaps even greater than that, but at a minimum 3NF.
https://stackoverflow.com/questions/6625569/what-is-the-level-of-database-normalization-that-is-practically-enough

To create the database and confirm it was successful, I ran the following commands:

```
tom@atlas:~$ mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 39
Server version: 8.0.33-0ubuntu0.22.04.4 (Ubuntu)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE GCSEAdventOfCode;
Query OK, 1 row affected (0.01 sec)

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| GCSEAdventOfCode   |
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
+--------------------+
7 rows in set (0.00 sec)

mysql> EXIT;
Bye
tom@atlas:~$
```

To create the tables within the database, I used a .sql file containing SQL script that is run every time the program starts up.

This file initially looked like this:

```
CREATE TABLE IF NOT EXISTS users
(
    email    CHAR(128) NOT NULL PRIMARY KEY,
    password TEXT      NOT NULL,
    progress INTEGER   NOT NULL
);
```

This code snipped is what creates the users table. “CREATE TABLE” are the keywords to create a table, and it is under the condition “IF NOT EXISTS” meaning that if a table by the name “users” exists, it will not execute any of this code, otherwise it will execute it all. The “NOT NULL” keywords mean that the value must not be null when adding an entry into the database, and if there is an empty “NOT NULL” labelled attribute, the program will throw an error. In this case, the unique identifier, annotated by “PRIMARY KEY” is email. This is because each student has a unique email address.

Database class

This class holds the method that makes a connection between my project and the database.

```
14
        Initializes the database source with the server name, port number, database name, user and
        password.
        Returns: DataSource the source from the
        Throws: SQLException – if unable to connect to database
23 @    public static DataSource initDataSource() throws SQLException {  1 usage  Tom *
24
25          MysqlDataSource dataSource = new MysqlConnectionPoolDataSource();
```

The method, named initDataSource is a public static method which returns type DataSource. As there is a chance of the datasource failing to be established, this method may throw an SQLException, and is therefore added to the method signature.

The first variable, dataSource, is of type MysqlDataSource and is from the JDBC library.

The following methods are executed on the variable dataSource.

```
27        dataSource.setServerName("localhost");
28        dataSource.setPortNumber(3306);
29        dataSource.setDatabaseName("GCSEAdventOfCode");
30        // Database user username
31        dataSource.setUser("root");
32        // Database user password
33        dataSource.setPassword("Password123!");
```

Above, the server name, or more commonly IP, is defined, as well as the port number, the database name, and the user's username and password. In this case I created a root user with a basic password as this server is being hosted locally and this does not pose a security risk.

```
35        try (Connection conn = dataSource.getConnection()) {
36            if (!conn.isValid( timeout: 1)) {
37                throw new SQLException("Failed to establish database connection.");
38            }
39        }
40
41        return dataSource;
42    }
```

This try catch statement is where the attempt to connect to the database occurs. The getConnection is called on the datasource that was modified earlier. The isValid method is then run on this connection, and if after 1 second no response is sent, an SQLException is thrown. If a response is set then the program continues and the dataSource is returned as it must be valid.

MySQLCredentialsStore

This class is a public class that implements the CredentialStore interface, previously defined. The aim of this class is to security and effectively carry out the same job that the InMemoryCredentialsStore was doing, but with the implementation of an external MySQL database.

Due to convention, all of the SQL queries have been extracted out to constants. In Java, this has made them private static final Strings.

```
16  public class MySQLCredentialsStore implements CredentialsStore {  2 usages  Tom *
17
18      private static final String SQL_ADD = "INSERT INTO users VALUES(?, ?, ?)";  1 usage
19      private static final String SQL_EXISTS = "SELECT COUNT(*) FROM users WHERE email = ?;";
20      private static final String SQL_VALIDATE =  1 usage
21          "SELECT COUNT(*) FROM users WHERE email = ? AND password = ?;";
22
```

The above constants will make more sense once the methods within this class have been documented.

The first method, add, does the exact same job as the InMemoryCredentialsStore and adds a user to the system, in this case the database.

```
    Adds a user into the database
    Params:  email – the user's email
             password – the user's password
    Returns: boolean of whether successful
29      @Override  Tom *
30      public boolean add(String email, String password) {
31          // Check the user does not already exist in the database
32          if (exists(email)) {
33              return false;
34          }
35          try (Connection conn = WicketApplication.getConnection()) {
36              PreparedStatement stmt = conn.prepareStatement(SQL_ADD);
37              stmt.setString( parameterIndex: 1, email);
38              stmt.setString( parameterIndex: 2, password);
39              stmt.setInt( parameterIndex: 3,  x: 1);
40              stmt.executeUpdate();
41              return true;
42          } catch (SQLException e) {
43              Logger.getLogger(WicketApplication.class.getName()).log(Level.SEVERE, e.getMessage());
44          }
45          return false;
```

The main thing to note is that this method requires multiple database calls as it must ensure that a user does not already exist within the database before adding them.

The next method, which is already called in the method above, is the “exists” method. This method takes a user's email and checks whether there is already an entry with that email in the database. As the email is the primary key there are no duplicates present.

There are numerous ways of checking whether there is already an entry in the database, but I decided that the best way would be by implementing the SQL count keyword. As you can see from the query defined at the top of the class, the basic logic is getting the count of users who match the email as the one given in the arguments, and if this is equal to one, then the user exists.

It’s key to point out that I have understood that my code is not bombproof and that errors may occur, and that is why I have added the second clause to check whether the count returns greater than 1. Theoretically, this should not be possible but if it were to occur errors would most definitely be thrown, so it is best to plan for the worst and include this clause so the issue can be picked up sooner.

MySQLChallengeStore

Similarly, this class bridges the Challenges table and the website together. It implements the interface ChallengeStore which can be seen below.

```
4
                1 implementation
5   public interface ChallengeStore {  3 usages   Tom *
        1 implementation
6       Optional<Challenge> getNextChallenge(String user);  1 usage   Tom
7
8   }
```

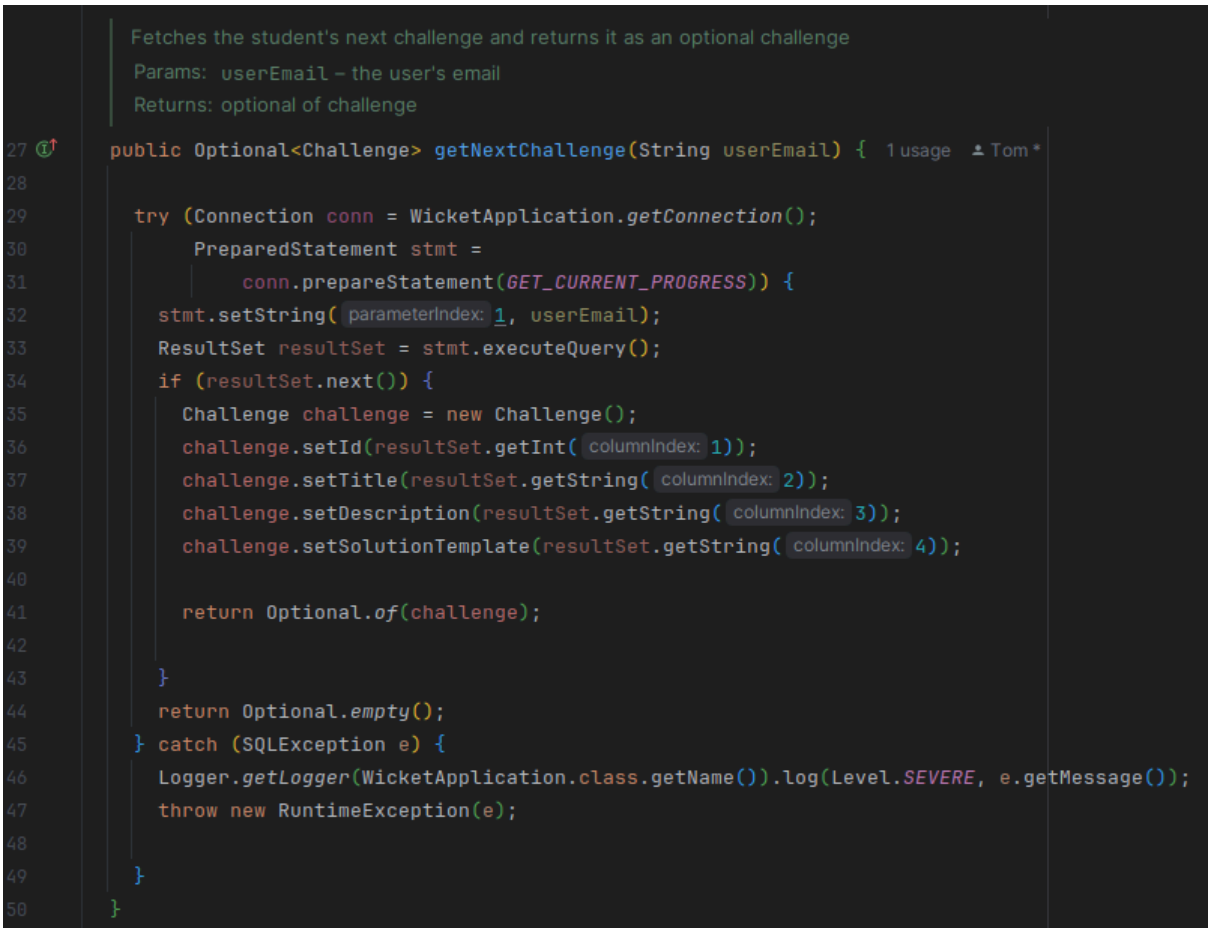
The only constant in the MySQLChallengeStore class is the SQL Query required to get the next challenge for a student.

```
17      private static final String GET_NEXT_CHALLENGE =  1 usage
18          "SELECT * FROM challenges WHERE id = (SELECT progress FROM users where email = ?)";
19
```

This class only houses one method which is getNextChallenge. This method takes one argument of type String which is the student's email address, again, the unique identifier.

In this code block, an SQL query is sent to the database and the id, or number, of the next challenge is fetched. With this, the next challenge is fetched.

A new challenge is created and the attributes received from the challenge table are assigned to the challenge. To ensure that the chance of a null pointer exception is reduced, I opted to use optionals in this method. If the SQL query fails or does not work as expected, an empty optional is returned, otherwise an Optional of the challenge is returned. In the rest of the code a simple Optional.ifPresent can be used to check that the optional is not null and that the challenge is there. The challenge is then retrieved into type Challenge by using Optional.get, simple!

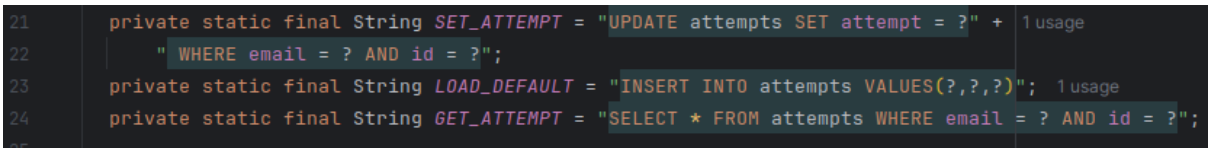


```
    Fetches the student's next challenge and returns it as an optional challenge
    Params:  userEmail – the user's email
    Returns: optional of challenge
27  public Optional<Challenge> getNextChallenge(String userEmail) {  1 usage  Tom *
28
29      try (Connection conn = WicketApplication.getConnection();
30           PreparedStatement stmt =
31               conn.prepareStatement(GET_CURRENT_PROGRESS)) {
32          stmt.setString( parameterIndex: 1, userEmail);
33          ResultSet resultSet = stmt.executeQuery();
34          if (resultSet.next()) {
35              Challenge challenge = new Challenge();
36              challenge.setId(resultSet.getInt( columnIndex: 1));
37              challenge.setTitle(resultSet.getString( columnIndex: 2));
38              challenge.setDescription(resultSet.getString( columnIndex: 3));
39              challenge.setSolutionTemplate(resultSet.getString( columnIndex: 4));
40
41              return Optional.of(challenge);
42
43          }
44          return Optional.empty();
45      } catch (SQLException e) {
46          Logger.getLogger(WicketApplication.class.getName()).log(Level.SEVERE, e.getMessage());
47          throw new RuntimeException(e);
48
49      }
50  }
```

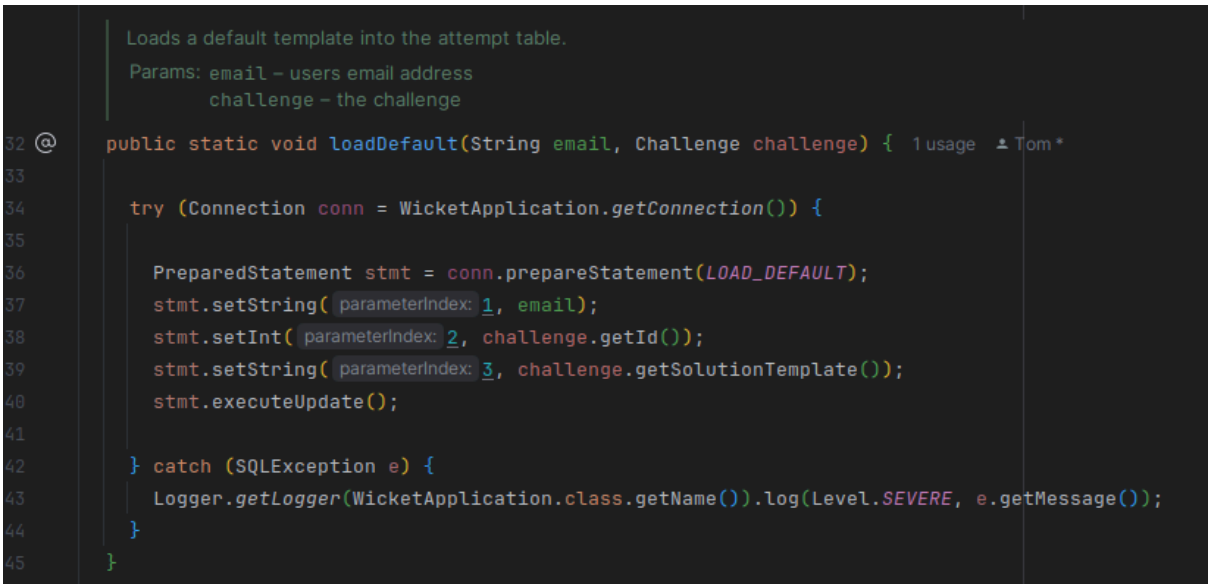
MySQLAttemptsStore

This class holds the methods that bridge the attempts from the database to the website. I feel like I'm repeating myself a little bit here…

Here they are, the queries that are being used stored as private static final strings, or constants at the top of the class.



The first method is loadDefault and prepares the Attempts table with a student template ready to load the first time they log in.

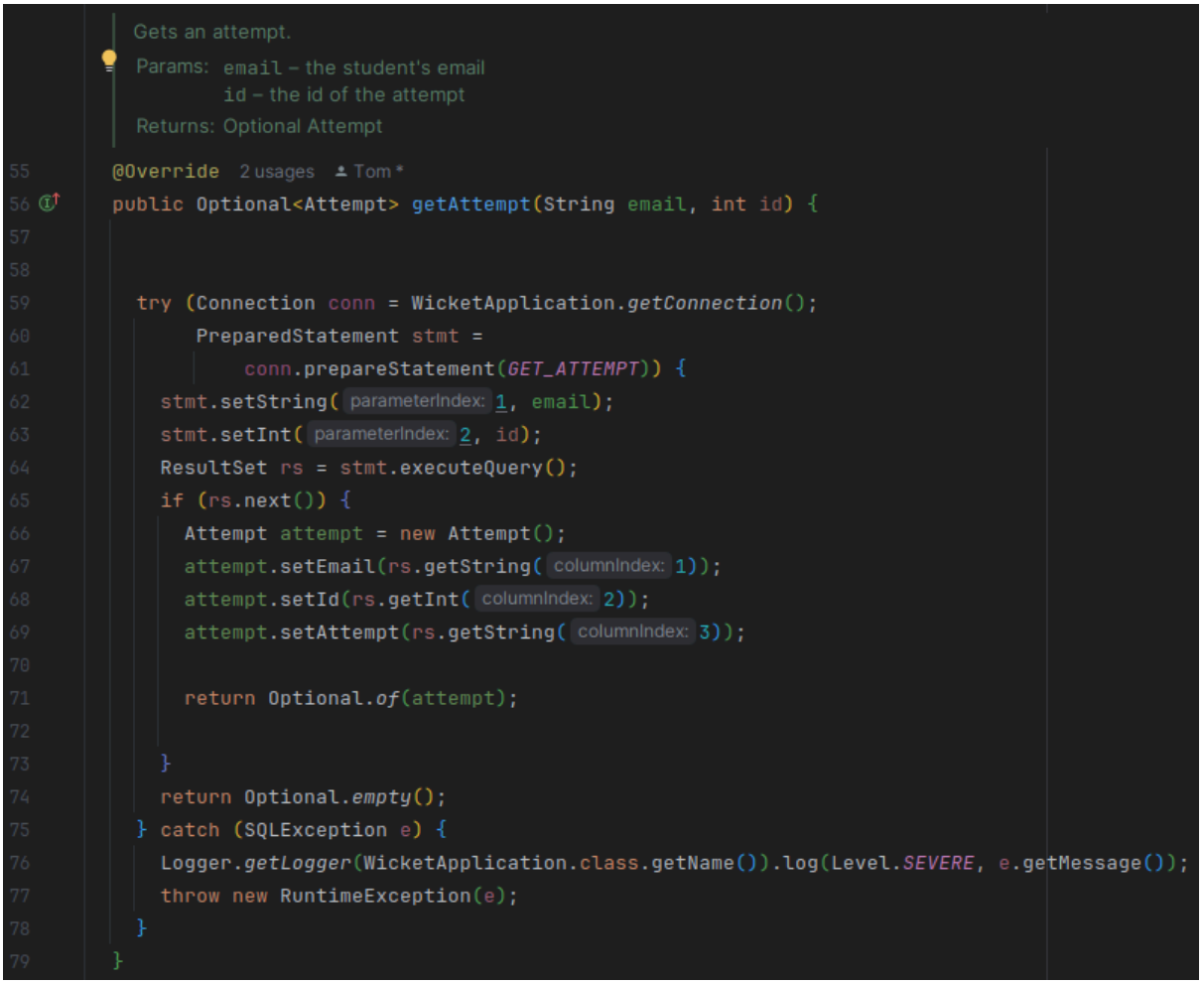


This method should only be run once per user as it creates a clean slate by fetching the template from the challenges table and placing it in the attempt attribute for their record.

Again, to minimise the risk of things breaking, the SQL is run inside a try-catch statement where if an error is caught, it is printed out to the console.

I could've used $printStackTrace, however, using the Logger is a much more robust method of logging.

The next method is the getAttempt method, and the clue is in the name, it gets an attempt from the user given the email of the user and the id of the attempt.



```
    Gets an attempt.
    Params: email – the student's email
            id – the id of the attempt
    Returns: Optional Attempt
55  @Override  2 usages  Tom *
56  public Optional<Attempt> getAttempt(String email, int id) {
57
58
59      try (Connection conn = WicketApplication.getConnection();
60           PreparedStatement stmt =
61                   conn.prepareStatement(GET_ATTEMPT)) {
62          stmt.setString( parameterIndex: 1, email);
63          stmt.setInt( parameterIndex: 2, id);
64          ResultSet rs = stmt.executeQuery();
65          if (rs.next()) {
66              Attempt attempt = new Attempt();
67              attempt.setEmail(rs.getString( columnIndex: 1));
68              attempt.setId(rs.getInt( columnIndex: 2));
69              attempt.setAttempt(rs.getString( columnIndex: 3));
70
71              return Optional.of(attempt);
72
73          }
74          return Optional.empty();
75      } catch (SQLException e) {
76          Logger.getLogger(WicketApplication.class.getName()).log(Level.SEVERE, e.getMessage());
77          throw new RuntimeException(e);
78      }
79  }
```

Note the use of optionals here again to minimise the chance of a null pointer exception.

This method is quite similar to the getChallenge method in the MySQLChallengeStore in that given certain parameters, it is able to construct a new object, in this case of type Attempt, and return it.

The final method in this class is the setAttempt method. This method takes a String, the student's email, an int, the id of the challenge and a String, which is the student's attempt itself.

```
    Adds a students attempt to the database
    Params: email – the student's email
            id – the id of the attempt
            attempt – the students attempt at the challenge
87  @Override  1 usage  Tom *
88  public void setAttempt(String email, int id, String attempt) {
89
90      try (Connection conn = WicketApplication.getConnection()) {
91
92          PreparedStatement stmt = conn.prepareStatement(SET_ATTEMPT);
93          stmt.setString( parameterIndex: 2, email);
94          stmt.setInt( parameterIndex: 3, id);
95          stmt.setString( parameterIndex: 1, attempt);
96          stmt.executeUpdate();
97
98      } catch (SQLException e) {
99          Logger.getLogger(WicketApplication.class.getName()).log(Level.SEVERE, e.getMessage());
100     }
101 }
```

This method is logically quite simple and adds a student's attempt to the database. The reason I decided to do this is to create a sort of save-as-you-submit system. This means that if a student were to close the webpage after submitting an attempt, it would not be lost.

Alpha testing

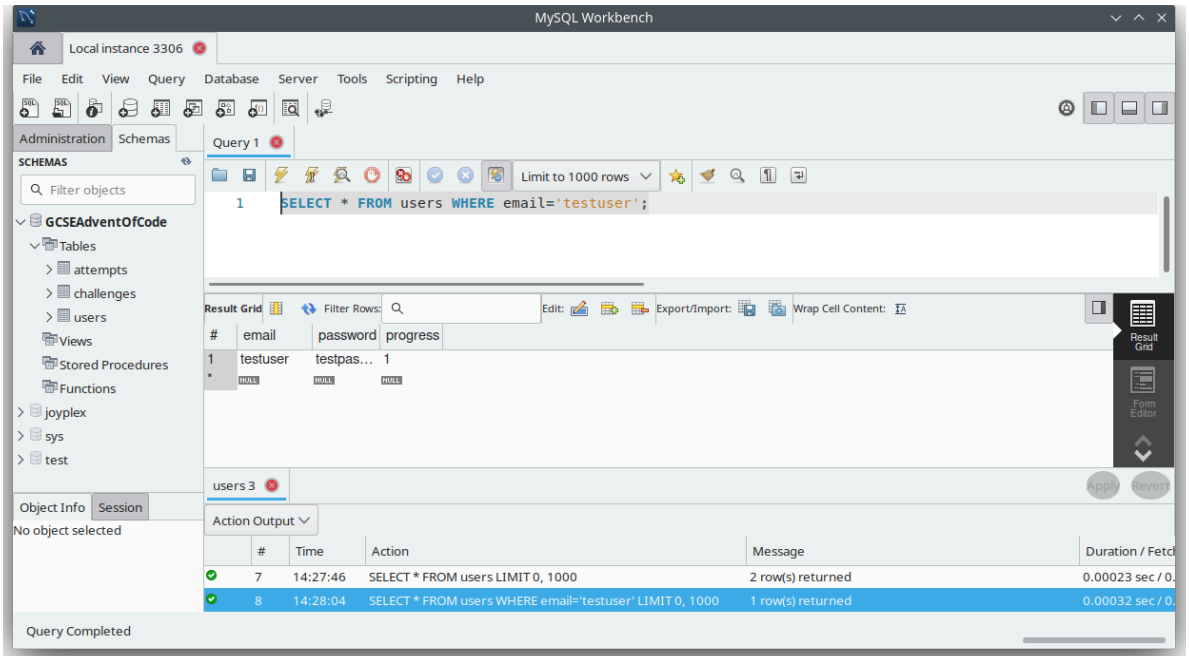
For each table I used SQL statements to test the database that had been created to ensure that data items could be added to the database and that duplicate items were not possible. The first query I used for each scheme was to add a new row. The second was to ensure that the query had been executed correctly and that the scheme had been updated. The final query was exactly the same as the first and ensured that duplicate items could not be added to the schema.

Users table

```
INSERT INTO users values ('testuser', 'testpassword', 1);
```

```
1 row(s) affected
```

```
SELECT * FROM users WHERE email='testuser';
```



First query rerun result:

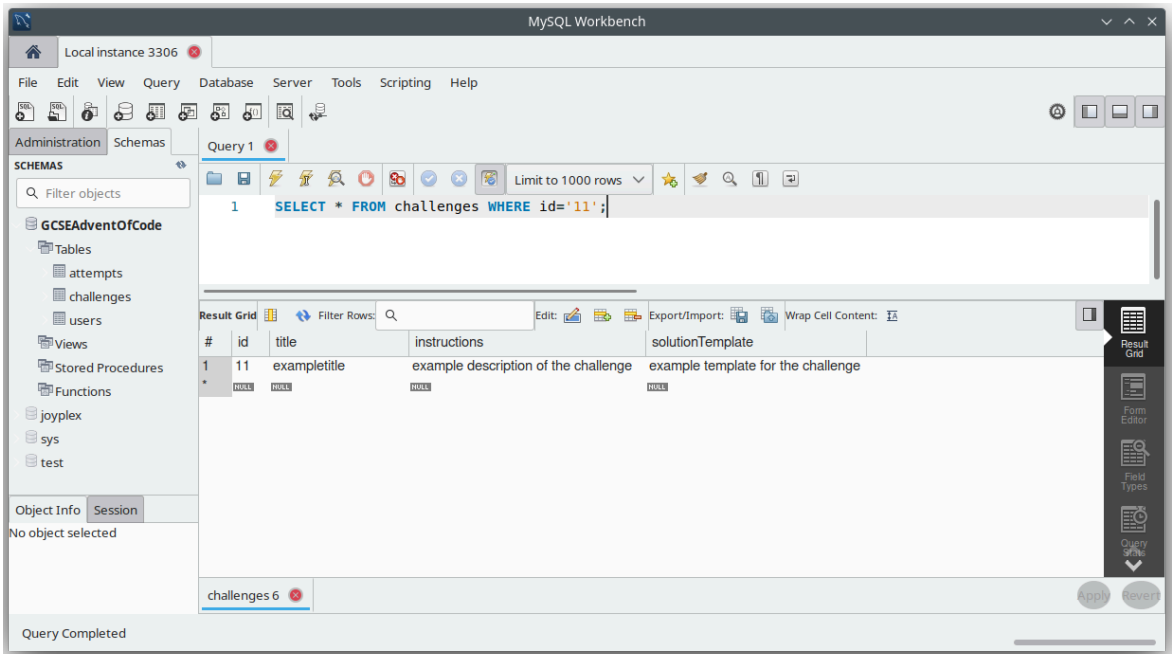
```
Error Code: 1062. Duplicate entry 'testuser' for key
'users.PRIMARY'
```

Challenges table

```
INSERT INTO challenges values (11, 'exampletitle', 'example description of the challenge', 'example template for the challenge);
```

```
1 row(s) affected
```

```
SELECT * FROM challenges WHERE id='11';
```



First query rerun result:

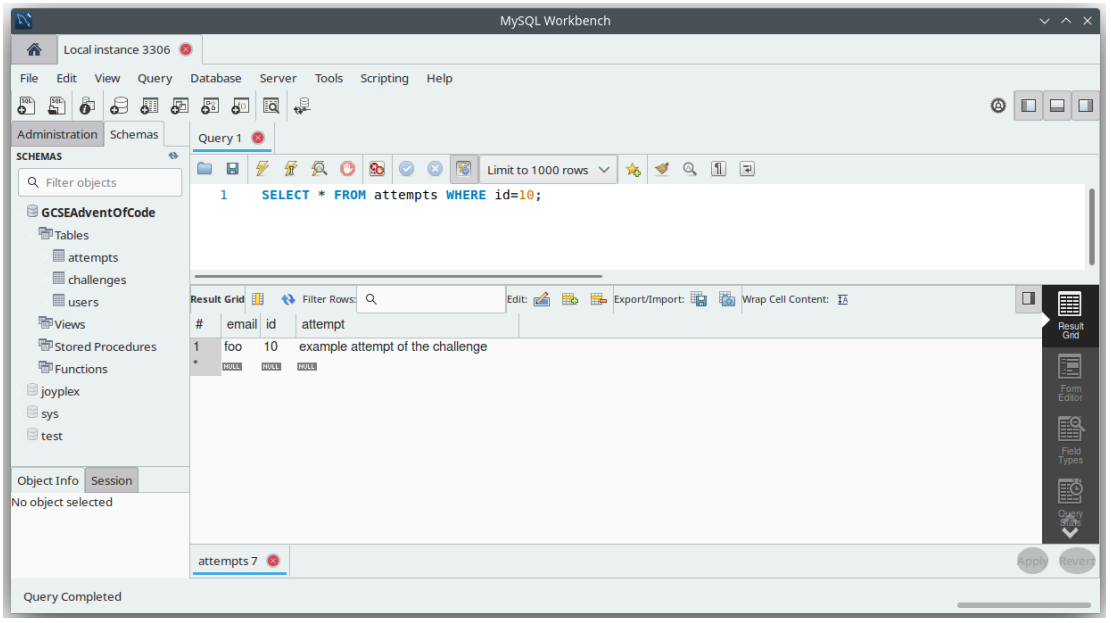
```
Error Code: 1062. Duplicate entry 'testuser' for key 'users.PRIMARY'
```

Attempts table

```
INSERT INTO challenges values ('foo', 10, 'example attempt of the challenge');
```

```
1 row(s) affected
```

```
SELECT * FROM attempts WHERE id='11';
```



First query rerun result:

```
Error Code: 1062. Duplicate entry 'testuser' for key 'users.PRIMARY'
```

Stakeholder feedback

The teachers and students using the system could not tell the difference between when the system was using in memory data storage and the database. As this prototype did not affect much of the visual aspects of the system there was no more comments from either students or teachers.

Validation

I have used validation in many different aspects of my project ranging from password input to the code given as solutions to code.

Validation is required to ensure that the data that I am using within my database is of the type and size that I need. This way I can ensure I know what types I am being given in my code, prevent errors in the program, and know that the data a student inputs is correct.

It is required on both my sign-up and sign-in pages because the student asked to input an email address and a password. Without an external connection, it can be challenging to determine whether an email exists or not. To counter this I used regex to test whether the entered email matches the pattern of a normal email address, which in my case is (string)@(string).(string). The regex I used for this is as follows:

^(?=.{1,64}@)[A-Za-z0-9_-]+(\.[A-Za-z0-9_-]+)*@[^-][A-Za-z0-9-]+(\.[A-Za-z0-9-]+)*(\.[A-Za-z]{2,})$

Preventing attacks

A common weakness with input boxes is the ability to SQL Inject. This is where the user, or in this case a student, enters their own SQL query into the input box. The effects of this can be a log-in with an incorrect password if the student were to enter “password OR 1=1”, or if they were to input “password; DROP DATABASE GCSEAdventOfCode”, I would lose all data ever inputted into my database.

There are a few ways of preventing SQL statements from being executed, one way is by sanitising data inputs and another is by using Prepared Statements. Sanitising your data inputs is where you remove or escape any characters that could be interpreted as SQL, such as a semicolon. The issue with this is, that today, people use random characters to make their password secure, which includes semicolons so their passwords may become broken with sanitization. Another option, which I opted for, was Prepared Statements. These work very simply by sending the query and the data to the database separately. This prevents any chances of SQL injection as the data that they have inputted is never really executed. This way means that the user can still use secure characters without the chance of errors.

Error log

Error #1 - SQLIntegrityConstraintViolationException

The error, a java.sql.SQLIntegrityConstraintViolationException was being thrown every time I started, or restarted my program.

```
java.sql.SQLIntegrityConstraintViolationException Create breakpoint : Duplicate entry '1' for key 'challenges.PRIMARY'
    at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:117)
    at com.mysql.cj.jdbc.exceptions.SQLExceptionsMapping.translateException(SQLExceptionsMapping.java:122)
    at com.mysql.cj.jdbc.ClientPreparedStatement.executeInternal(ClientPreparedStatement.java:916)
    at com.mysql.cj.jdbc.ClientPreparedStatement.execute(ClientPreparedStatement.java:354)
    at org.swainston.WicketApplication.initDB(WicketApplication.java:119)
    at org.swainston.WicketApplication.init(WicketApplication.java:60)
```

The error had been caused by the following section of code, within my setupdb.sql file.

```
24  INSERT INTO challenges values (1,
25          'Average speed check',
26          'Create a program that calculates the average speed, in mph, between two
27           speed cameras that are exactly 1 mile apart. You will be given
28           two times in epoch in order of each camera.'
29             , '1');
30
```

The error stemmed from the fact that this file was run every time my project started or restarted, due to the following block of code.

```
    Initialises the database and uses the setUpDB.sql to populate the database.
    Throws: SQLException – if prepared statement fails to execute.
            IOException – if content within the SQL file is invalid.
118 private void initDB() throws SQLException, IOException {  1 usage   Tom *
119   String setup;
120   try (InputStream inputStream = getClass().getClassLoader().getResourceAsStream( name: "setUpDB.sql")) {
121     if (inputStream != null) {
122       setup = new String(inputStream.readAllBytes());
123     } else {
124       throw new AssertionError( detailMessage: "Failed to read bytes of resource.");
125     }
126   } catch (IOException ex) {
127     Logger.getLogger(WicketApplication.class.getName()).log(Level.SEVERE, ex.getMessage());
128     return;
129   }
```

The “INSERT” commands within my SQL code were therefore being run, even if the database table was already populated. This made matters even more confusing as this error did not initially occur when I wrote the code as the table had not been populated.

I researched the error and I quickly found a solution on https://stackoverflow.com/questions/812437. The solution was simple, add the keyword “IGNORE” following “INSERT” for all of my challenges.

The IGNORE keyword in SQL allows the statement to be run and returns a warning, instead of an error if it is a duplicate entry.

```
24  INSERT IGNORE INTO challenges values (1,
25          'Average speed check',
26          'Create a program that calculates the average speed, in mph, between two
27           speed cameras that are exactly 1 mile apart. You will be given
28           two times in epoch in order of each camera.'
29             , '1');
30
31  INSERT IGNORE INTO challenges values (2,
32          'Code cracker',
33          'You''ve forgotten your 4-digit phone password, however, you remember
34  which numbers it included. You will be given these numbers and you must
35  output every possible combination. Let''s hope your phone doesn''t keep count
36  of attempts...', '2');
```

After adding these and compiling, there were no fatal errors in the console but I was expecting some warnings, however, there were none and only my coded alert said that the file had been run correctly and the database had been started.

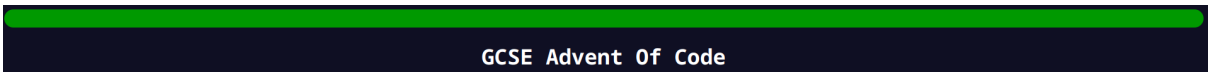
```
[main] INFO org.eclipse.jetty.server.session - node0 Scavenging every 600000ms
[main] INFO org.apache.wicket.util.file.WebXmlFile - web.xml: url mapping found for filter with name wicket.GCSEAdventOfCode: [/*]
[main] INFO org.apache.wicket.Application - [wicket.GCSEAdventOfCode] init: Wicket core library initializer
Database loaded & ready!
[main] INFO org.apache.wicket.protocol.http.WebApplication - [wicket.GCSEAdventOfCode] Started Wicket version 9.14.0 in DEVELOPMENT mode
```

Error #2 - Content Security Policy

I had initially created the front-end pages using Visual Studio code, and once they were created I opened them using their file path, for example, filesystem/home/website/index.html meaning that there was no security.

The icons that I used to create my navigation bar were from an external site, FontAwesome. This meant that my header contained links to external sites. Apache Wicket did not approve of this due to its strong security. Having external sites in the header can mean that a site is fraudulent or dangerous therefore it blocks links to the sites, subsequently removing my icons from the navigation bar.

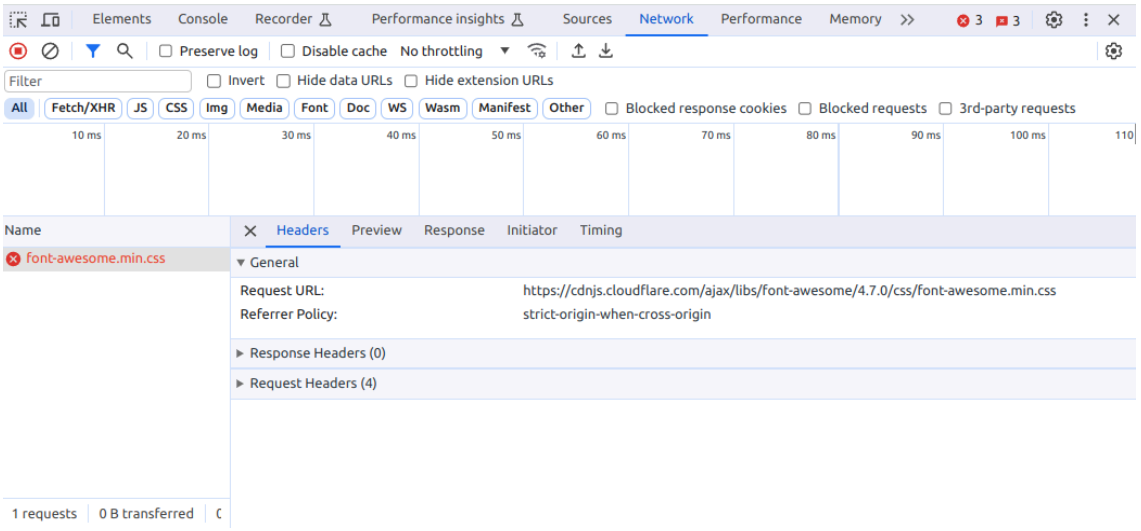
My navigation bar looked as follows when I tried to run it on IntelliJ using Apache Wicket for the first time:



Although the icons had disappeared, the button was still functional and there, as seen below so this was not an error that caused anything to break per se.

I had an idea that it would've been related to the fact that the website was now being hosted locally, and not just being opened from a file. I decided to use chrome's inspect element to allow me to see the network headers.

The following was displayed after I used CTRL+ALT+I, selected "Network" and looked under "headers".



As you can see in the image, the font-awesome.min.css is being blocked, meaning that the library that my icons are contained in is being blocked, and thus not showing my icons.

The solution to this required some digging into CSP, or “Content Security Policy”. I was lucky to find the solution, and code examples to fix my issue quite quickly. The solution was found on the official content security policy website, at this URL:

https://csplite.com/csp202/

Within the page, it suggested adding the following to my application page, in my case, WicketApplication.java.

To configure Content Security Policy you should do something like:

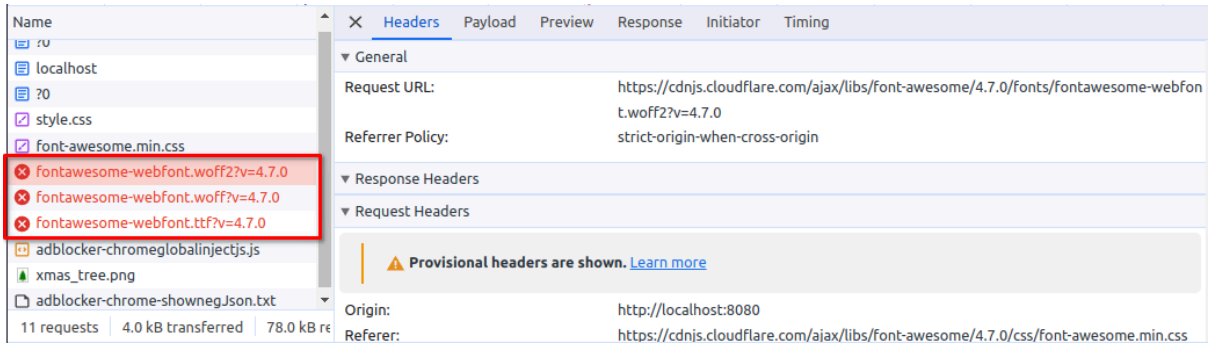
```
myApplication.getCspSettings().blocking().clear()
    .add(CSPDirective.DEFAULT_SRC, CSPDirectiveSrcValue.NONE)
    .add(CSPDirective.STYLE_SRC, CSPDirectiveSrcValue.SELF)
    .add(CSPDirective.SCRIPT_SRC, CSPDirectiveSrcValue.SELF)
    .add(CSPDirective.IMG_SRC, "'self' data:")
    .add(CSPDirective.FONT_SRC, "https://maxcdn.bootstrapcdn.com");
```

I could tell this was what I needed because, in the final line, you are able to see that it is allowing an external style sheet from an external link, which is exactly what I was doing which was causing the problem.

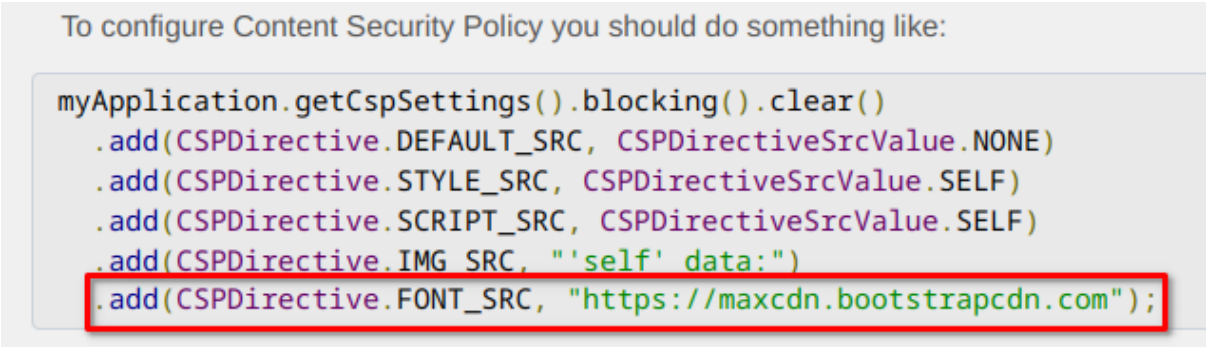
I added the following code in my WicketApplication.java file and ran the website again.

```
66      // Required to allow custom fonts to be displayed and not be blocked by Content Security Policy
67      getCspSettings().blocking()
68          .add(CSPDirective.STYLE_SRC, CSPDirectiveSrcValue.SELF)
69          // google fonts
70          .add(CSPDirective.STYLE_SRC,  ...values: "https://fonts.googleapis.com/css")
71          // FontAwesome fonts stylesheet
72          .add(CSPDirective.STYLE_SRC,
73              ...values: "https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css");
```

Unfortunately, this did not fix the problem, but it definitely got me closer to a solution, as the navbar was no longer small, and there were now three errors in the network headers.



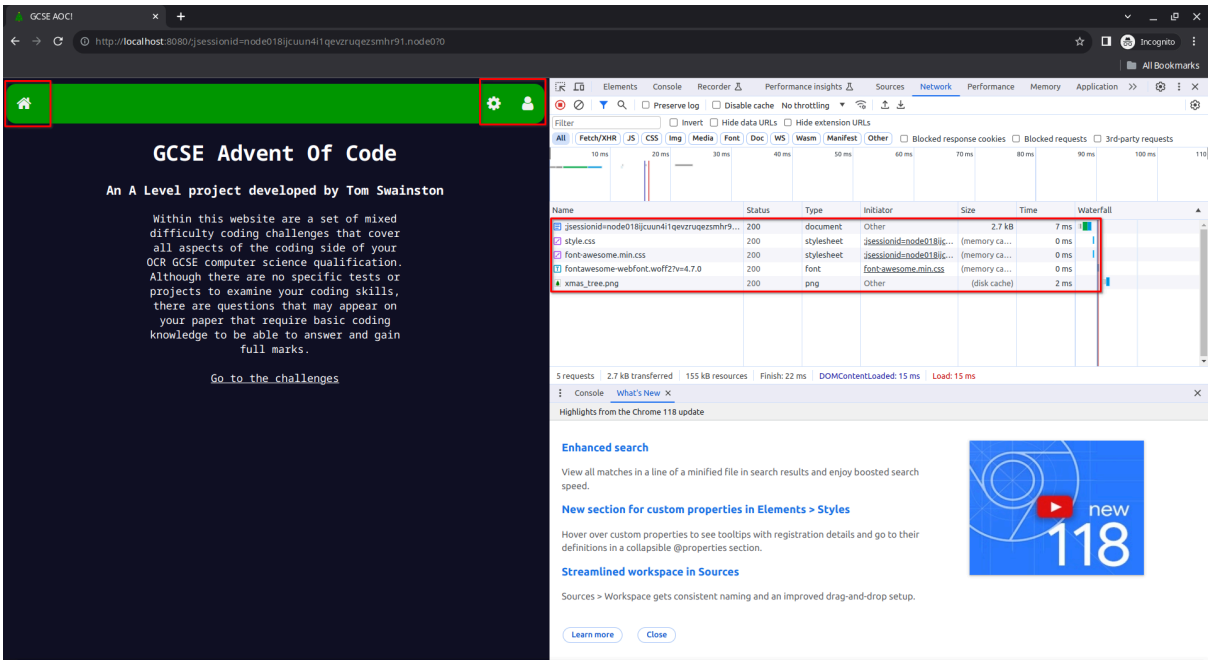
As you can see, there is now one error for each item in the nav bar. On the CSP website, in the same block of code that I used before, you can see there is an option to add a font source as FONT_SRC, as seen below.



I used this template for each request URL that was giving an error to create the following.

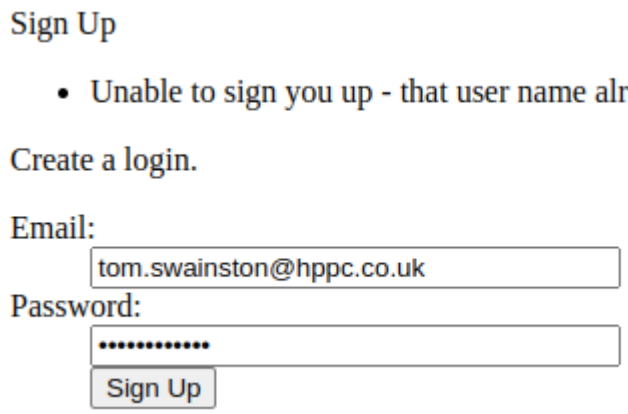
```
66      // Required to allow custom fonts to be displayed and not be blocked by Content Security Policy
67      getCspSettings().blocking()
68          .add(CSPDirective.STYLE_SRC, CSPDirectiveSrcValue.SELF)
69          // Google fonts
70          .add(CSPDirective.STYLE_SRC,  ...values: "https://fonts.googleapis.com/css")
71          // FontAwesome fonts stylesheet
72          .add(CSPDirective.STYLE_SRC,
73                  ...values: "https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css")
74          // FontAwesome custom font
75          .add(CSPDirective.FONT_SRC,
76                  ...values: "https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/fonts/fontawesome-webfont.woff2")
77          .add(CSPDirective.FONT_SRC,
78                  ...values: "https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/fonts/fontawesome-webfont.woff")
79          .add(CSPDirective.FONT_SRC,
80                  ...values: "https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/fonts/fontawesome-webfont.ttf");
```

After adding this, I restarted the project, and opened the website and the issue appeared to have been resolved. The nav bar was there and functional, and there were no errors under the network.



Error #3 - Sign up logical flow error

This was a slightly odd error that occurred whenever I tried to create a new account for a student. The problem that occurred is that when you would input login credentials for the website, it would successfully add credentials to the database, given that there were no duplicates, however, it would send a message to the webpage saying that the account already existed, and it could not be created. As seen below



The source of this error was within the SignUpPage class, and was quite a silly logical error in code that must've been written whilst half asleep. The error was caused by the fact that the users credentials were added to the credential store before checking whether they existed within the credential store, so in whatever case, any new credentials inputted into the box would already exist by the time the front-end checks to see whether they exist.

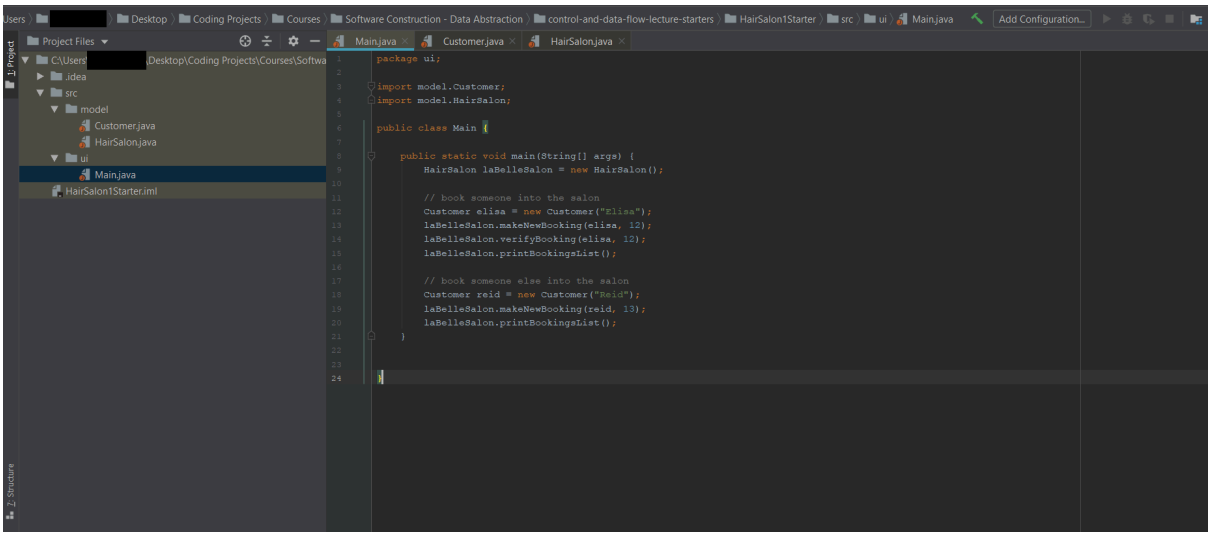
The fix to this error was quite simple, and consisted of a reordering of if-statements, to create a correct logical flow. As seen below.

```
122         SignInSession signInSession = getMySession();
123
124         // Check if the sign-up is successful
125         if (signInSession.signUp(username, password)) {
126
127           // Sign the user in
128           if (signInSession.signIn(username, password)) {
129             continueToOriginalDestination();
130           } else {
131             // Get the error message from the properties file associated with the Component
132             String loginError = getString( key: "loginError",  model: null,  defaultValue: "Sign in details not recognised.");
133
134             // Register the error message with the feedback panel
135             error(loginError);
136           }
137         } else {
138           // Get the error message from the properties file associated with the Component
139           String errorMessage;
140           if (signInSession.userExists(getUsername())) {
141             errorMessage = getString( key: "loginError",  model: null,
142                 defaultValue: "Unable to sign you up - that user name already exists");
143           } else {
144             errorMessage = getString( key: "loginError",  model: null,  defaultValue: "Unable to sign you up");
145           }
146           // Register the error message with the feedback panel
147           error(errorMessage);
```

Error #4 - Broken modules and faulty merge

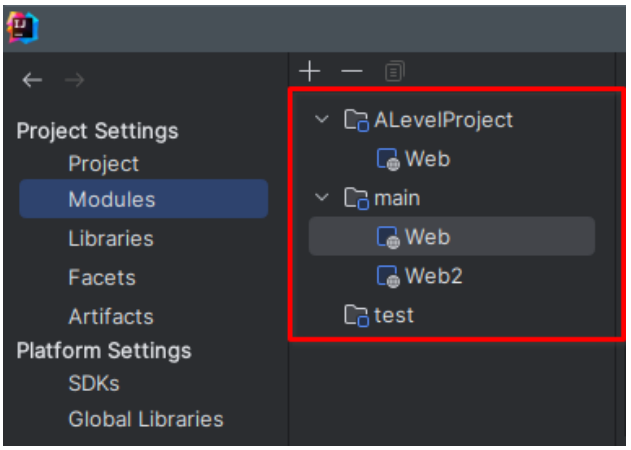
This error occurred more towards the end of my project, near evaluation and just before robustness testing. This error was less of a code error, but more of a hosting error and an IntelliJ issue, I like to think…

One morning I opened my project and all of my files down the project dropdown of IntelliJ IDEA had turned orange, similar to this image that I found on the internet.



I hovered over each file and it said “Module not specified”, which was odd as I had not touched the project structure at all, at least for weeks previous to this and the project was working the day before.

I opened my project structure to see something very odd. All of my modules had vanished… This is what the modules section looks like now the error has been addressed.



The red box seen in the image was completely empty and all of my modules had vanished! Peculiar.

I'm not totally sure what caused this to happen, but I will assume it was a dodgy update of some sort that broke the project structure, and I was unable to identify which update had caused this.

I searched on the internet my problem and this thread popped up.

https://intellij-support.jetbrains.com/hc/en-us/community/posts/360003582959-Java-files-are-orange-cannot-run-program

This thread had some viable fixes, however, none worked and I was even more confused as to what had happened to my project.

I decided it would be best to manually add each module back by hand, which took me a while, but I eventually finished.

After doing all that, something odd started to happen and all of my Jetty code had syntax highlighting with an error stating that methods did not exist in the Jetty library, even though this code had been run multiple times before with no issues at all.

In my pom.xml, the configuration file for maven, nothing appeared to be out of the ordinary. The Jetty dependency, seen below, had no errors pointing towards an out of date or deprecated version.

```
102         <dependency>
103             <groupId>org.eclipse.jetty</groupId>
104             <artifactId>jetty-http</artifactId>
105             <version>10.0.11</version>
106         </dependency>
```

I had a look back at GitHub commits, and it turns out a slightly dodgy merge had resulted in a set version of Jetty being put into "version" to combat this I reverted the merge, just in case other problems had been caused by this faulty merge, and it revered to the Jetty 9 version, set as a constant further up in the file.

```
96          <dependency>
97              <groupId>org.eclipse.jetty</groupId>
98              <artifactId>jetty-http</artifactId>
99              <version>${jetty9.version}</version>
100         </dependency>
101
```

Evaluation

There were many criteria within the success criteria that were met fully, and a few that were met partially. Therefore I am happy to say that the project was a partial success.

The success criteria

Criteria to be met	How it will be measured

A website that can be used in the most common internet browsers, devices and operating systems without any problems or decreased speed	• Open and explore the website on different internet browsers such as:
	  ○ Chrome
	  ○ Firefox
	  ○ Internet Explorer
	  ○ Safari
	• Open the website on different devices, such as:
	  ○ Surface Pro
	  ○ Mobile Phone
	  ○ iPad
	  ○ Laptop
	  ○ Personal Computer
	• Open the website on the following operating systems
	  ○ Linux (Unix)
	  ○ Linux (Chrome)
	  ○ iOS
	  ○ macOS
	  ○ Windows 10/11

A solution that assists students in gaining the most amount of marks in the coding aspect of their course and final exam	• Analyse end-of-year exam results for students who used this application and students who didn’t • Let the students sit a coding exam before and after using this solution
Securely hold the students' progress and login information in a database that is secure from breaches and leaks	• Attempt the most common methods of attack on the solution • Hire white box and black box testers to attempt to penetrate

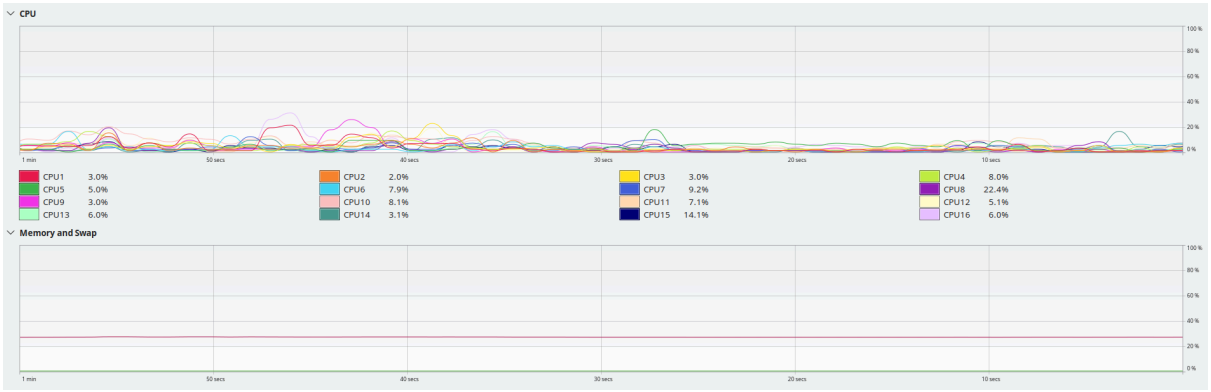
	my levels of security
A secure teacher portal that allows teachers to view certain student's progress and manage their account	● Using a teacher account: ○ Attempt to initiate a password reset on a student ○ Attempt to select a student and see if you can retrieve their challenge data ○ Attempt to delete a student’s account and ensure that a 2FA prompt is received

Criterion #1

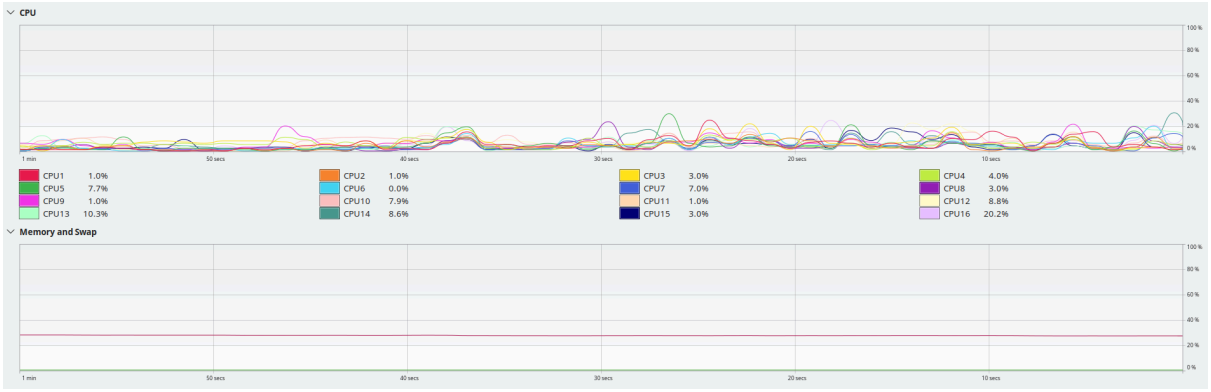
“A website that can be used in the most common internet browsers, devices and operating systems without any problems or decreased speed”

I am confident in saying that this criterion **has been met.**

To test whether the website used excess data, I firstly turned on the Jetty server to ensure that that was not a factor in the system resources. I let the system monitor measure for 1 minute and this was the outcome.



After opening the website and attempting a challenge this is what the system monitor was showing



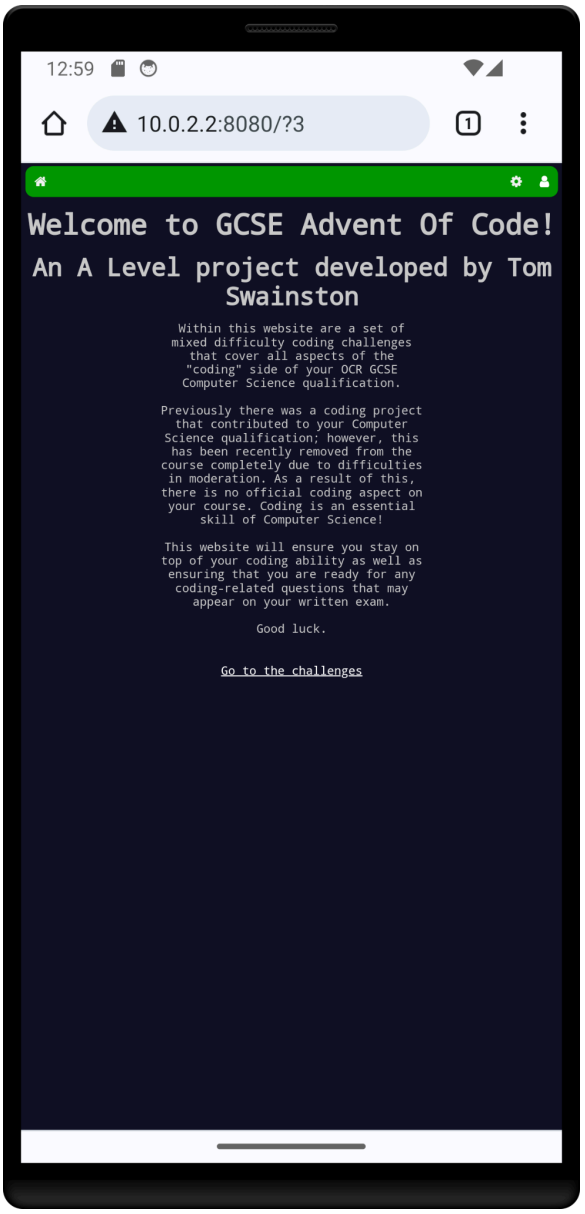
As you can see there is a very minimal difference, almost negligible difference on the system monitor. I am therefore happy to say that it does not use up large amounts of processing power or memory

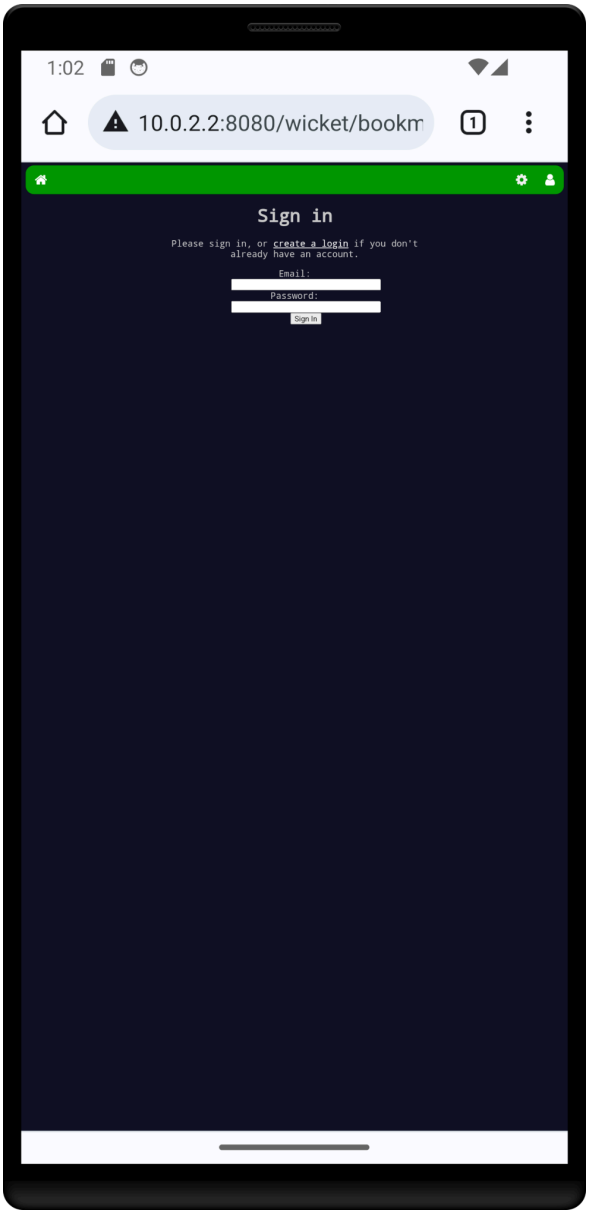
That is the general criteria met, however, in the “ways that it will be measured” there are a few specific criteria that it mentions, such as being able to run the project on different operating systems and browsers.

The website was accessible through phones, however, due to the portrait view from a phone and the inconvenient keypad, I would not recommend using this project on a phone.

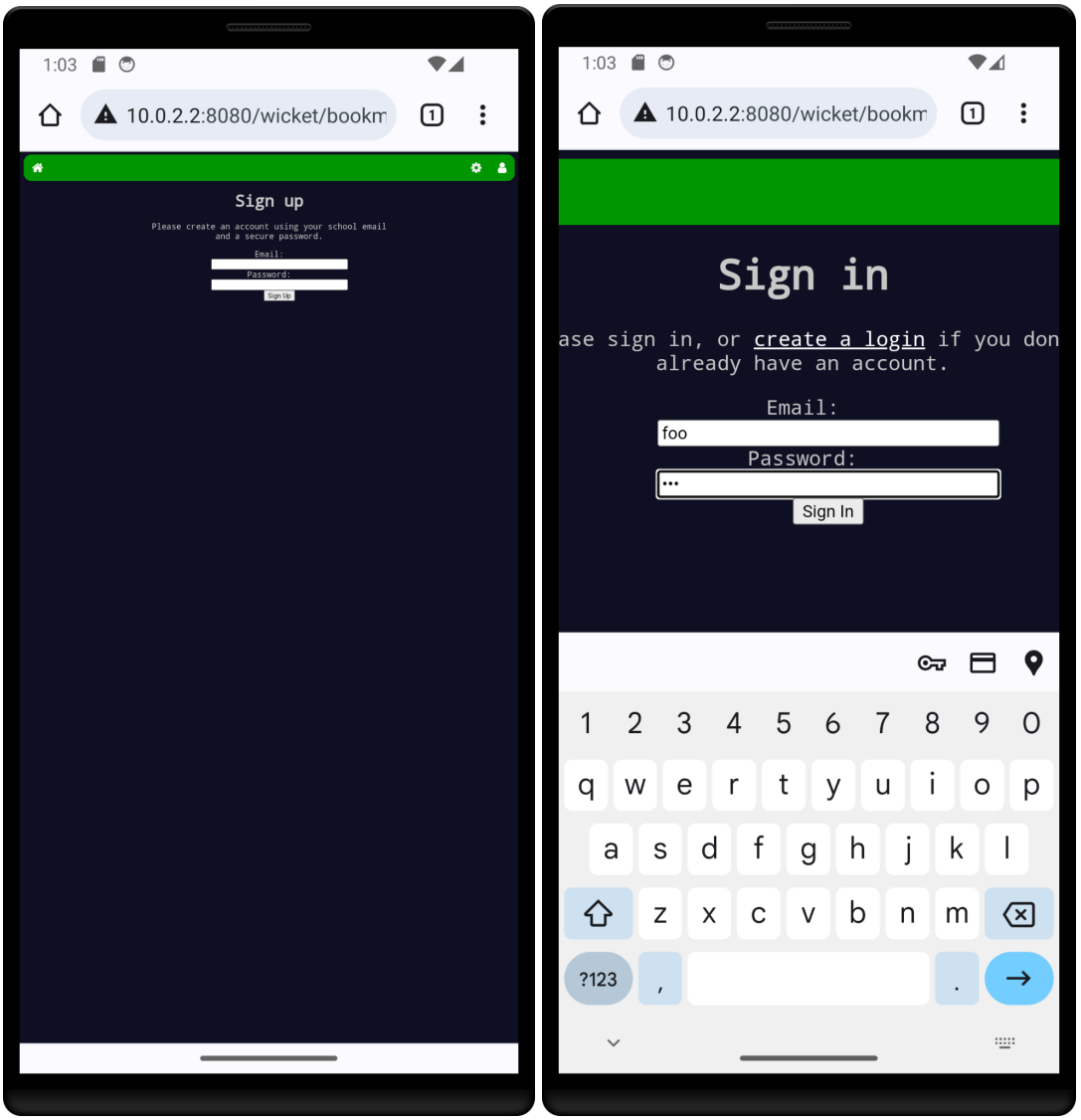
To run the website on a phone I decided to use an emulator from Android Studio. I used a Pixel 3 emulator with Android Oreo (Android version 8). To access the localhost website from the emulator I used the loopback address 10.0.2.2:8080.

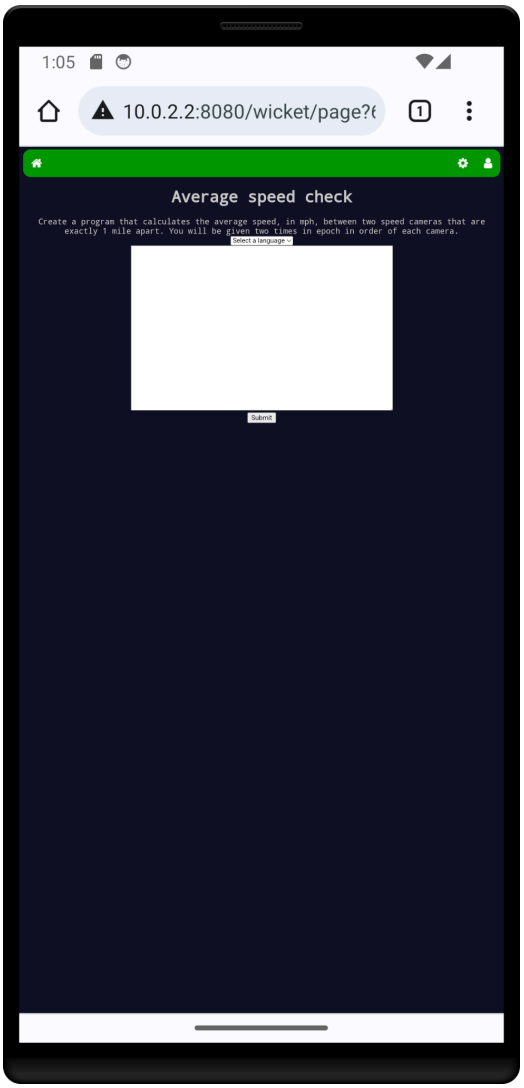
Below is the homepage and the sign in page when the phone is in portrait mode.





Below is the sign-in page when fully zoomed out and the view when the text boxes are selected in portrait mode.

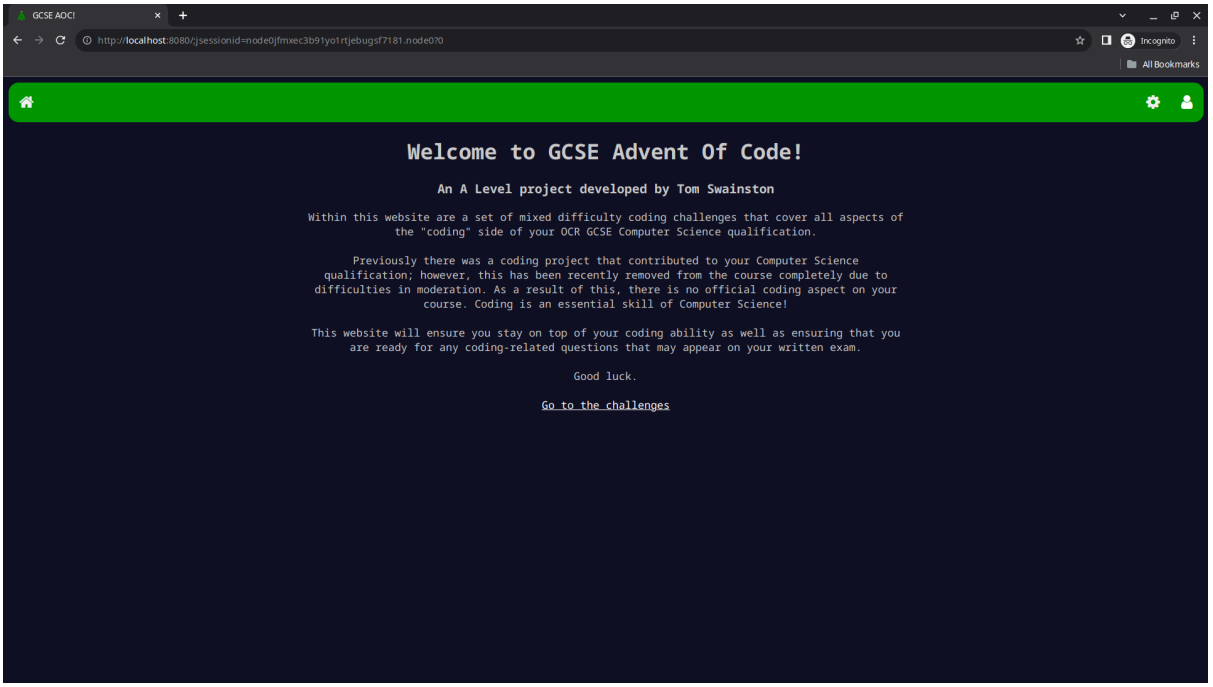


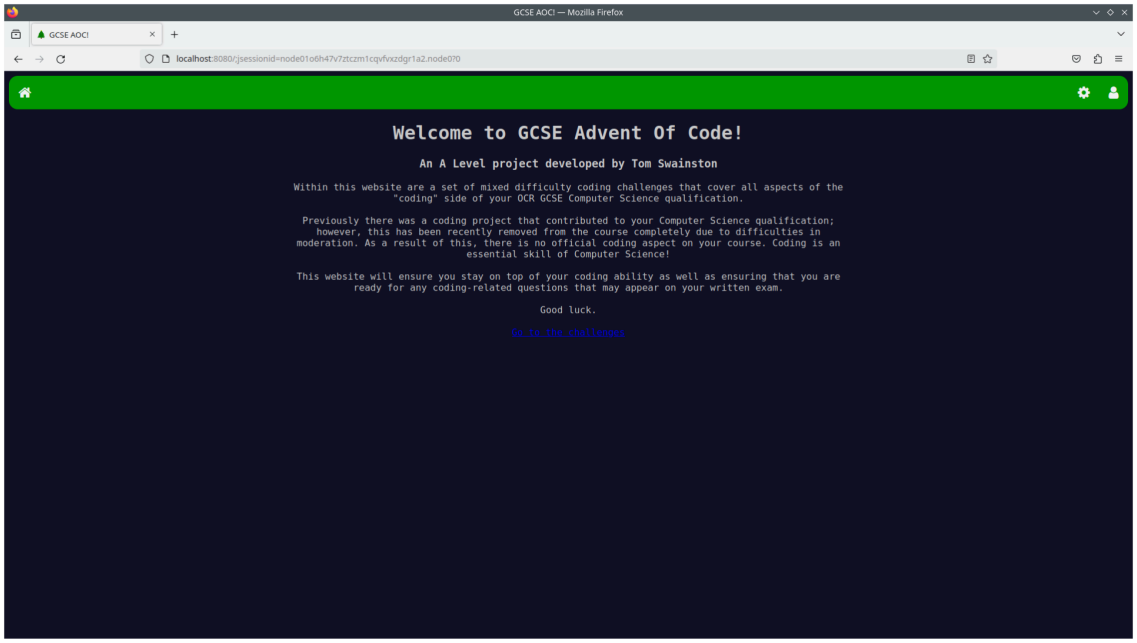


This is the challenge page in portrait mode

Here are landscape view of the homepage and sign-in page

I was able to successfully run the website on Chrome, Firefox and Internet Explorer.



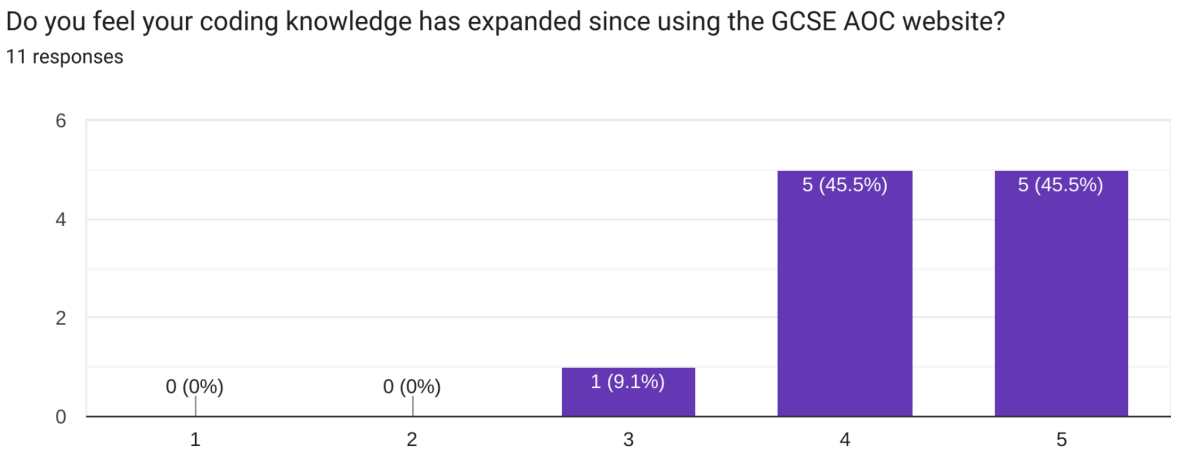


Criterion #2

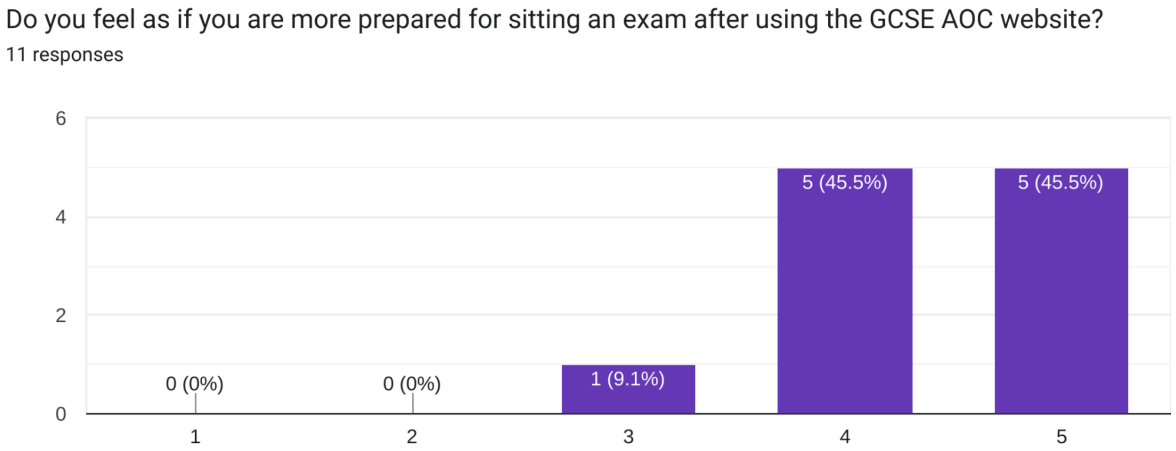
“A solution that assists students in gaining the most amount of marks in the coding aspect of their course and final exam”

Unfortunately, this is quite a difficult criterion to measure without the project being publicly available for a large portion of time. In this case, I’ve focused on the second part of the measurement recommendation of “Let the students sit a coding exam before and after using this solution”.

Student feedback form

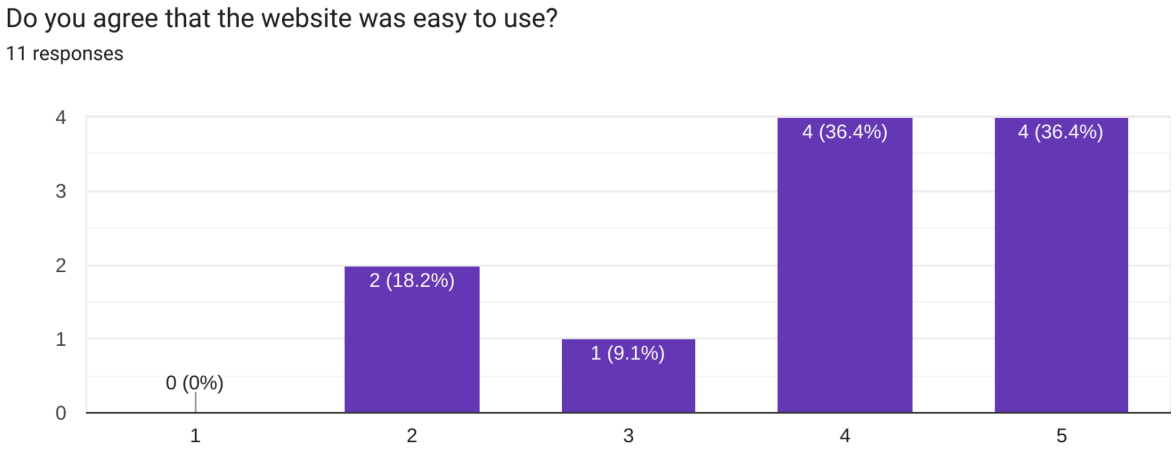


From this chart it is clear to see that most of the students felt that their coding knowledge had been expanded after using the website and completing some challenges. Although this is not solid data and does not show whether their coding knowledge has really improved, it is quite a clear indicator as to what has happened.

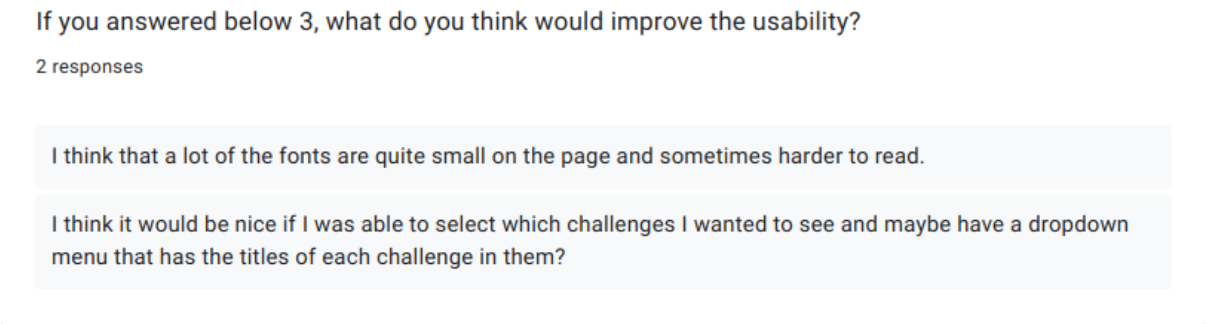


One of the key aims of my solution was to prepare students for exams. From this graph it is clear to see that all students benefited from using the solution and they feel more prepared for sitting an exam. It's also clear to see that a large portion of the students benefited greatly from using the solution!

Useability testing



A slightly more spread out set of results here, I think it's obvious that the website in general is usable and fairly straightforward to navigate, however, there are definitely improvements to be made, and below are two comments from the students.



I think I agree with the students in both cases, and that the changing of the font sizes would be pretty easy to implement. The second one would also not be super difficult to implement but would definitely need some time and more stakeholder feedback to design the menu itself and where it will be placed on the challenge page itself.

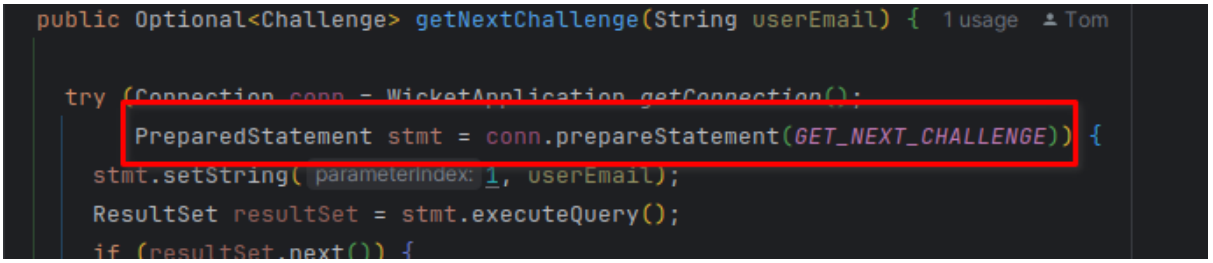
I was unable to have a student sit an exam before or after using the application, however, I sent a challenge across to a student and had them attempt it. The students attempted it, conducting their own research online and eventually produced a solution to the challenge. I asked the student after whether they thought doing this challenge had expanded their knowledge, and the student agreed. They said that they now understood how to use the internet properly to help with coding related issues and that it had secured knowledge relating to a specification point for their GCSE. I am happy to therefore confidently say that this criterion **has been met**.

Criterion #3

“Securely hold the students' progress and login information in a database that is secure from breaches and leaks”

Now this criterion is possibly the most important, and the most likely to not be met. As students are able to input their own code into the program, this could risk security breaches.

The first possible breach could be through SQL Injection. This basic method of breaching data has clearly been looked into and prevented in the project. This is thanks to the use of PreparedStatements for all SQL queries, as shown below.



With the use of PreparedStatements it is not possible to SQL Inject. More on this is discussed above in the Data Dictionary.

SQL Injections were attempted in the **“SQL Injection”** under **“Beta tests”**and all fail which securely back up my point of the system being not vulnerable to SQL Injection.

The second big issue is that students could write a piece of Java code that could run and delete parts of my code, or take down the server. I initially thought this was a problem until I tried to cause damage to the server myself. What I realised is that almost all malicious code you can write in Java requires imports, which the students do not have access to, therefore for the other vulnerabilities that are left they are simply searched for before compilation. I am confident to say this criterion **has been met**.

Criterion #4

“A secure teacher portal that allows teachers to view certain student's progress and manage their account”

Unfortunately there is not yet a teacher portal available for teachers to be able to change passwords and set challenges. This is simply due to time constraints and the already sufficient amount of code (> 2.4k lines).

This criterion **has <u>not</u> been met.**

In the future, adding a teacher portal would not be difficult at all. Some slight adjustments to the database may be required, including to the credentials table and some other additional classes and webpages for the teacher to be able to use the UI effectively, but this is definitely achievable and could be developed in the future.

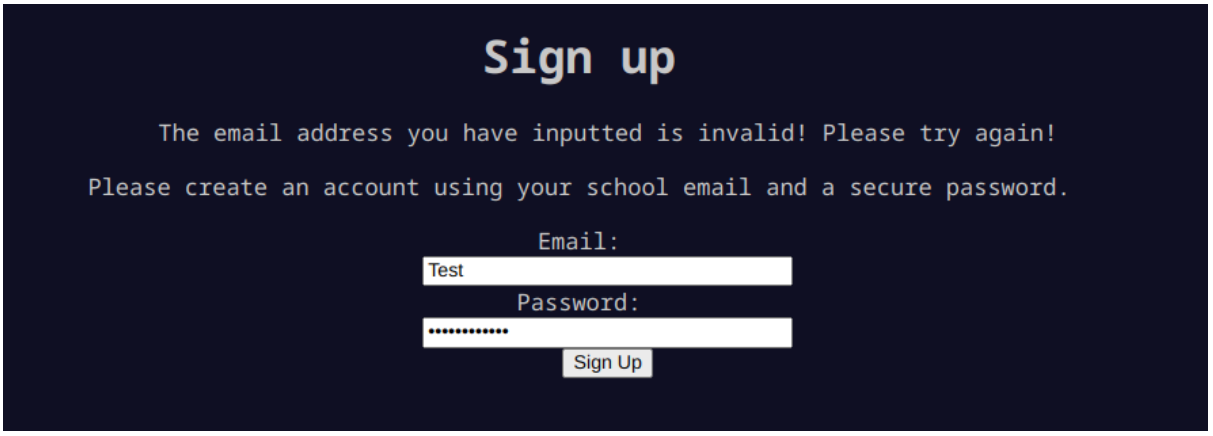
Beta test - Functionality

Invalid email inputs

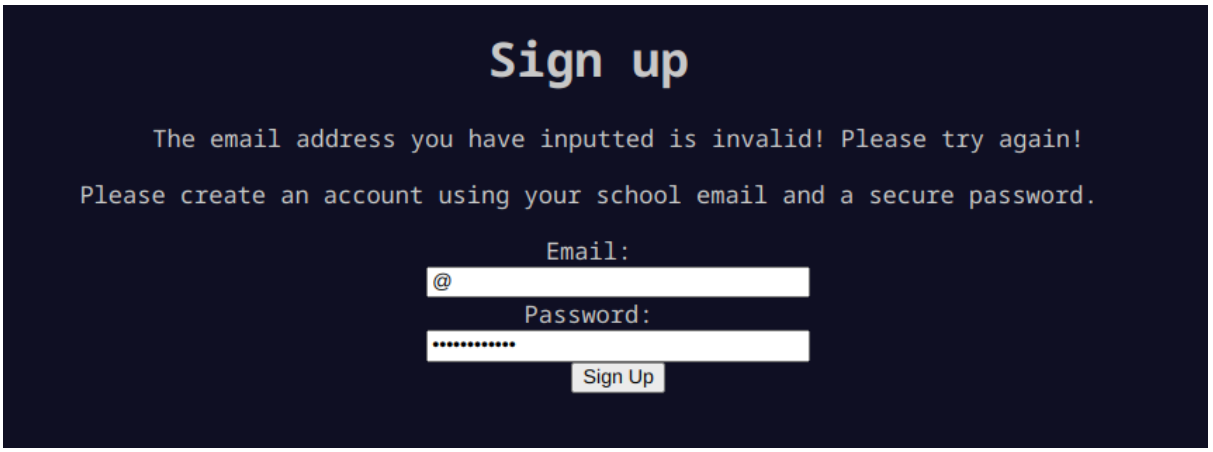
Based on the order of the beta test plan, the first item to test is the email input box, which is validated on the signup page. The data that is being used is close to valid email addresses, but are still not capable of being valid email addresses therefore they should all be rejected by the program.

Data	Output	Result
Test, @, test@, @gmail.com, test@com, “Test email@gmail.com”	“Email address invalid.” for all	Pass ✅

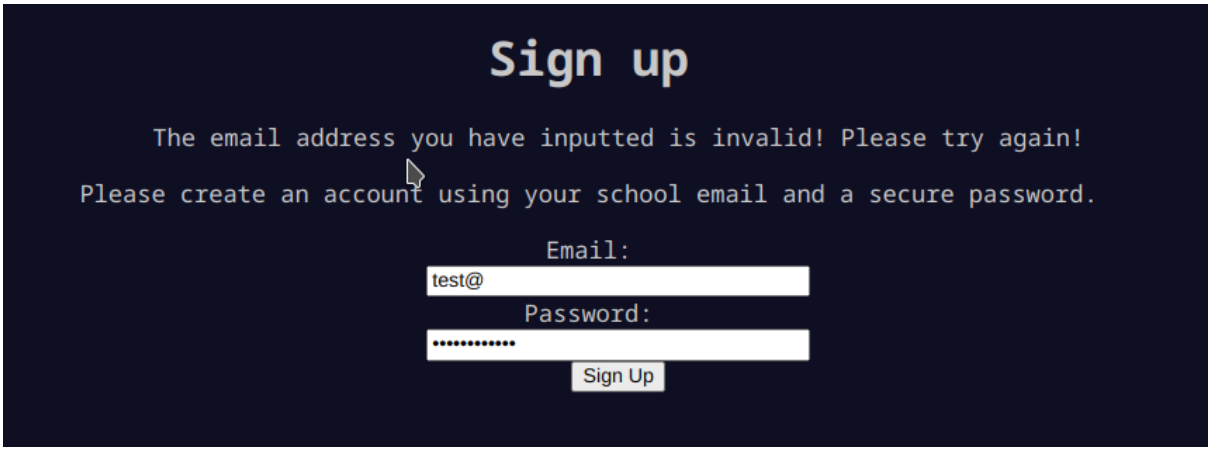
For this test I will be using “test” as the email input. This should not work as “test” would not be a valid email address.



For this test, I will only be using the “@” symbol. Again this email should not work, however, it tests validation to ensure it is not just looking for an “@” symbol.



For this test, I will be using “test@”. This steps up from the previous test to show that the validation isn’t just looking for a string prior to an “@” symbol.



This test uses “@gmail.com”. This is a valid domain name, however, it cannot be a valid email address as there is no string prior to the “@” symbol

Similar to above, this test checks that the validation is not looking for a string, then an @ symbol, then another string.

Finally, this test data would be valid if it weren't for the blank space symbol, so this ensures that an account with a space cannot be used.

Valid email and password

The next three are all to check that legitimate email addresses and passwords allow you to create an account. This is possibly the most important check of them all. It is all great having a system that prevents any non-existent email from being used, but it's not so great if it doesn't allow real emails either!

Data	Output	Result
tom.swainston@hppc.co.uk	Successful email	Pass ✅
tom.swainston@hppc.co.uk	Unsuccessful password	Fail ❌
Password123!	Successful password	Pass ✅

It is clear from the failure in the first password, but the pass with the second password that an error lies somewhere in the RegEx being used. This issue will be addressed in Error #4

Challenges

Test	Evidence	Comment
Remove package	========== ======== java.lang.NoClassDefFou	The code returns an error to the user. **Test passed**

	ndError: Challenge8 (wrong name: org/swainston/Challenge8) ========== ========	
Change the method name	[Line: 3, org.swainston.Challenge8 is not abstract and does not override abstract method upperCaseConverter(java .lang.String) in org.swainston.tests.Upper CaseChallenge.UpperCas eChallenge in /Challenge8.java]	The code returns an error to the user. **Test passed**
Enter random text	[Line: 1, reached end of file while parsing in /Challenge8.java]	The code returns an error to the user. **Test passed**
Change the method to return the wrong type	[Line: 8, incompatible types: int cannot be converted to java.lang.String in /Challenge8.java]	The code returns an error to the user. **Test passed**

Beta test - Robustness

Null email inputs

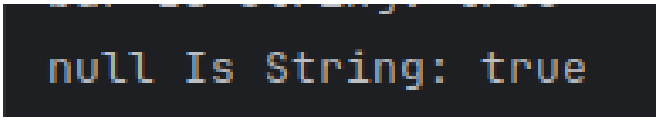
This one is slightly more difficult to test as it requires a debug in the console to output the results that are added to the credential store. I have done this by introducing this temporary line of code into the credentials class.

```
credentialMap.put(email, password);
System.out.println(credentialMap.get(email) + " Is String: " + (credentialMap.get(email) instanceof String));
return true;
```

The only place that this can be tested currently is in the password box, as the email box would not reach adding the code due to validation.

Let's try in the password box using a valid email.

This is what is outputted into the console



The null input has been stored as a string, and not as a null value.
Test passed!✅

SQL Injection

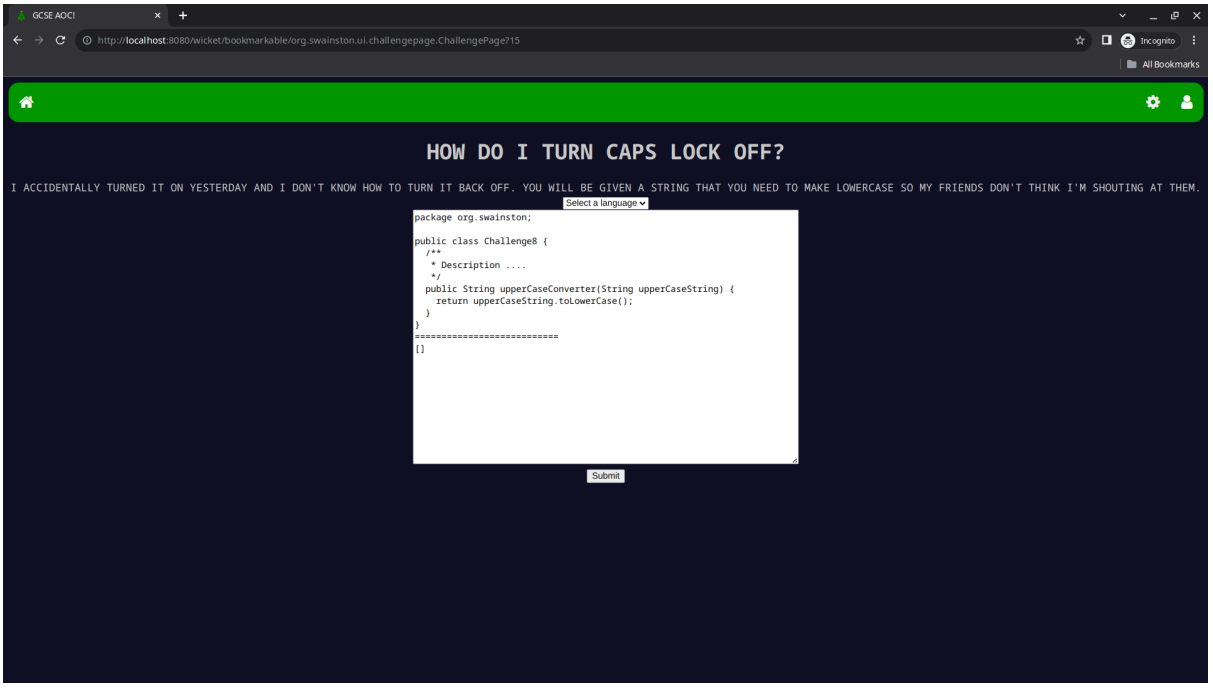
To test SQL Injection the following statement was inputted into the text box in the system which communicate directly to the database

"Qwerty OR 1=1"

After pressing sign-up, the system notifies the user that the email is not valid and does not gain them access to the system. Test **passed.**

Challenges

Firstly I entered the text from the first beta test and this was the result.



The code successfully compiled and the tests run were a success as no test failed messages occurred. Test **passed.**

The system is robust as shown from the following tests below. The system takes all inputs from the user and executes the code without failing. It can therefore be classes as robust.

The only failure of robustness could be said to be the trust within the user as there are certain holes that allow the user to close down the backend and execute unwanted code from.

Test	Evidence	Comment
Remove or change the class access modifier	========== java.lang.RuntimeException: java.lang.IllegalAccessException: class org.swainston.ChallengePageForm cannot access a member of class org.swainston.Challenge8 with modifiers "" ==========	The code returns an error to the user. **Test passed**
Remove or change the method access modifier	========== [Line: 7, upperCaseConverter(java.lang.String) in org.swainston.Challenge8 cannot implement upperCaseConverter(java.lang.String) in org.swainston.tests.UpperCaseChallenge.UpperCaseChallenge attempting to assign weaker access privileges; was public in /Challenge8.java] ==========	The code returns an error to the user. **Test passed**
Add invalid imports	[Line: 3, <identifier> expected in /Challenge8.java, Line: 3, <identifier> expected in /Challenge8.java]	The code returns an error to the user. **Test passed**
Change the parameter type	[Line: 3, org.swainston.Challenge8 is not abstract and does not override abstract method	The code returns an error to the user. **Test passed**

	upperCaseConverter(java.lang.String) in org.swainston.tests.UpperCaseChallenge.UpperCaseChallenge in /Challenge8.java, Line: 8, int cannot be dereferenced in /Challenge8.java]	
Completely empty the text input box	System returns the previously saved code that was in the box	The code returns an error to the user. **Test passed**
Attempt System.exit()	Process finished with exit code 0	Server shuts down. Test **failed.**

Maintainability of the solution

This solution has the capability to be hosted on a small, low-powered VPS or even a Raspberry Pi. The only maintenance that could be required would include new challenges being added to the database, which currently requires going into the project and changing the SetUpDB.sql file, however, this could easily be implemented into a webpage where new challenges are added through a user interface. Similarly, another change that may need to happen to the challenges is if the specification were to change. Currently, a lot of the challenges are specific to the current OCR GCSE specification, which some might say is slightly outdated. When this specification is updated and a new one is released, the points that students are required to know how to code may change. This would require new challenges to be added *as well as* the old challenges being removed from the database.

As far as the actual code, there is little to no maintenance required long term to keep the system running. The only code maintenance that may be required could be down to this:

Jetty 9 is not one of the most recent Jetty versions. In fact, IntelliJ recognises and notified me that Jetty 9 has a vulnerability. The most recent version is Jetty 12. Sadly it is not as simple as changing the numbers in the pom.xml file. New file paths have been added to Jetty 12 meaning that certain parts of the code may no longer be functional if the version were to jump quite largely from 9 to 12. However, the solution is still fully functional with Jetty 9 so there is no urgent need for an update.

Reporting bugs

In the case that bugs are found within the project by students and other users, it makes sense to add some sort of contact to the main page of the website. I decided to leave my email contact on the homepage which allows students to contact me if they find a bug. I have also noted that students are able to send me emails if they think there could be improvements or new features added to the website that would increase the overall satisfaction and teaching capabilities of the website.

GitHub

Throughout the project I was updating my project via GitHub, making frequent commits to the repository as well as utilising branches to run tests on. The GitHub repository can be found online, via an invite, so succeeding developers are able to view my previous changes, as well as fork, clone and push to my work.

Potential development of the solution

There are endless potential development opportunities for this solution. A few are listed below:

- Multi-Language support
    - One of the most prominent updates that this system could undergo is the use of multi-language support. Currently, the only supported language is Java as this was the most straightforward language to implement as my project is written in Java, although this was still by no means easy in the slightest! The most commonly used language at GCSE is Python as it is quite easy to catch onto and follows quite logically, with less confusing conventions and language keywords, unlike Java. If this solution were to allow Python it could become much more accessible to students. Other popular languages such as the C family, or Javascript could be implemented as well as a way to expand a student's knowledge of coding. More niche languages such as Rust could also be implemented to really push students and prepare them beyond what is asked for at GCSE
- Teacher portals
    - This was briefly touched upon above, but the development opportunity surrounding a teacher portal would be very useful to both students and teachers. For the teachers, it would mean that they can set students tasks to do for homework, track how long they have been on the project, and perhaps even how long they have spent on a certain challenge. As all submitted attempts are stored in the database the teacher could look through the students' attempts and find the most common error and address the class making teaching much more efficient and more personalised to the class. Another use of the teacher portal would be the ability to change the password if a student were to forget, which makes the time when a student forgets their password much easier as it would take a few clicks of a button, instead of any need for email communication and password resets etc.
- Leaderboards
    - It was briefly mentioned in the previous sections of this documentation, but the use of leaderboards could come in very useful for both student and teacher. Leaderboards could be based on many things, such as the least attempts for a challenge, the least time spent on a challenge or the total number of challenges the students have completed. Not only would the leaderboard allow teachers to check who is doing the best in the class and reward them, but it would also allow students to become competitive and boost class morale.

Limitations of the solution

Due to time constraints and other unexpected problems that I encountered, I have not completed as much of the solution as I had initially planned to complete. The current most prominent limitation of the solution is that the students are not able to step through each challenge as they wish to. However, implementing a solution to this limitation would not be difficult. It requires an extra block of code to auto-refresh the page when the student selects a challenge they wish to do. This would allow the student to step through the challenges as they wish and even try harder challenges if they already understand the basics of coding and have previous coding experience.

Other limitations of the solution include:

- Exam boards
    - Due to both my GCSE and A-Level Computer Science being with OCR, that is the exam board that I have based all of the challenges off, including what would be required by the student for each challenge. To make the solution less limited it would be required to examine other popular exam boards, such as AQA and Edexcel and see what aspects of coding they require. New challenges would then need to be written and added to the solution, as well as the option to choose which exam board the students are using.
- GCSE only
    - As I created this solution as a way to combat OCRs removal of the GCSE Computer Science coding aspect, this solution is only wholly effective for GCSE students.

How the other limitations can be solved:

- Exam boards
    - This limitation requires a little bit more than just adding content. Each exam board has different ways of assessing coding, and other exam boards may even have a project that is a required part of the coursework, making my system partially redundant. Some research would be required to see how each exam board assesses coding abilities and then the system could be catered for each exam board
- GCSE only
    - This system could be catered to A-Level students as well as GCSE students. The system would need to be modified to include more difficult challenges, but these could be added on top of the challenges for GCSE students, giving valuable revision.

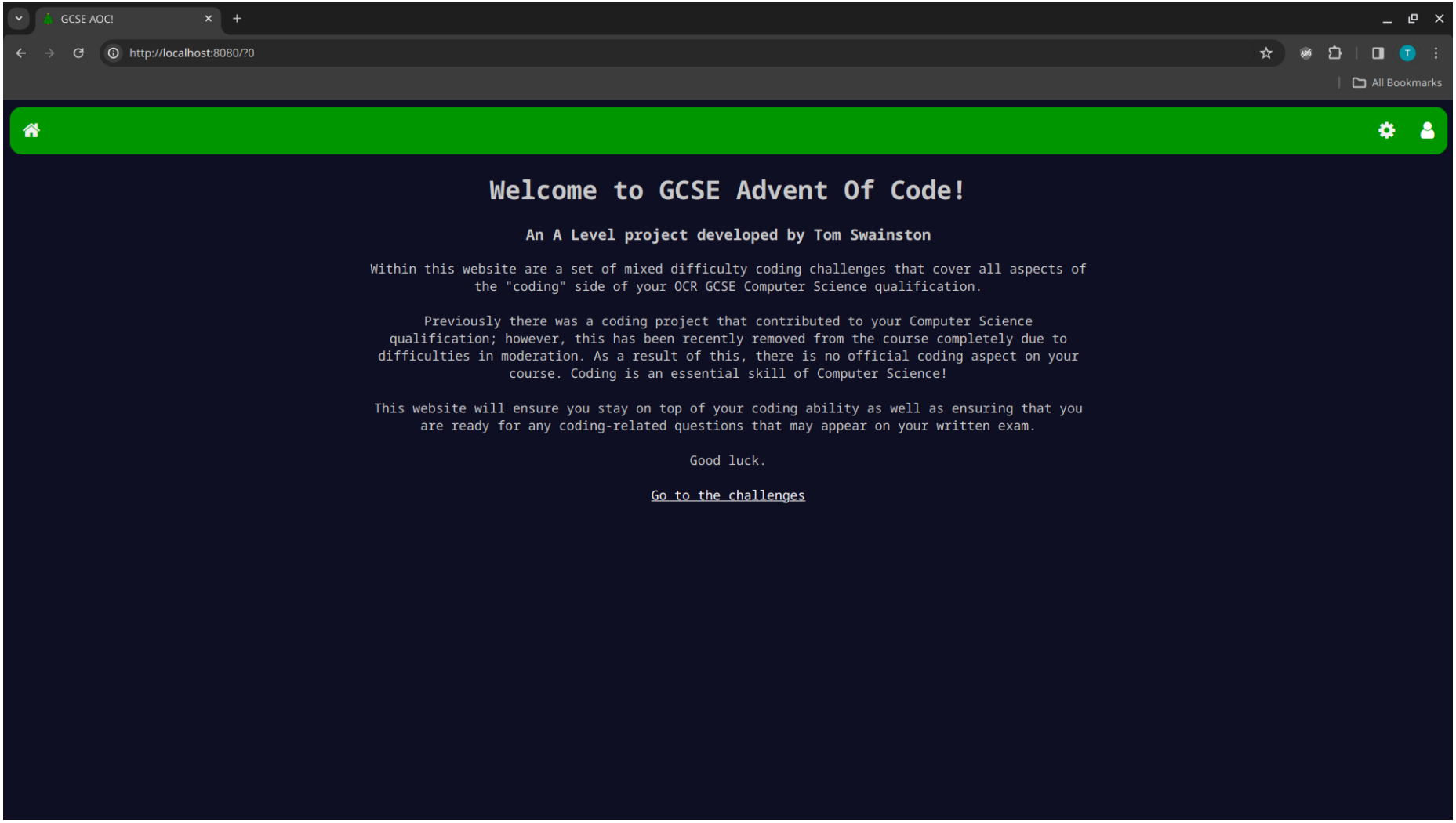
Usability of the User Interface

As the only part of my solution that the student sees will be the front end, it’s essential that it is clear and straightforward to use the website. The following section analyses each section of each webpage evaluating its useability from a student's point of view.

Home page useability

The homepage of the website introduces what the solution is about and the necessity for the solution.

The homepage is very user-friendly and easy to navigate and contains the navigation bar that all of the other web pages use. The two options for progression past the homepage are by either clicking the hyperlink below the description of the webpage which states “Go to the challenges”, or by using the navigation bar login icon for the student to either create an account or log in. The go-to challenges hyperlink will only take the students to the challenge if they already have an active signed-in session, otherwise they will be prompted to sign in/register. This home-page is a success. It is a functional page that serves its purpose of describing the general project and allows the student to navigate around. The only features that could perhaps be added are cosmetic features and not functional features, such as more colours or photos of content within the website.


GCSE AOC!
http://localhost:8080/?0
All Bookmarks
Welcome to GCSE Advent Of Code!
An A Level project developed by Tom Swainston
Within this website are a set of mixed difficulty coding challenges that cover all aspects of the "coding" side of your OCR GCSE Computer Science qualification.
Previously there was a coding project that contributed to your Computer Science qualification; however, this has been recently removed from the course completely due to difficulties in moderation. As a result of this, there is no official coding aspect on your course. Coding is an essential skill of Computer Science!
This website will ensure you stay on top of your coding ability as well as ensuring that you are ready for any coding-related questions that may appear on your written exam.
Good luck.
Go to the challenges

Register/sign-in page

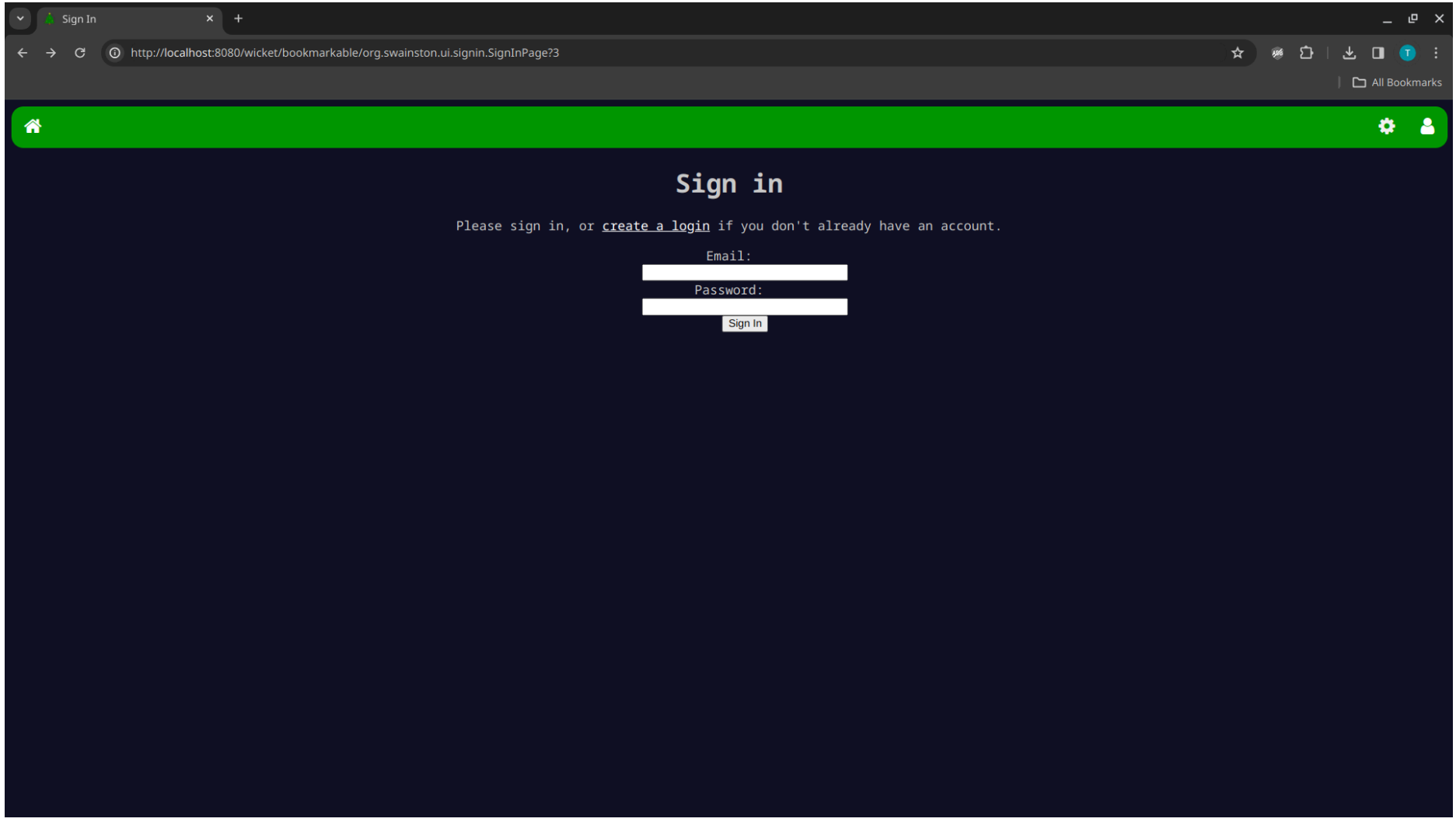
There is very little content in both the sign-in and sign up page, and this is the reason that I decided to go for a very minimalistic approach that allows the student to not get confused and sets a clear set of directions of what the student needs to do to either log-in or register.

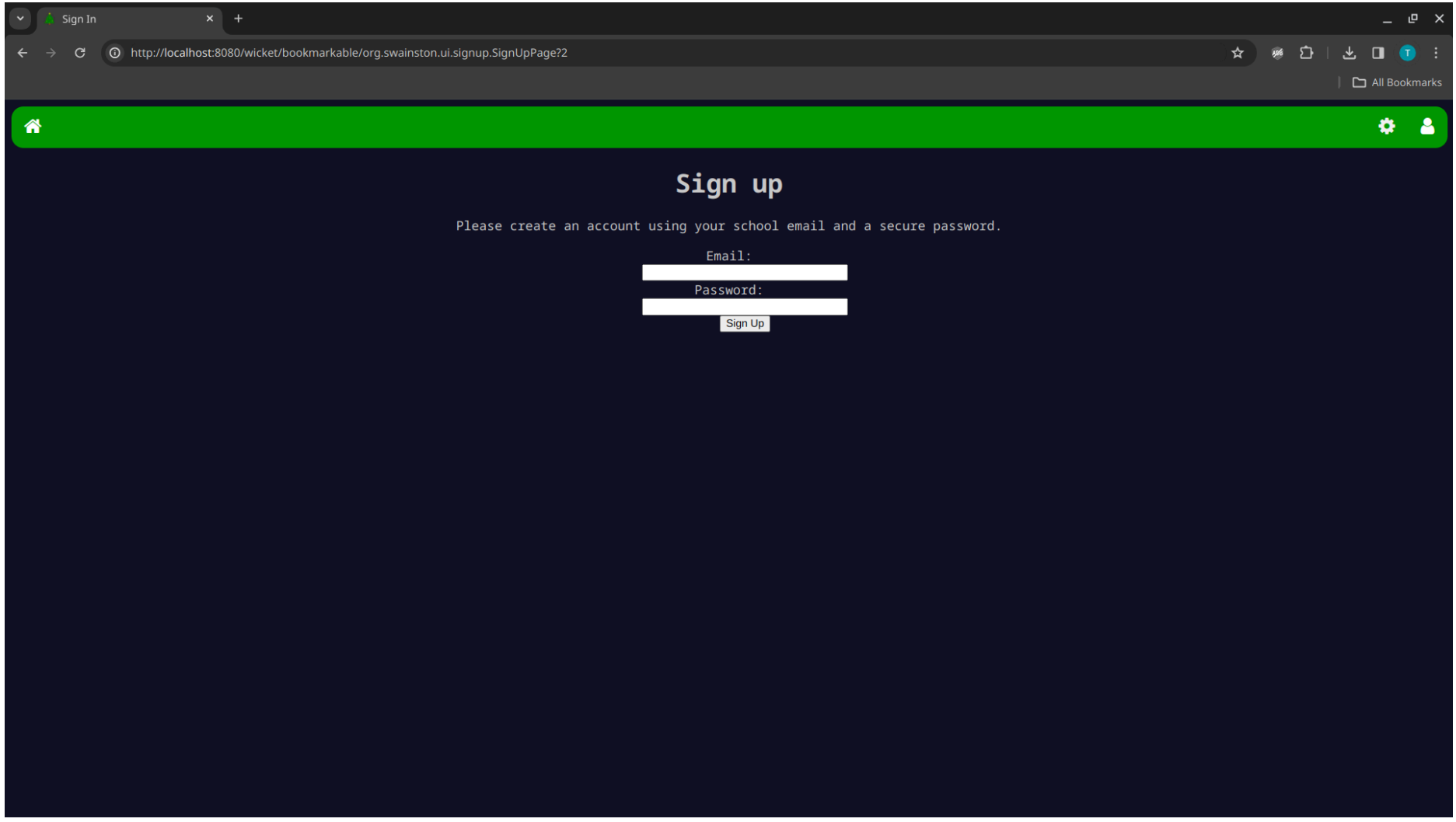
In all cases where a student is required to be signed in, they are sent to the sign-in page. Within the sign-in page, the first thing they will read is asking whether they have an account already and a redirect to the sign-up page.

The only two boxes in both of the pages are the email and password boxes, as well as a basic button which signs the user in or creates the account. This level of minimalism is essential for student useability.

Similarly, with the homepage, both the signup and sign-in pages have a navigation bar to keep consistency and allow the user to always be able to easily access the homepage

The sign-in and signup pages work functionally as they are meant to. They validate inputs and add data items to the database. The two pages are also interconnected meaning that each page can be accessed from the other. Based on this, **both pages are a success.**


Sign In
http://localhost:8080/wicket/bookmarkable/org.swainston.ui.signin.SignInPage?3
All Bookmarks
Sign in
Please sign in, or create a login if you don't already have an account.
Email:
Password:
Sign In


Sign In
http://localhost:8080/wicket/bookmarkable/org.swainston.ui.signup.SignUpPage?2
All Bookmarks
Sign up
Please create an account using your school email and a secure password.
Email:
Password:
Sign Up

Challenge page

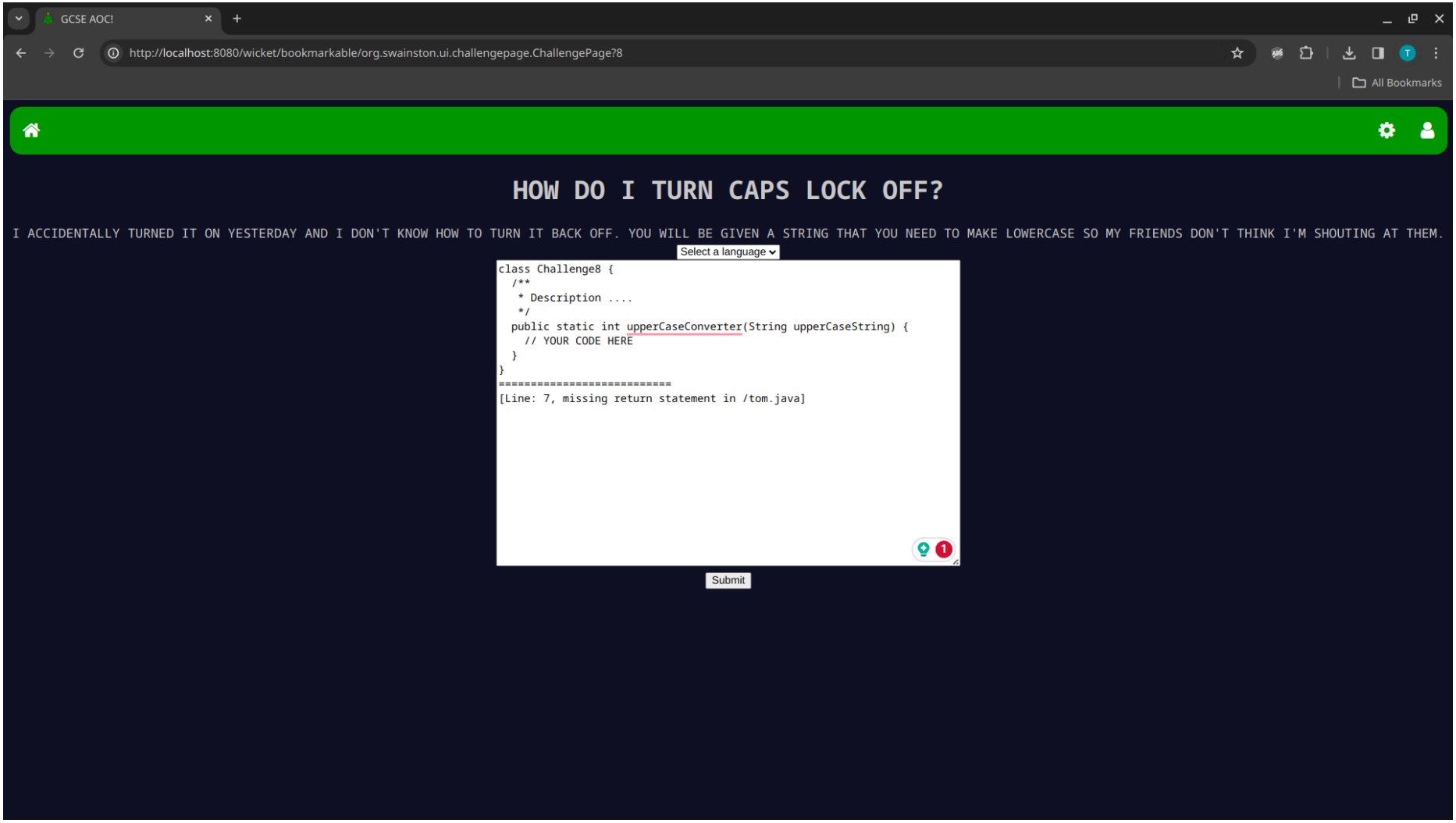
Sticking with the theme of being simple, the challenge page is consistent. The page is very easy to use and understand, even when seeing it for the first time. It's quite clear what is for what on the page. Of course, this page also contains the navigation bar so the student can reach the homepage, and sign out with one click.

The box that the student uses to input code is simple to use, although it is recommended that the student uses an IDE of their choice to code the solution to the challenge and then input it, however, it is possible to code straight into the text box and it is not difficult to do this.

The submit button is nothing complicated, and the feedback is fast and responsive to the student. Once the button has been pressed, the program attempts to compile the program, and if this fails it will give the error to the student in the same text box below a self-generated separator. If the error occurs within the program this error will also be shown below the separator. Finally, if the tests that are run on the student code do not pass, the code and the expected, and actual output are shown to the student, this means that the student is able to easily identify where an error or bug in their solution may be.

In the future it would be good for the challenge page to look how it was initially set out as a wireframe in the design.

This challenge page was a **partial success**. The page is functional for taking users inputted challenge text, compiling and testing their code, however, the student does not have any free choice between the challenge which should be a feature. This can be met by adding a button or drop-down menu that the student can use to select the challenge that they would like to see or go to.


GCSE AOC!
http://localhost:8080/wicket/bookmarkable/org.swainston.ui.challengepage.ChallengePage?8
All Bookmarks
HOW DO I TURN CAPS LOCK OFF?
I ACCIDENTALLY TURNED IT ON YESTERDAY AND I DON'T KNOW HOW TO TURN IT BACK OFF. YOU WILL BE GIVEN A STRING THAT YOU NEED TO MAKE LOWERCASE SO MY FRIENDS DON'T THINK I'M SHOUTING AT THEM.
Select a language
class Challenge8 {
/**
* Description ....
*/
public static int upperCaseConverter(String upperCaseString) {
// YOUR CODE HERE
}
}
==========
[Line: 7, missing return statement in /tom.java]
Submit

Code appendix

ChalengePage.html

```
<!DOCTYPE html>
<html xmlns:wicket="http://wicket.apache.org">
<head>
   <meta charset="utf-8">
   <title>GCSE AOC!</title>
   <meta name="robots" content="noindex,nofollow">
   <link rel="stylesheet"

href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css"
         type="text/css">
   <link rel="stylesheet" href="style.css" type="text/css" title="Stylesheet"/>

   <link rel="icon" href="xmas_tree.png">

   <script src="https://kit.fontawesome.com/79cfa1bf9a.js"
           crossorigin="anonymous"></script>
</head>

<body>

<div class="navbar">
   <a id="home" wicket:id="go-home"><i class="fa fa-home"></i></a>
   <a id="signout" wicket:id="sign-out"><i class="fa fa-user"></i></a>
   <a id="settings"><i class="fa fa-gear"></i></a>
</div>

<div class="main">
   <div id="left">
       <div id="title">
           <h1>
               <wicket:label wicket:id="title">Title</wicket:label>
           </h1>
       </div>

       <div id="instructions">
           <wicket:label wicket:id="instructions">Instructions</wicket:label>
       </div>

       <div id="language">
           <select name="language" id="language">
               <option value="java">Java</option>
               <option value="python" disabled>Python</option>
               <option value="c" disabled>C</option>
               <option value="none" selected disabled>Select a language
               </option>
           </select>
       </div>

       <form wicket:id="form-submit-code">
           <div id="input">
               <textarea id="w3review" wicket:id="textarea-code"
                         name="w3review" rows="21" cols="65"></textarea>
           </div>
           <input type="submit" value="Submit"/>
       </form>

   </div>
</div>
```

```
</body>
</html>
```

HomePage.html

```
<!DOCTYPE html>
<html xmlns:wicket="http://wicket.apache.org">

<wicket:head>
   <meta charset="utf-8">
   <title>GCSE AOC!</title>
   <link rel="stylesheet" href="style.css" type="text/css">
   <link rel="stylesheet"

href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css"
         type="text/css">
   <link rel="icon" href="xmas_tree.png">
</wicket:head>

<body>

<div class="navbar">
   <a id="home"><i class="fa fa-home"></i></a>
   <a wicket:id="signIn" id="signout"><i class="fa fa-user"></i></a>
   <a id="settings"><i class="fa fa-gear"></i></a>

</div>

<div class="main">
   <h1>Welcome to GCSE Advent Of Code!</h1>
   <h3>An A Level project developed by Tom Swainston</h3>
   <p>Within this website are a set of mixed difficulty coding challenges that
       cover all aspects of the "coding" side
       of your OCR GCSE Computer Science qualification.
       <br> <br> Previously there
       was a coding project that contributed to your Computer Science
       qualification; however, this has been recently removed from the
       course completely due to difficulties in moderation.
       As a result of
       this, there is no official coding aspect on your course.
       Coding is an
       essential skill of Computer Science! <br> <br> This website will ensure you
       stay on top of your coding ability as well as ensuring that you are
       ready for any coding-related questions that may appear on your
       written exam. <br><br> Good luck.</p>
   <br/>

   <p><a wicket:id="challengePageLink">Go to the challenges</a></p>
</div>

</body>

</html>
```

SignInPage.html

```
<html xmlns:wicket="http://wicket.apache.org">
<head>
   <meta charset="utf-8">
   <title>Sign In</title>
   <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css"
type="text/css">
   <link rel="stylesheet" href="style.css" type="text/css"
         title="Stylesheet"/>
   <link rel="icon" href="xmas_tree.png">
   <script src="https://kit.fontawesome.com/79cfa1bf9a.js"
crossorigin="anonymous"></script>
</head>
<body>

<div class="navbar">
   <a id="home" wicket:id="go-home"><i class="fa fa-home"></i></a>
   <a id="signout" wicket:id="sign-out"><i class="fa fa-user"></i></a>
   <a id="settings"><i class="fa fa-gear"></i></a>
</div>


<div class="main" >

   <h1>Sign in</h1>

   <p>
       Please sign in, or <a wicket:id="signUpLink">create a login</a> if you
       don't already have an account.
   </p>

   <form wicket:id="signInForm">
       <dl>
           <dt>Email:</dt>
           <dd><input type="text" wicket:id="username" size="30"/></dd>
           <dt>Password:</dt>
           <dd><input type="password" wicket:id="password" size="30"/></dd>
           <dd><input type="submit" wicket:id="signInFormSubmit" name="submit" value="Sign
In"/></dd>
       </dl>
   </form>
</div>
</body>
</html>
```

SignUpPage.html

```
<html xmlns:wicket="http://wicket.apache.org">
<head>
   <meta charset="utf-8">
   <title>Sign In</title>
   <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css"
type="text/css">
   <link rel="stylesheet" href="style.css" type="text/css"
         title="Stylesheet"/>
   <link rel="icon" href="xmas_tree.png">
   <script src="https://kit.fontawesome.com/79cfa1bf9a.js"
crossorigin="anonymous"></script>
</head>
<body>

<div class="navbar">
   <a id="home" wicket:id="go-home"><i class="fa fa-home"></i></a>
   <a id="signout" wicket:id="sign-out"><i class="fa fa-user"></i></a>
   <a id="settings"><i class="fa fa-gear"></i></a>
</div>


<div class="main">

   <h1>Sign up</h1>

   <span wicket:id="feedback"/>
   <p>Please create an account using your school email and a secure password.</p>

   <form wicket:id="signUpForm">
       <dl>
           <dt>Email:</dt>
           <dd><input wicket:id="username" type="text" size="30"/></dd>
           <dt>Password:</dt>
           <dd><input wicket:id="password" type="password" size="30"/></dd>
           <dd>
               <input type="submit" wicket:id="signUpFormSubmit" name="submit" value="Sign
Up"/>
           </dd>
       </dl>
   </form>
</div>
</body>
</html>
```

Database.java

```
package org.swainston.database;

import com.mysql.cj.jdbc.MysqlConnectionPoolDataSource;
import com.mysql.cj.jdbc.MysqlDataSource;

import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.SQLException;

/**
* Holds the method that makes a connection with the database.
*/
public class Database {

 /**
  * Initializes the database source with the server name, port number,
database name, user and
  * password.
  *
  * @return DataSource the source from the
  *
  * @throws SQLException if unable to connect to database
  */
 public static DataSource initDataSource() throws SQLException {

   MysqlDataSource dataSource = new MysqlConnectionPoolDataSource();

   dataSource.setServerName("localhost");
   dataSource.setPortNumber(3306);
   dataSource.setDatabaseName("GCSEAdventOfCode");
   // Database user username
   dataSource.setUser("root");
   // Database user password
   dataSource.setPassword("Password123!");

   try (Connection conn = dataSource.getConnection()) {
     if (!conn.isValid(1)) {
       throw new SQLException("Failed to establish database connection.");
     }
   }

   return dataSource;
 }
}
```

MySQLAttemptsStore.java

```
package org.swainston.database;

import org.swainston.Attempt;
import org.swainston.AttemptsStore;
import org.swainston.Challenge;
import org.swainston.WicketApplication;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Optional;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
* Holds the method that inserts a users default information into the attempts table.
*/
public class MySQLAttemptsStore implements AttemptsStore {

 private static final String SET_ATTEMPT =
     "UPDATE attempts SET attempt = ?" + " WHERE email = ? AND id = ?";
 private static final String LOAD_DEFAULT = "INSERT INTO attempts VALUES(?,?,?)";
 private static final String GET_ATTEMPT = "SELECT * FROM attempts WHERE email = ? AND id = ?";

 /**
  * Loads a default template into the attempt table.
  *
  * @param email     users email address
  * @param challenge the challenge
  */
 public static void loadDefault(String email, Challenge challenge) {

   try (Connection conn = WicketApplication.getConnection()) {

     PreparedStatement stmt = conn.prepareStatement(LOAD_DEFAULT);
     stmt.setString(1, email);
     stmt.setInt(2, challenge.getId());
     stmt.setString(3, challenge.getSolutionTemplate());
     stmt.executeUpdate();

   } catch (SQLException e) {
     Logger.getLogger(WicketApplication.class.getName()).log(Level.SEVERE, e.getMessage());
   }
 }

 /**
  * Gets an attempt.
  *
  * @param email the student's email
  * @param id    the id of the attempt
  *
  * @return Optional Attempt
  */
 @Override
 public Optional<Attempt> getAttempt(String email, int id) {


   try (Connection conn = WicketApplication.getConnection();
        PreparedStatement stmt = conn.prepareStatement(GET_ATTEMPT)) {
     stmt.setString(1, email);
     stmt.setInt(2, id);
     ResultSet rs = stmt.executeQuery();
     if (rs.next()) {
       Attempt attempt = new Attempt();
       attempt.setEmail(rs.getString(1));
       attempt.setId(rs.getInt(2));
       attempt.setAttempt(rs.getString(3));

       return Optional.of(attempt);

     }
     return Optional.empty();
```

```
    } catch (SQLException e) {
      Logger.getLogger(WicketApplication.class.getName()).log(Level.SEVERE, e.getMessage());
      throw new RuntimeException(e);
    }
  }

  /**
   * Adds a students attempt to the database
   *
   * @param email   the student's email
   * @param id      the id of the attempt
   * @param attempt the students attempt at the challenge
   */
  @Override
  public void setAttempt(String email, int id, String attempt) {

    try (Connection conn = WicketApplication.getConnection()) {

      PreparedStatement stmt = conn.prepareStatement(SET_ATTEMPT);
      stmt.setString(2, email);
      stmt.setInt(3, id);
      stmt.setString(1, attempt);
      stmt.executeUpdate();

    } catch (SQLException e) {
      Logger.getLogger(WicketApplication.class.getName()).log(Level.SEVERE, e.getMessage());
    }
  }
}
```

MySQLChallengeStore.java

```
package org.swainston.database;

import org.swainston.Challenge;
import org.swainston.ChallengeStore;
import org.swainston.WicketApplication;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Optional;
import java.util.logging.Level;
import java.util.logging.Logger;

public final class MySQLChallengeStore implements ChallengeStore {

 private static final String GET_NEXT_CHALLENGE =
     "SELECT * FROM challenges WHERE id = (SELECT progress FROM users where email =
?)";


 /**
  * Fetches the student's next challenge and returns it as an optional
  * challenge
  * @param userEmail the user's email
  * @return optional of challenge
  */
 public Optional<Challenge> getNextChallenge(String userEmail) {

   try (Connection conn = WicketApplication.getConnection();
        PreparedStatement stmt = conn.prepareStatement(GET_NEXT_CHALLENGE)) {
     stmt.setString(1, userEmail);
     ResultSet resultSet = stmt.executeQuery();
     if (resultSet.next()) {
       Challenge challenge = new Challenge();
       challenge.setId(resultSet.getInt(1));
       challenge.setTitle(resultSet.getString(2));
       challenge.setDescription(resultSet.getString(3));
       challenge.setSolutionTemplate(resultSet.getString(4));

       return Optional.of(challenge);

     }
     return Optional.empty();
   } catch (SQLException e) {
     Logger.getLogger(WicketApplication.class.getName()).log(Level.SEVERE,
e.getMessage());
     throw new RuntimeException(e);

   }
 }
}
```

MySQLCredentialStore.java

```
package org.swainston.database;

import org.swainston.CredentialsStore;
import org.swainston.WicketApplication;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLDataException;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class MySQLCredentialsStore implements CredentialsStore {

 private static final String SQL_ADD = "INSERT INTO users VALUES(?, ?, ?)";
 private static final String SQL_EXISTS = "SELECT COUNT(*) FROM users WHERE email = ?;";
 private static final String SQL_VALIDATE =
     "SELECT COUNT(*) FROM users WHERE email = ? AND password = ?;";

 /**
  * Adds a user into the database
  * @param email the user's email
  * @param password the user's password
  * @return boolean of whether successful
  */
 @Override
 public boolean add(String email, String password) {
   // Check the user does not already exist in the database
   if (exists(email)) {
     return false;
   }
   try (Connection conn = WicketApplication.getConnection()) {
     PreparedStatement stmt = conn.prepareStatement(SQL_ADD);
     stmt.setString(1, email);
     stmt.setString(2, password);
     stmt.setInt(3, 1);
     stmt.executeUpdate();
     return true;
   } catch (SQLException e) {
     Logger.getLogger(WicketApplication.class.getName()).log(Level.SEVERE, e.getMessage());
   }
   return false;
 }


 /**
  * Check whether a user exists in the database
  * @param email the email of the user you are querying
  * @return boolean of whether the email can be matched to an account
  */
 @Override
 public boolean exists(String email) {

   try (var conn = WicketApplication.getConnection();
        var stmt = conn.prepareStatement(SQL_EXISTS)) {
     stmt.setString(1, email);
     var resultSet = stmt.executeQuery();

     while (resultSet.next()) {
       if (resultSet.getInt(1) == 1) {
         return true;
       } else if (resultSet.getInt(1) > 1) {
         throw new SQLDataException("Duplicate user found");
```

```
        }
      }
      return false;
    } catch (SQLException e) {
      Logger.getLogger(WicketApplication.class.getName()).log(Level.SEVERE, e.getMessage());
      return false;
    }
  }

  /**
   * Validate whether a user's credentials are correct
   * @param email the email of the user
   * @param password the user's password
   * @return if the email and password exist in the database
   */
  @Override
  public boolean validate(String email, String password) {
    try (var conn = WicketApplication.getConnection();
         var stmt = conn.prepareStatement(SQL_VALIDATE)) {
      stmt.setString(1, email);
      stmt.setString(2, password);
      var resultSet = stmt.executeQuery();

      while (resultSet.next()) {
        if (resultSet.getInt(1) >= 1) {
          return true;
        }
      }
      return false;
    } catch (SQLException e) {
      Logger.getLogger(WicketApplication.class.getName()).log(Level.SEVERE, e.getMessage());
      return false;
    }
  }
}
```

UpperCaseChallenge

```
package org.swainston.tests.UpperCaseChallenge;

public interface UpperCaseChallenge {
 /**
  * Method that converts a string from uppercase to all lowercase
  * @param text string that requires lowercasing
  * @return lowercase string
  */
 String upperCaseConverter(String text);

}
```

UpperCaseChallengeImpl

```
package org.swainston.tests.UpperCaseChallenge;

public class UpperCaseChallengeImpl implements UpperCaseChallenge {
 @Override
 public String upperCaseConverter(String text) {
   return text.toLowerCase();
 }
}
```

UpperCaseChallengeImplTest.java

```
package org.swainston.tests.UpperCaseChallenge;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.function.Executable;

public class UpperCaseChallengeImplTest {

 private UpperCaseChallenge upperCaseChallenge = new UpperCaseChallengeImpl();

 /**
  * Allows other classes to parse the upperCaseChallenge to this class
  *
  * @param upperCaseChallenge the student challenge object
  */
 public void setUpperCaseChallenge(UpperCaseChallenge upperCaseChallenge) {
   this.upperCaseChallenge = upperCaseChallenge;
 }

 /**
  * Tests to see whether a string with random upper case chars is converted to lower case
chars
  */
 @Test
 public void upperCaseConverter_all_chars() {
   String test = "AbcDeFghIjKlmNopQrsTuVwXYZ";
   String expected = "abcdefghijklmnopqrstuvwxyz";
   String actual = upperCaseChallenge.upperCaseConverter(test);
   Assertions.assertEquals(expected, actual);
 }

 /**
  * Tests whether given numbers, it does not alter the string at all
  */
 @Test
 public void upperCaseConverter_all_num() {
   String test = "23748923749";
   String expected = "23748923749";
   String actual = upperCaseChallenge.upperCaseConverter(test);
   Assertions.assertEquals(expected, actual);
 }

 /**
  * Tests to see whether given a null value, the code throws a NullPointerException
  */
 @Test
 public void upperCaseConverter_null() {
   Executable executable = () -> upperCaseChallenge.upperCaseConverter(null);
   Assertions.assertThrows(NullPointerException.class, executable);
 }

 /**
  * Tests whether to see a string of spaces is kept the same and not changed
  */
 @Test
 public void upperCaseConverter_all_spaces() {
   String test = "    ";
   String expected = "    ";
   String actual = upperCaseChallenge.upperCaseConverter(test);
   Assertions.assertEquals(expected, actual);
 }

 /**
```

```
  * Tests whether given an empty string, it returns an empty string again
  */
 @Test
 public void upperCaseConverter_empty_string() {
   String test = "";
   String expected = "";
   String actual = upperCaseChallenge.upperCaseConverter(test);
   Assertions.assertEquals(expected, actual);
 }
}
```

ChallengePage.java

```
package org.swainston.ui.challengepage;

import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.link.Link;
import org.apache.wicket.model.PropertyModel;
import org.opentest4j.AssertionFailedError;
import org.swainston.*;
import org.swainston.database.MySQLAttemptsStore;
import org.swainston.database.MySQLChallengeStore;
import org.swainston.ui.homepage.HomePage;

import java.util.Optional;

/**
* This class handles the challenge page and the submitted in the challenge page.
*/
public class ChallengePage extends WebPage implements AuthenticatedWebPage {

 private static final long serialVersionUID = 1L;

 private final ChallengeStore challengeStore;
 private final AttemptsStore attemptsStore;

 /**
  * Constructs the challenge page and layout.
  */
 public ChallengePage() {
   challengeStore = new MySQLChallengeStore();
   attemptsStore = new MySQLAttemptsStore();
   layout();
 }

 /**
  * Holds all the code required to run the page,
  * process the students' code and return feedback to the student
  */
 private void layout() {
   String user = getMySession().getUser();

   // Optional variables are used to ensure that no null variables are able to leak into code
   Optional<Challenge> nextChallenge = challengeStore.getNextChallenge(user);

   // Checks that the optional is not empty
   if (nextChallenge.isEmpty()) {
     // Theoretically unreachable
     throw new AssertionFailedError("Unable to find next challenge for user.");
   }

   // Get the challenge on the basis that it is not empty
   Challenge challenge = nextChallenge.get();
   Optional<Attempt> attempt = attemptsStore.getAttempt(user, challenge.getId());
   if (attempt.isEmpty()) {
     MySQLAttemptsStore.loadDefault(user, challenge);
   }

   add(new Label("title", new PropertyModel<>(challenge, "title")));
   add(new Label("instructions", new PropertyModel<>(challenge, "description")));
   add(new Link<Void>("go-home") {
     @Override
     public void onClick() {
       setResponsePage(HomePage.class);
     }
```

```
    });
    add(new Link<Void>("sign-out") {
      @Override
      public void onClick() {
        getMySession().signOut();
        setResponsePage(HomePage.class);
      }
    });

    add(new ChallengePageForm(user, challenge, attemptsStore));
  }


  private SignInSession getMySession() {
    return (SignInSession) getSession();
  }

}
```

HomePage.java

```
package org.swainston.ui.homepage;

import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.link.Link;
import org.apache.wicket.request.mapper.parameter.PageParameters;
import org.swainston.AuthenticatedWebPage;
import org.swainston.ui.challengepage.ChallengePage;

/**
* As this page does not implement {@link AuthenticatedWebPage} it does not
require auth to access
*/
public class HomePage extends WebPage {
 private static final long serialVersionUID = 1L;

 /**
  * Method containing the scripts required to run the home page
  *
  * @param parameters page parameters required for execution
  */
 public HomePage(final PageParameters parameters) {
   super(parameters);

   // Response page for the hyperlink "Go to challenges"
   add(new Link<Void>("challengePageLink") {
     public void onClick() {
       setResponsePage(new ChallengePage());
     }
   });

   // Response page for the navbar sign in button
   add(new Link<Void>("signIn") {
     public void onClick() {
       setResponsePage(new ChallengePage());
     }
   });
 }
}
```

SignInPage.java

```
package org.swainston.ui.signin;

import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.form.Button;
import org.apache.wicket.markup.html.form.Form;
import org.apache.wicket.markup.html.form.PasswordTextField;
import org.apache.wicket.markup.html.form.TextField;
import org.apache.wicket.markup.html.link.Link;
import org.apache.wicket.model.PropertyModel;
import org.apache.wicket.request.mapper.parameter.PageParameters;
import org.swainston.SignInSession;
import org.swainston.ui.homepage.HomePage;
import org.swainston.ui.signup.SignUpPage;

import java.util.HashMap;
import java.util.Map;

/**
* Class containing scripting for the sign in page
*/
public class SignInPage extends WebPage {
   private static final long serialVersionUID = 1L;

    /**
     * Constructing method that is run to load the active components of the page
     * @param parameters page parameters, required for super class
     */
   public SignInPage(final PageParameters parameters) {
        super(parameters);

        // Form used for taking users credentials
        add(new SignInForm("signInForm"));

        // Component that links users to create a login
        add(new Link<Void>("signUpLink") {
            public void onClick() {
                setResponsePage(SignUpPage.class);
            }
        });

        // Navigation bar home button
        add(new Link<Void>("go-home") {
            public void onClick() {
                setResponsePage(HomePage.class);
            }
        });

        // REDUNDANT, REDIRECTS TO HOMEPAGE, SAME AS ABOVE+
        add(new Link<Void>("sign-out") {
            @Override
            public void onClick() {
                setResponsePage(HomePage.class);
            }
        });
   }

    /**
     * Class required to create the form that takes the users credentials
     */
   private static final class SignInForm extends Form<Void> {
        private static final String USERNAME = "username";
        private static final String PASSWORD = "password";
```

```
        private final Map<String, String> properties = new HashMap<>();

        /**
         * Method that creates and handles the data given within the form
         * @param id the wicket:id of the HTML component
         */
        public SignInForm(final String id) {
            super(id);

            add(new TextField<>(USERNAME, new PropertyModel<String>(properties, USERNAME)));
            add(new PasswordTextField(PASSWORD, new PropertyModel<>(properties, PASSWORD)));
            add(new Button("signInFormSubmit"));
        }

        @Override
        public void onSubmit() {

            // Get session info for the user
            var signInSession = getMySession();

            // Sign the user in and proceed to the original page that was the destination
before
            // flow was interrupted by this login action.
            if (signInSession.signIn(getUsername(), getPassword())) {
                continueToOriginalDestination();
            } else {
                // Get the error message from the properties file associated with the
Component
                String loginError = getString("loginError", null, "Sign in details not
recognised.");

                // Register the error message with the feedback panel
                error(loginError);
            }
        }

        private String getPassword() {
            return properties.get(PASSWORD);
        }

        private String getUsername() {
            return properties.get(USERNAME);
        }

        private SignInSession getMySession() {
            return (SignInSession) getSession();
        }
    }
}
```

SignUpPage.java

```
package org.swainston.ui.signup;

import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.form.Button;
import org.apache.wicket.markup.html.form.Form;
import org.apache.wicket.markup.html.form.PasswordTextField;
import org.apache.wicket.markup.html.form.TextField;
import org.apache.wicket.markup.html.link.Link;
import org.apache.wicket.markup.html.panel.FeedbackPanel;
import org.apache.wicket.model.IModel;
import org.apache.wicket.model.PropertyModel;
import org.apache.wicket.request.mapper.parameter.PageParameters;
import org.apache.wicket.util.value.ValueMap;
import org.swainston.SignInSession;
import org.swainston.ui.homepage.HomePage;

import java.util.regex.Pattern;

/**
* <p>Contains the script for the sign-up HTML page.
* <br><br>
* Does not implement {@link org.swainston.AuthenticatedWebPage} as this page does not
* require authentication to be viewed.</p>
*
* @see org.apache.wicket.markup.html.WebPage
*/
public class SignUpPage extends WebPage {

 private static final long serialVersionUID = 1L;

 /**
  * Constructing method that initliaises the HTML components
  *
  * @param parameters page parameters for super class
  */
 public SignUpPage(final PageParameters parameters) {
   super(parameters);
   add(new SignUpForm("signUpForm"));
   add(new FeedbackPanel("feedback"));

   add(new Link<String>("go-home", (IModel) () -> "foo") {
     @Override
     public void onClick() {
       setResponsePage(HomePage.class);
     }
   });
   add(new Link<String>("sign-out", (IModel) () -> "foo") {
     @Override
     public void onClick() {
//               getMySession().signOut();
       setResponsePage(HomePage.class);
     }
   });

 }

 /**
  * The subclass that houses the code to handle the form used for the sign-up
  * @see org.apache.wicket.markup.html.form.Form
  * @see org.apache.wicket.util.value.ValueMap
  */
 private static final class SignUpForm extends Form<Void> {
```

```
    // Constants required to populate value map
    private static final String USERNAME = "username";
    private static final String PASSWORD = "password";

    private final ValueMap properties = new ValueMap();

    /**
     * Constructing method that initialises the components of the form
     * @param id form id
     */
    public SignUpForm(final String id) {
      super(id);

      //Username text field
      add(new TextField<>(USERNAME, new PropertyModel<String>(properties, USERNAME)));

      // Password text field
      add(new PasswordTextField(PASSWORD, new PropertyModel<>(properties, PASSWORD)));

      // Sign up button
      add(new Button("signUpFormSubmit"));
    }

    /**
     * <p>Submit method that is executed when the sign-up button is pressed <br>
     *  Handles all validation required for the user accounts </p>
     */
    @Override
    public void onSubmit() {

      // Validation for the email
      Pattern emailPattern = Pattern.compile(

"^(?=.{1,64}@)[A-Za-z0-9_-]+(\\.[A-Za-z0-9_-]+)*@[^-][A-Za-z0-9-]+(\\.[A-Za-z0-9-]+)*(\\.[A
-Za-z]{2,})$");

      // Validation for the password using regex
      Pattern passwordPattern = Pattern.compile("^[a-zA-Z0-9_]+$");

      String username = getUsername();

      if (!emailPattern.matcher(username).matches()) {
        // Get the error message from the properties file associated with the Component
        String loginErrorInvalidUsername = getString("loginErrorInvalidUsername", null,
            "The email address you have inputted is invalid! Please try again!");
        error(loginErrorInvalidUsername);
        return;
      }

      String password = getPassword();

      if (!passwordPattern.matcher(password).matches()) {
        // Get the error message from the properties file associated with the Component
        String invalidPassword = getString("loginErrorInvalidPassword", null,
            "Only letters, number, underscore in password");
        error(invalidPassword);
        return; // break to ensure that no unnecessary session is created
      }

      // Sign up, and return to the original destination page.
      SignInSession signInSession = getMySession();

      // Check if the sign-up is successful
      if (signInSession.signUp(username, password)) {
```

```
        // Sign the user in
        if (signInSession.signIn(username, password)) {
          continueToOriginalDestination();
        } else {
          // Get the error message from the properties file associated with the Component
          String loginError = getString("loginError", null, "Sign in details not
recognised.");

          // Register the error message with the feedback panel
          error(loginError);
        }
      } else {
        // Get the error message from the properties file associated with the Component
        String errorMessage;
        if (signInSession.userExists(getUsername())) {
          errorMessage = getString("loginError", null,
              "Unable to sign you up - that user name already exists");
        } else {
          errorMessage = getString("loginError", null, "Unable to sign you up");
        }
        // Register the error message with the feedback panel
        error(errorMessage);
      }
    }

    /**
     *
     * @return password from properties ValueMap
     */
    private String getPassword() {
      return properties.getString(PASSWORD);
    }

    /**
     *
     * @return username from properties ValueMap
     */
    private String getUsername() {
      return properties.getString(USERNAME);
    }

    /**
     *
     * @return Apache Wicket session cast to type {@link SignInSession}
     */
    private SignInSession getMySession() {
      return (SignInSession) getSession();
    }
  }
}
```

Attempt.java

```
package org.swainston;

import java.io.Serializable;

/**
* Class which holds information about a students given attempt
*/
public class Attempt implements Serializable {

 private int id;
 private String email;
 private String attempt;

 public int getId() {
   return id;
 }

 public void setId(int id) {
   this.id = id;
 }

 public String getEmail() {
   return email;
 }

 public void setEmail(String email) {
   this.email = email;
 }

 public String getAttempt() {
   return attempt;
 }

 public void setAttempt(String attempt) {
   this.attempt = attempt;
 }
}
```

AttemptClassLoader.java

```
package org.swainston;

/**
* Class which assists the loading of students code without the need of
* creating a new file in the project filestream
*/
public class AttemptClassLoader extends ClassLoader {

 private final byte[] bytes;

 /**
  * Takes the bytes from the students attempt and sets them to the final
  * variable above
  * @param bytes the students attempt
  */
 public AttemptClassLoader(byte[] bytes) {
   this.bytes = bytes;
 }

 /**
  * This finds a class from the binary name of the class
  * @param name
  *           The <a href="#binary-name">binary name</a> of the class
  *
  * @return an unspecified type of class
  */
 @Override
 protected Class<?> findClass(String name) {
   return defineClass(name, bytes, 0, bytes.length);
 }
}
```

AttemptsStore.java

```
package org.swainston;

import java.util.Optional;

public interface AttemptsStore {

 Optional<Attempt> getAttempt(String user, int id);

 void setAttempt(String email, int id, String attempt);

}
```

AuthenticatedWebPage.java

```
package org.swainston;

/**
* Marker interface to indicate that a page should be viewable by
authenticated users only.
*/
public interface AuthenticatedWebPage {}
```

Challenge.java

```
package org.swainston;

/**
* Class that holds information surrounding a challenge
*/
public class Challenge {

 private Integer id;
 private String title;
 private String description;
 private String solutionTemplate;

 /**
  *
  */
 public Challenge() {
 }

 /**
  * Fetches solution template for a challenge
  *
  * @return set template for the solution defined in the database
  */
 public String getSolutionTemplate() {
   return solutionTemplate;
 }

 /**
  * Sets the template for a certain solution
  *
  * @param solutionTemplate the template for the solution
  */
 public void setSolutionTemplate(String solutionTemplate) {
   this.solutionTemplate = solutionTemplate;
 }

 /**
  * @return id of a challenge
  */
 public Integer getId() {
   return id;
 }

 /**
  * Sets the id of a challenge
  *
  * @param id the challenge id
  */
 public void setId(Integer id) {
   this.id = id;
 }
```

```
  /**
   * @return title of the challenge
   */
  public String getTitle() {
    return title;
  }

  /**
   * Sets the title of a challenge
   *
   * @param title the new title
   */
  public void setTitle(String title) {
    this.title = title;
  }

  /**
   * @return the description of a challenge
   */
  public String getDescription() {
    return description;
  }

  /**
   * Sets the description of a challenge
   *
   * @param description the new description
   */
  public void setDescription(String description) {
    this.description = description;
  }
}
```

ChallengePageForm.java

```
package org.swainston;

import org.apache.wicket.markup.html.form.Form;
import org.apache.wicket.markup.html.form.TextArea;
import org.apache.wicket.model.IModel;
import org.apache.wicket.model.Model;
import org.apache.wicket.model.PropertyModel;
import org.swainston.tests.UpperCaseChallenge.UpperCaseChallenge;
import org.swainston.tests.UpperCaseChallenge.UpperCaseChallengeImplTest;
import org.swainston.ui.challengepage.ChallengePage;

import java.lang.reflect.InvocationTargetException;
import java.util.Collections;

/**
* Students' code is compiled and tests called on it within this class
*/
public class ChallengePageForm extends Form<Attempt> {

 private final Challenge challenge;
 private final String user;
 private final AttemptsStore attemptsStore;

 /**
  * ChallengePageForm object
  *
  * @param user          the student
  * @param challenge     the challenge the student is attempting
  * @param attemptsStore the attempt store that holds the student's attempts
  */
 public ChallengePageForm(String user, Challenge challenge, AttemptsStore
attemptsStore) {
   super("form-submit-code");

   this.challenge = challenge;
   this.user = user;
   this.attemptsStore = attemptsStore;

   Attempt modelObject = attemptsStore.getAttempt(user, challenge.getId()).get();
   setModel(new Model<>(modelObject));
   add(new TextArea<>("textarea-code",
       new PropertyModel<Attempt>(modelObject, "attempt")));
 }

 /**
  * The function that is run when the student presses the submit button
  */
 @Override
 protected void onSubmit() {

   TextArea<Attempt> textArea = (TextArea<Attempt>) get("textarea-code");
   IModel<Attempt> textAreaModel = textArea.getModel();

   String studentCode = "";
```

```
    String error = "";
    Compiler.Result result = null;
    try {

      Object attemptObject = textAreaModel.getObject();
      studentCode = (String) attemptObject;

      String codeImplementingTest = studentCode.replace("class Challenge8",
          "class Challenge8 implements " +
              "org.swainston.tests.UpperCaseChallenge.UpperCaseChallenge");
      Attempt attempt = new Attempt();
      attempt.setAttempt(codeImplementingTest);

      System.out.println("[" + codeImplementingTest + "]");

      result = Compiler.checkCompiles(attempt);


      if (result.isCompiles()) {
        var byteArray = result.getByteArrayOutputStream().toByteArray();


        AttemptClassLoader attemptClassLoader = new AttemptClassLoader(byteArray);
        try {
          String source_name = "org.swainston.Challenge8";
          Class<?> aClass = attemptClassLoader.findClass(source_name);
          Object object = aClass.getDeclaredConstructor().newInstance();

          UpperCaseChallenge upperCaseChallenge =
UpperCaseChallenge.class.cast(object);
          UpperCaseChallengeImplTest upperCaseChallengeImplTest = new
UpperCaseChallengeImplTest();
          upperCaseChallengeImplTest.setUpperCaseChallenge(upperCaseChallenge);

          try {
            upperCaseChallengeImplTest.upperCaseConverter_all_num();
            upperCaseChallengeImplTest.upperCaseConverter_all_chars();
            upperCaseChallengeImplTest.upperCaseConverter_all_spaces();
            upperCaseChallengeImplTest.upperCaseConverter_empty_string();
            upperCaseChallengeImplTest.upperCaseConverter_null();
          } catch (Error e) {
            result.setErrors(Collections.singletonList(e.toString()));
          }

        } catch (InstantiationException | IllegalAccessException |
NoSuchMethodException |
                 InvocationTargetException e) {
          throw new RuntimeException(e);
        }
      }
    } catch (Error | Exception e) {
      error = e.toString();
    }

    // Store the original code that the student submitted, not the code modified to
implement the
    // test methods.
    if (!error.isEmpty()) {
```

```
      studentCode += "\n============================\n" + error;
    }

    if (result != null) {
      result.setSubmission(studentCode);
      attemptsStore.setAttempt(user, challenge.getId(), result.toString());
    } else {
      System.out.println("Unexpected error");
    }

    setResponsePage(ChallengePage.class);
  }
}
```

ChallengeStore.java

```
package org.swainston;

import java.util.Optional;

public interface ChallengeStore {
 Optional<Challenge> getNextChallenge(String user);
}
```

Compiler.java

```
package org.swainston;

import javax.tools.*;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.net.URI;
import java.util.ArrayList;
import java.util.List;

/**
* Taken from <a
*
href="https://docs.oracle.com/javase/8/docs/api/javax/tools/JavaCompiler.html">Java
doc</a>
*/
public class Compiler {

 public static Result checkCompiles(Attempt challenge) {
   /**
    * A file object used to represent source coming from a string.
    */
   class JavaSourceFromString extends SimpleJavaFileObject {
     /**
      * The source code of this "file".
      */
     final String code;

     /**
      * Constructs a new JavaSourceFromString.
      *  @param name the name of the compilation unit represented by this file
object
      *  @param code the source code for the compilation unit represented by this
file object
      */
     JavaSourceFromString(String name, String code) {
       super(URI.create(
               String.format("string:///%s%s", name.replace('.', '/'),
Kind.SOURCE.extension)),
           Kind.SOURCE);
       this.code = code;
     }

     @Override
     public CharSequence getCharContent(boolean ignoreEncodingErrors) {
       return code;
     }
   }

   Result result = new Result(challenge.getAttempt());
   String submission = result.getSubmission();

   JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();
   DiagnosticCollector<JavaFileObject> diagnosticCollector = new
DiagnosticCollector<>();
```

```
   JavaSourceFromString source = new JavaSourceFromString("Challenge8",
submission); // TODO
   List<JavaSourceFromString> sources = new ArrayList<>();
   sources.add(source);

   List<String> options = List.of("-Xlint:unchecked");
   StandardJavaFileManager mgr = compiler.getStandardFileManager(null, null, null);

   JavaFileManager fileManager = new ForwardingJavaFileManager(mgr) {
     @Override
     public JavaFileObject getJavaFileForOutput(Location location, String
className,
                                                JavaFileObject.Kind kind,
FileObject sibling)
         throws IOException {
       JavaFileObject simpleJavaFileObject =
           new SimpleJavaFileObject(URI.create(className), kind) {
             @Override
             public OutputStream openOutputStream() {
               ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream();
               result.setByteArrayOutputStream(byteArrayOutputStream);
               return byteArrayOutputStream;
             }
           };
       return simpleJavaFileObject;
     }
   };


   JavaCompiler.CompilationTask task =
       compiler.getTask(null, fileManager, diagnosticCollector, options, null,
sources);
   task.call();
   List<String> errors = new ArrayList<>();
   for (Diagnostic<? extends JavaFileObject> diagnostic :
diagnosticCollector.getDiagnostics()) {
     errors.add(String.format("Line: %d, %s in %s", diagnostic.getLineNumber(),
         diagnostic.getMessage(null), diagnostic.getSource().getName()));
   }

   result.setCompiles(errors.isEmpty());
   result.setErrors(errors);

   return result;
 }

 public static class Result {

   private static final String MARKER = "\n============================";

   private boolean compiles;
   private List<String> errors;
   private String submission;
   private ByteArrayOutputStream byteArrayOutputStream;

   Result(String attempt) {
     int index = attempt.indexOf(MARKER);
```

```
      if (index > 0) {
        attempt = attempt.substring(0, index);
      }
      submission = attempt;
    }

    public ByteArrayOutputStream getByteArrayOutputStream() {
      return byteArrayOutputStream;
    }

    public void setByteArrayOutputStream(ByteArrayOutputStream
byteArrayOutputStream) {
      this.byteArrayOutputStream = byteArrayOutputStream;
    }

    public String getSubmission() {
      return submission;
    }

    public void setSubmission(String submission) {
      this.submission = submission;
    }

    public boolean isCompiles() {
      return compiles;
    }

    public void setCompiles(boolean compiles) {
      this.compiles = compiles;
    }

    public List<String> getErrors() {
      return errors;
    }

    public void setErrors(List<String> errors) {
      this.errors = errors;
    }

    @Override
    public String toString() {
      return getSubmission() + MARKER + "\n" + getErrors();
    }
  }
}
```

CredentialsStore.java

```
package org.swainston;

public interface CredentialsStore {
 boolean add(String email, String password);

 boolean exists(String email);

 boolean validate(String email, String password);
}
```

InMemoryCredentialsStore.java

```
package org.swainston;

import java.util.HashMap;
import java.util.Map;

/**
* This class is only used for local testing prior to the implementation of
a database.
*
* @see org.swainston.database.MySQLCredentialsStore
*/
public class InMemoryCredentialsStore implements CredentialsStore {

 private final Map<String, String> credentialMap = new HashMap<>();

 @Override
 public boolean add(String email, String password) {
   if (!exists(email)) {
     credentialMap.put(email, password);
     System.out.println(credentialMap.get(email) + " Is String: " +
(credentialMap.get(email) instanceof String));
     return true;
   } else {
     return false;
   }
 }

 @Override
 public boolean exists(String email) {
   return credentialMap.containsKey(email);
 }

 @Override
 public boolean validate(String email, String password) {
   return (credentialMap.containsKey(email) &&
credentialMap.get(email).equals(password));
 }
}
```

SignInSession.java

```
package org.swainston;

import org.apache.wicket.authroles.authentication.AuthenticatedWebSession;
import org.apache.wicket.authroles.authorization.strategies.role.Roles;
import org.apache.wicket.request.Request;
import org.swainston.database.MySQLCredentialsStore;

/**
* Used for verifying credentials for sessions
*/
public final class SignInSession extends AuthenticatedWebSession {

 // Credential store is set to desired store
 private final CredentialsStore credentialsStore = new MySQLCredentialsStore();

 private String username;

 /**
  * Constructing method that is run when the class is called
  * @param request the request object (as per apache javadoc)
  */
 public SignInSession(Request request) {
   super(request);
   // Credentials that always exist to allow for testing
   if (!credentialsStore.exists("foo")) {
     credentialsStore.add("foo", "bar");
   }
 }

 /**
  * Trivial sign out method
  */
 @Override
 public void signOut() {
   super.signOut();
 }

 /**
  * Authenticates the given username and password.
  *
  * @param username the username
  * @param password the password
  *
  * @return true if the user was authenticated
  */
 @Override
 public boolean authenticate(final String username, final String password) {
   if (credentialsStore.validate(username, password)) {
     this.username = username;
   } else {
     this.username = null;
   }
   return this.username != null;
 }

 /**
```

```
   * Validates whether a user exists
   *
   * @param username the username
   *
   * @return whether the user exists
   */
  public boolean userExists(String username) {
    return credentialsStore.exists(username);
  }

  /**
   * Creates a new user in the credential store
   *
   * @param username the validated username
   * @param password the password
   *
   * @return whether the sign-up was a success
   */
  public boolean signUp(final String username, final String password) {
    if (!userExists(username)) {
      return credentialsStore.add(username, password);
    } else {
      return false;
    }
  }

  /**
   *  Getter method for the username attribute
   * @return the username
   */
  public String getUser() {
    return username;
  }

  /**
   * @param user New user
   */
  public void setUser(final String user) {
    this.username = user;
  }

  @Override
  public Roles getRoles() {
    return null;
  }
}
```

Start.java

```
package org.swainston;

import org.eclipse.jetty.jmx.MBeanContainer;
import org.eclipse.jetty.server.HttpConfiguration;
import org.eclipse.jetty.server.HttpConnectionFactory;
import org.eclipse.jetty.server.Server;
import org.eclipse.jetty.server.ServerConnector;
import org.eclipse.jetty.webapp.WebAppContext;

import javax.management.MBeanServer;
import java.lang.management.ManagementFactory;

/**
* Separate startup class for people that want to run the examples directly. Use
parameter
* -Dcom.sun.management.jmxremote to startup JMX (and e.g. connect with jconsole).
*/
public class Start {

 /**
  * Main function for the Wicket Application host
  */
 public static void main(String[] args) {
   // Assigned system property - can be retrieved using
System.getProperty("wicket.configuration")
   System.setProperty("wicket.configuration", "development");

   Server server = new Server();

   HttpConfiguration httpConfiguration = new HttpConfiguration();
   httpConfiguration.setSecureScheme("https");
   httpConfiguration.setSecurePort(8443);
   httpConfiguration.setOutputBufferSize(32768);

   ServerConnector connector = new ServerConnector(server, new
HttpConnectionFactory(httpConfiguration));
   // Sets the port for the server to run on
   connector.setPort(8080);
   connector.setIdleTimeout(1000 * 60 * 60);

   server.addConnector(connector);

   WebAppContext webAppContext = new WebAppContext();
   webAppContext.setServer(server);
   webAppContext.setContextPath("/");
   webAppContext.setWar("src/main/webapp");

   server.setHandler(webAppContext);

   MBeanServer mBeanServer = ManagementFactory.getPlatformMBeanServer();
   MBeanContainer mBeanContainer = new MBeanContainer(mBeanServer);
   server.addEventListener(mBeanContainer);
   server.addBean(mBeanContainer);

   try {
     server.start();
```

```
      server.join();
    } catch (Exception e) {
      e.printStackTrace();
      System.exit(100);
    }
  }
}
```

WicketApplication.java

```
package org.swainston;

import org.apache.wicket.RestartResponseAtInterceptPageException;
import org.apache.wicket.Session;
import org.apache.wicket.authorization.IAuthorizationStrategy;
import org.apache.wicket.csp.CSPDirective;
import org.apache.wicket.csp.CSPDirectiveSrcValue;
import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.protocol.http.WebApplication;
import org.apache.wicket.request.Request;
import org.apache.wicket.request.Response;
import org.apache.wicket.request.component.IRequestableComponent;
import org.swainston.database.Database;
import org.swainston.ui.homepage.HomePage;
import org.swainston.ui.signin.SignInPage;

import javax.sql.DataSource;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
* Application object for your web application. If you want to run this application without
* deploying, run the Start class.
*/
public class WicketApplication extends WebApplication {
 /**
  * @see org.apache.wicket.Application#getHomePage()
  */
 @Override
 public Class<? extends WebPage> getHomePage() {
   return HomePage.class;
 }

 private static DataSource DATASOURCE;

 public static Connection getConnection() throws SQLException {
   return DATASOURCE.getConnection();
 }

 @Override
 public Session newSession(Request request, Response response) {
   return new SignInSession(request);
 }

 /**
  * @see org.apache.wicket.Application#init()
  */
 @Override
 public void init() {

   try {
     DATASOURCE = Database.initDataSource();
     initDB();
   } catch (SQLException | IOException e) {
     e.printStackTrace();
   }

   super.init();
```

```
   // Required to allow custom fonts to be displayed and not be blocked by Content Security
Policy
   getCspSettings().blocking()
        .add(CSPDirective.STYLE_SRC, CSPDirectiveSrcValue.SELF)
        // Google fonts
        .add(CSPDirective.STYLE_SRC, "https://fonts.googleapis.com/css")
        // FontAwesome fonts stylesheet
        .add(CSPDirective.STYLE_SRC,

"https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css")
        // FontAwesome custom font
        .add(CSPDirective.FONT_SRC,

"https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/fonts/fontawesome-webfont.woff2"
)
        .add(CSPDirective.FONT_SRC,

"https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/fonts/fontawesome-webfont.woff")
        .add(CSPDirective.FONT_SRC,

"https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/fonts/fontawesome-webfont.ttf");

    //   .add(CSPDirective.SCRIPT_SRC, "https://kit.fontawesome.com/79cfa1bf9a.js");



   // Register the authorization strategy
   getSecuritySettings().setAuthorizationStrategy(
        new IAuthorizationStrategy.AllowAllAuthorizationStrategy() {
          @Override
          public <T extends IRequestableComponent> boolean isInstantiationAuthorized(
              Class<T> componentClass) {
            // Check if the page requires auth, if it does it will implement marker class, and
checks
            // if the session is signed in
            if (AuthenticatedWebPage.class.isAssignableFrom(componentClass)) {
              if (((SignInSession) Session.get()).isSignedIn()) {
                return true;
              }
              // Session not signed in and page requires auth. Send them to auth, then
redirect
              //  to the original page
              // continueToOriginalDestination when logged in.
              throw new RestartResponseAtInterceptPageException(SignInPage.class);
            }
            // The Page we're being directed to does not require authentication.
            return true;
          }
        });
 }


 /**
  * Initialises the database and uses the <a href="setUpDB.sql">setUpDB.sql</a> to populate
the
  * database.
  *
  * @throws SQLException if prepared statement fails to execute.
  * @throws IOException  if content within the SQL file is invalid.
  */
 private void initDB() throws SQLException, IOException {
   String setup;
   try (InputStream inputStream =
getClass().getClassLoader().getResourceAsStream("setUpDB.sql")) {
```

```
      if (inputStream != null) {
        setup = new String(inputStream.readAllBytes());
      } else {
        throw new AssertionError("Failed to read bytes of resource.");
      }
    } catch (IOException ex) {
      Logger.getLogger(WicketApplication.class.getName()).log(Level.SEVERE,
ex.getMessage());
      return;
    }
    String[] queries = setup.split(";");
    for (String query : queries) {
      try (Connection conn = DATASOURCE.getConnection();
           PreparedStatement stmt = conn.prepareStatement(query)) {
        stmt.execute();
      }
    }
    System.out.println("Database loaded & ready!");
  }
}
```

style.css

```
body {
   background: #0f0f23;
   font-family: "Source Code Pro", monospace;
   color: #cccccc;
}

.navbar {
   overflow: hidden;
   background-color: #009900;
   border-radius: 15px;
}

.navbar a {
   float: left;
   color: #f2f2f2;
   text-align: center;
   padding: 14px 16px;
   text-decoration: none;
   font-size: 24px;
}

.navbar a:hover {
   color: black;
   background-color: white;
}

.navbar a#signout {
   float: right;
}

.navbar a#settings {
   float: right;
}

.main {
   text-align: center;
}

.main p {
   width: 50%;
   margin: auto;
}

.p a {
   color: #cccccc;
}

a:-webkit-any-link {
   color: white;
   cursor: pointer;
   text-decoration: underline;
```

```
    text-align: center;
}
```

Pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Licensed to the Apache Software Foundation (ASF) under one or more
  contributor license agreements.  See the NOTICE file distributed with
  this work for additional information regarding copyright ownership.
  The ASF licenses this file to You under the Apache License, Version 2.0
  (the "License"); you may not use this file except in compliance with
  the License.  You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
-->
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">

   <modelVersion>4.0.0</modelVersion>
   <groupId>org.swainston</groupId>
<!--    <artifactId>ALevelProjectTest</artifactId>-->
   <artifactId>ALevelProject</artifactId>
   <packaging>war</packaging>
   <version>1.0-SNAPSHOT</version>
   <name>quickstart</name>
   <description></description>
   <!--
      <organization>
         <name>company name</name>
         <url>company url</url>
      </organization>
   -->
   <licenses>
      <license>
         <name>The Apache Software License, Version 2.0</name>
         <url>http://www.apache.org/licenses/LICENSE-2.0.txt</url>
         <distribution>repo</distribution>
      </license>
   </licenses>
   <properties>
      <wicket.version>9.14.0</wicket.version>
      <jetty9.version>9.4.51.v20230217</jetty9.version>
      <slf4j.version>2.0.7</slf4j.version>
      <junit.version>5.9.3</junit.version>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <!-- allowed values: R7, 1.0, 1.5, 2.0 or none -->
      <wtp.version>none</wtp.version>
   </properties>
   <dependencies>
      <!--  WICKET DEPENDENCIES -->
      <dependency>
         <groupId>org.apache.wicket</groupId>
```

```
            <artifactId>wicket-core</artifactId>
            <version>${wicket.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.wicket</groupId>
            <artifactId>wicket-auth-roles</artifactId>
            <version>${wicket.version}</version>
        </dependency>

        <!-- LOGGING DEPENDENCIES - SLF4J-SIMPLE -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <!--  JUNIT DEPENDENCY FOR TESTING -->

        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-api</artifactId>
            <version>5.7.2</version>
<!--            <scope>test</scope>-->
        </dependency>

        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>5.7.2</version>
<!--            <scope>test</scope>-->
        </dependency>

        <dependency>
            <groupId>org.eclipse.jetty</groupId>
            <artifactId>jetty-http</artifactId>
            <version>${jetty9.version}</version>
        </dependency>


        <!--  JETTY DEPENDENCIES FOR TESTING  -->
        <dependency>
            <groupId>org.eclipse.jetty</groupId>
            <artifactId>jetty-webapp</artifactId>
<!--            <scope>test</scope>-->
            <version>${jetty9.version}</version>
        </dependency>

        <dependency>
            <groupId>org.eclipse.jetty</groupId>
            <artifactId>jetty-jmx</artifactId>
            <version>${jetty9.version}</version>
<!--            <scope>test</scope>-->
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.32</version>
```

```
    </dependency>

</dependencies>
<build>
    <resources>
        <resource>
            <filtering>false</filtering>
            <directory>src/main/resources</directory>
        </resource>
        <resource>
            <filtering>false</filtering>
            <directory>src/main/java</directory>
            <includes>
                <include>**</include>
            </includes>
            <excludes>
                <exclude>**/*.java</exclude>
            </excludes>
        </resource>
    </resources>
    <testResources>
        <testResource>
            <filtering>false</filtering>
            <directory>src/test/resources</directory>
        </testResource>
        <testResource>
            <filtering>false</filtering>
            <directory>src/test/java</directory>
            <includes>
                <include>**</include>
            </includes>
            <excludes>
                <exclude>**/*.java</exclude>
            </excludes>
        </testResource>
    </testResources>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.10.1</version>
            <inherited>true</inherited>
            <configuration>
                <source>11</source>
                <target>11</target>
                <encoding>UTF-8</encoding>
                <showWarnings>true</showWarnings>
                <showDeprecation>true</showDeprecation>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
            <version>3.3.2</version>
        </plugin>
        <plugin>
            <groupId>org.eclipse.jetty</groupId>
            <artifactId>jetty-maven-plugin</artifactId>
```

```
            <version>${jetty9.version}</version>
            <configuration>
               <systemProperties>
                  <systemProperty>
                     <name>maven.project.build.directory.test-classes</name>
                     <value>${project.build.directory}/test-classes</value>
                  </systemProperty>
               </systemProperties>

<jettyXml>${project.basedir}/src/test/jetty/jetty.xml,${project.basedir}/src/test/j
etty/jetty-ssl.xml,${project.basedir}/src/test/jetty/jetty-http.xml,${project.based
ir}/src/test/jetty/jetty-https.xml</jettyXml>
            </configuration>
         </plugin>
         <plugin>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.22.2</version>
         </plugin>
         <plugin>
            <artifactId>maven-failsafe-plugin</artifactId>
            <version>2.22.2</version>
         </plugin>
      </plugins>
   </build>

   <repositories>
      <repository>
         <id>Apache Nexus</id>
         <url>https://repository.apache.org/content/repositories/snapshots/</url>
         <releases>
            <enabled>false</enabled>
         </releases>
         <snapshots>
            <enabled>true</enabled>
         </snapshots>
      </repository>
   </repositories>
   </project>
```