



NUS SCHOOL OF COMPUTING

AY 2021/2022 SEMESTER 1

CS3219 SOFTWARE ENGINEERING PRINCIPLES AND PATTERNS

FINAL REPORT

Prepared By:

Group 18

Name Student ID

Cheak Han Wei A0196589B

Foo Kai En A0205204Y

Koh Rui Ling A0206345L

Low Shu Min Samantha A0205011H

Table of Contents

1 Introduction	5
1.1 Background and Purpose	5
1.2 SHReK Tech	5
2 Individual Contributions	7
3 Requirements of SHReK Tech	9
3.1 Functional Requirements	9
F1. Authentication	10
F2. Starting and Ending a Technical Interview	12
F3. Matching Users	14
F4. Technical Interview — Coding Questions	16
F5. Technical Interview — Interview Questions	18
F6. Technical Interview — Communication	19
3.2 Non-Functional Requirements	21
NF1. Security	22
NF2. Performance	24
NF3. Usability	26
4 Architecture Design of SHReK Tech	28
4.1 4-Tier Architecture	28
4.1.1 4-Tier Architecture Design	28
4.1.2 Rationale for 4-Tier Architecture Design	29
4.2 Vue Single Page Application	29
4.2.1 Rationale for Vue SPA	29
5 Design of Key Components of SHReK Tech	30
5.1 Authentication — User Account Creation	30
5.1.1 Design of User Account Creation	30
5.1.2 Rationale for Design of User Account Creation	31
5.2 Authentication — Session Token Management	31
5.2.1 Design of Session Token Management	31
5.2.2 Rationale for Design of Session Management	31

5.3 Authentication — Sensitive Request Management	33
5.3.1 Design of Sensitive Request Management	33
5.3.2 Rationale for Design of Sensitive Request Management	33
5.4 Waiting User Status	33
5.4.1 Design of Waiting User Status	33
5.4.2 Rationale for Design of Waiting User Status	34
5.5 Real-Time Communication Components	34
5.5.1 Design of Real-Time Communication Components	34
5.5.2 Rationale for Design of Real-Time Communication Components	35
5.6 Code Editor	36
5.6.1 Design of Code Editor	36
5.6.2 Rationale for Design of Code Editor	36
5.7 Coding Room	37
5.7.1 Design of Coding Room	37
5.7.2 Rationale for Design of Coding Room	37
5.8 Coding Questions	37
5.8.1 Design of Coding Questions	37
5.8.2 Rationale for Design of Coding Questions	38
6 DevOps	39
6.1 Agile Software Development Process	39
6.2 Continuous Integration/Continuous Deployment	39
6.2.1 Design and Rationale for CI/CD Pipeline	39
6.2.2 Implementation of CI/CD Pipeline	40
7 User Interface of SHReK Tech	42
7.1 Landing View	42
7.2 Home View	44
7.3 Matching View	45
7.4 Coding Room View	47
8 Improvements and Enhancements	49
8.1 Find a Match By Topic	49
8.2 Join a Technical Interview via an Invite Link	49

8.3 In-Built Compiler	49
8.4 Voice Chat/Video Conferencing	49
9 Reflections and Learning Points	50
10 Appendix	52
Appendix A. CI Test Cases	52
Test Case 1. Auth	52
Test Case 2. Coding Questions	52
Test Case 3. Interview Questions	53
Test Case 4. Sockets	53
Test Case 5. Sessions	54

1 Introduction

This final report outlines the development process and design decisions of our project, SHReK Tech, embarked on as part of the coursework for module CS3219 Software Engineering Principles and Patterns. This report provides insight into the requirements of our application, developer documentation, further enhancements which we hope to implement and our reflections throughout this learning journey.

1.1 Background and Purpose

By 2025, 98 million new jobs which require software development skills will be created around the world¹. To meet this demand, the number of freshmen enrolled in computing courses have rapidly increased and this number has almost tripled in the past five years for NUS School of Computing^{2,3}. A critical challenge that these undergraduates and job seekers must overcome to land their dream internships and jobs in technical roles is whiteboard-style technical interviews. Coupled with inspiration taken from PeerPrep⁴, the idea to create a solution for students and job seekers to brush up on their technical interview skills was formed.

1.2 SHReK Tech

SHReK Tech is a platform where users can collaborate and solve coding questions while articulating their thought process in a technical interview setting. The name SHReK is a quirky twist on the initials of the names of our group members — 'S' for Samantha, 'H' for Han Wei, 'R' for Rui Ling and 'K' for Kai En. To strengthen the brand identity of SHReK Tech, our UI takes on a green and brown theme, reminiscent of the green ogre, Shrek.

Having taken inspiration from PeerPrep, SHReK Tech offers similar features such as authentication, allowing users to select their preferred difficulty of coding questions, matchmaking, collaborative programming and communicating through a chatbox. Beyond these features, a unique selling point (USP) of SHReK Tech is the assignment of interviewer and interviewee roles to users to better simulate a technical interview. Users assuming the interviewer role will have access to a series of common interview questions which they can prompt their partners with.

¹ <https://businesstech.co.za/news/technology/521358/30-of-the-most-in-demand-skills-that-will-help-you-get-a-job-in-the-next-five-years/>

² <https://www.nus.edu.sg/registrar/docs/info/student-statistics/enrolment-statistics/undergraduate-studies/ug-enrol-20162017.pdf>

³ <https://www.nus.edu.sg/registrar/docs/info/student-statistics/enrolment-statistics/undergraduate-studies/ug-enrol-20202021.pdf>

⁴ <https://github.com/CS3219-SF-Principles-and-Patterns/cs3219-ay2021-s1-project-2020-s1-q23>

Furthermore, to aid users in finding a match to improve user experience, users are able to view if there is already a waiting user for each of the three difficulty levels for coding questions. In the event that there are no available matches but a user would still like to practise, users are able to proceed without having a match.

2 Individual Contributions

Name	Technical Contributions	Non-Technical Contributions
Cheak Han Wei	<p>Implement features</p> <ul style="list-style-type: none"> • Coding questions scraper <p>Write unit tests for</p> <ul style="list-style-type: none"> • Coding questions API endpoints • Generating coding questions of the right difficulty level <p>Fix bugs</p>	<ul style="list-style-type: none"> • Document project requirements • Contribute to final report • Design UI components for implemented features • Styling standardisation for all views
Foo Kai En	<p>Set up</p> <ul style="list-style-type: none"> • Frontend, server and MongoDB • CI • CD with Rui Ling <p>Implement features</p> <ul style="list-style-type: none"> • Matching • Waiting room status • Interview questions • Interview roles • CRDT • Authentication for REST API <p>Write unit tests for</p> <ul style="list-style-type: none"> • Ongoing user session API endpoint • Interview questions API endpoint • Socket events <p>Fix bugs</p>	<ul style="list-style-type: none"> • Document project requirements • Contribute to final report • Design UI components for implemented features • Update UI design to match 'SHReK' theme

Koh Rui Ling	<p>Set up</p> <ul style="list-style-type: none"> • CD with Kai En <p>Implement features</p> <ul style="list-style-type: none"> • Real-time chat and typing events • Real-time code editor • Timer component <p>Fix bugs</p>	<ul style="list-style-type: none"> • Document project requirements • Contribute to final report • Design UI components for implemented features
Low Shu Min Samantha	<p>Set up</p> <ul style="list-style-type: none"> • Database in GitHub Actions for CI <p>Implement features</p> <ul style="list-style-type: none"> • Sign up and email verification • Log in/Log out • Reset/Change password • Authentication for API endpoints <p>Write unit tests for</p> <ul style="list-style-type: none"> • Authentication API endpoints • User API endpoints <p>Fix bugs</p>	<ul style="list-style-type: none"> • Document project requirements • Contribute to final report • Design UI components for implemented features

3 Requirements of SHReK Tech

We utilised the Must-Should-Could-Wish prioritisation scheme to rank our functional and non-functional requirements. We have defined each rank as such:

Must	Requirements critical to the project for it to be considered a success. If any of these requirements are not satisfied, the project is a failure. Musts are essentially the very basic and fundamental requirements that the system should have to serve its purpose.
Should	Requirements that are important but not time pressing. These requirements could be thought of as being as important as Musts but are not as time-critical and can be delivered at a later date.
Could	Requirements that are desirable but not necessary. These requirements can be thought of as requirements that improve user experience.
Wish	Requirements that are desirable but may be difficult to achieve given the time frame of this project.

3.1 Functional Requirements

We have categorised the functional requirements by the following purposes:

- Authentication
- Starting and Ending a Technical Interview
- Matching Users
- Technical Interview — Coding Question
- Technical Interview — Interview Questions
- Technical Interview — Communication

Additionally, we have utilised a requirements traceability matrix to track the running history of our FRs through identification, design, code implementation and test.

F1. Authentication

S/N	Requirement	Priority	Design Reference	Code Reference	Test Case Reference
F1.1	The system should allow users to create an account with a username, email and password.	Must	Section 5.1	authController.js createUser()	authControllerTest.js 1.1.1 - 1.1.3
F1.2	The system should ensure that every account created has a unique username.	Must	Section 5.1	authController.js createUser()	authControllerTest.js 1.1.4
F1.3	The system should ensure that every account created has a unique email.	Must	Section 5.1	authController.js createUser()	authControllerTest.js 1.1.5
F1.4	The system should allow users to verify their email on account creation to ensure they have provided a valid email that they have access to.	Must	Section 5.1	authController.js verifyAccount Email()	authControllerTest.js 1.2.3
F1.5	The system should allow users to log into their accounts by entering their username and password.	Must	Section 5.2	authController.js userLogin()	authControllerTest.js 1.2.1 - 1.2.2
F1.6	The system should allow users to log out of their account.	Must	Section 5.2	authController.js userLogout()	authControllerTest.js 1.5.2

F1.7	The system should allow users to delete their account.	Should	Section 5.3	authController.js deleteUser()	authControllerTest.js 1.4.3 - 1.4.4
F1.8	The system should allow users to change their password.	Should	Section 5.3	authController.js updatePassword()	authControllerTest.js 1.4.1 - 1.4.2
F1.9	The system should allow users to reset their password through an email sent to their registered email account.	Should	Section 5.1	authController.js resetPassword Email() resetPassword()	authControllerTest.js 1.3.1

F2. Starting and Ending a Technical Interview

S/N	Requirement	Priority	Design Reference	Code Reference	Test Case Reference
F2.1	The system should allow users to start a technical interview by choosing the difficulty level of the coding question they wish to attempt.	Must	N.A.	socketController.js handleFindMatch Event()	socketControllerTest.js 4.2.1 - 4.2.2
F2.2	The system should allow users to end the technical interview.	Must	N.A.	socketController.js handleDisconnect EventWhenCoding()	socketControllerTest.js 4.7.3
F2.3	The system should allow users to see which difficulty levels have another user waiting for a match.	Should	Section 5.4	socketController.js handleWaitingUser ListenerEvent()	socketControllerTest.js 4.1.1 - 4.1.2
F2.4	The system should inform the remaining user if the other user in the technical interview has left the session.	Should	N.A.	socketController.js handleDisconnect EventWhenCoding()	socketControllerTest.js 4.7.3
F2.5	The system should only allow a user to start a technical interview if they do not have another ongoing interview.	Could	N.A.	socketController.js hasOngoingSession()	socketControllerTest.js 5.1.1 - 5.1.3

F2.6	The system should allow users to start a technical interview by choosing the topic of the coding question they wish to attempt.	Could	Not implemented
F2.7	The system should allow users to see which topics have another user waiting for a match.	Could	Not implemented

F3. Matching Users

S/N	Requirement	Priority	Design Reference	Code Reference	Test Case Reference
F3.1	The system should match two users who chose the same difficulty level of the coding question they wish to attempt.	Must	N.A.	socketController.js matchUsers()	socketControllerTest.js 4.2.1 - 4.2.2, 4.3.1
F3.2	The system should inform the users that no match is available if a match cannot be found within 30 seconds.	Must	N.A.	Matching.vue matchNotFoundModal component	socketControllerTest.js 4.7.3
F3.3	The system should allow users to choose another difficulty level if a match cannot be found within 30 seconds.	Must	N.A.	Matching.vue matchNotFoundModal component	Passed acceptance test
F3.4	The system should allow users to choose to wait another 30 seconds for the same difficulty level selected if a match cannot be found within 30 seconds.	Must	N.A.	Matching.vue matchNotFoundModal component	Passed acceptance test
F3.5	The system should allow users to choose to proceed without a match if a match cannot be found within 30 seconds.	Could	N.A.	socketController.js handleProceed WithoutMatchEvent()	socketControllerTest.js 4.4.1

F3.6	The system should match two users who chose the same topic of the coding question they wish to attempt.	Could	Not implemented
F3.7	The system should allow users to choose another topic if a match cannot be found within 30 seconds.	Could	Not implemented
F3.8	The system should allow users to choose to wait another 30 seconds for the same topic selected if a match cannot be found within 30 seconds.	Could	Not implemented
F3.9	The system should allow users to send an invite link to others to directly match with them.	Wish	Not implemented

F4. Technical Interview — Coding Questions

S/N	Requirement	Priority	Design Reference	Code Reference	Test Case Reference
F4.1	The system should select a coding question, which is of the preferred difficulty level for the technical interview.	Must	Section 5.8	codingController.js getCodingQuestion Id()	codingControllerTest.js 2.1.1
F4.2	The system should allow matched users in the technical interview to view the coding question.	Must	N.A	N.A	Passed acceptance test
F4.3	The system should select two different coding questions for a technical interview with two users where the two matched users will take turns to be an interviewer and an interviewee.	Should	Section 5.8	codingController.js getCodingQuestion Ids()	codingControllerTest.js 2.1.6 - 2.1.8
F4.4	The system should allow matched users to proceed to the second interview question.	Should	N.A.	socketController.js handleLoadNext QuestionEvent()	socketControllerTest.js 4.9.1
F4.5	The system should display how much time the interviewee has spent on the question.	Should	N.A.	CodingRoom.vue CountUpTimer component	Passed acceptance test

F4.6	The system should have a recommended time limit for each question based on the difficulty level chosen.	Should	N.A.	CodingRoom.vue recommendedTime component	Passed acceptance test
F4.7	The system should have sample test case outputs for users to verify their answers.	Should	N.A.	codingQuestion.js codingQuestion Schema	Passed acceptance test
F4.8	The system should provide links to online solutions of the coding question whenever possible (e.g. geeksforgeeks).	Should	N.A.	codingQuestion.js codingQuestion Schema	Passed acceptance test
F4.9	The system should select coding question(s), which is of the preferred topic of the two matched users, for the technical interview.	Could	Not implemented		
F4.10	The system should have an in-built compiler for users to run their code.	Wish	Not implemented		

F5. Technical Interview — Interview Questions

S/N	Requirement	Priority	Design Reference	Code Reference	Test Case Reference
F5.1	The system should randomly assign one matched user the interviewee role (entity solving the question) and the other the interviewer role (entity providing hints, checking the code).	Should	N.A.	socketController.js randSelect Interviewer()	socketControllerTest.js 4.3.1
F5.2	The system should suggest interview questions for the interviewer to ask the interviewee.	Should	N.A.	interviewQuestions Controller.js getInterview Questions()	interviewQuestions ControllerTest.js 3.1.1 - 3.1.3
F5.3	The system should allow only the interviewer to see the list of suggested interview questions.	Should	N.A.	CodingRoom.vue interviewQuestions Dropdown component	Passed acceptance test
F5.4	The system should allow users who are doing the session alone to view the interview questions.	Could	N.A.	CodingRoom.vue interviewQuestions Dropdown component	Passed acceptance test

F6. Technical Interview — Communication

S/N	Requirement	Priority	Design Reference	Code Reference	Test Case Reference
F6.1	The system should allow both users in a technical interview to write code that is viewable by both users.	Must	Section 5.5	socketController.js handleUpdateCode Event()	socketControllerTest.js 4.8.1
F6.2	The system should allow both users in a technical interview to concurrently edit the code.	Must	Section 5.6	CodingRoom.vue automergeCode data	Passed acceptance test
F6.3	The system should resolve conflicts in the concurrent code edits of both users in a technical interview.	Must	Section 5.6	CodingRoom.vue automergeCode data	Passed acceptance test
F6.4	The system should allow both users in a technical interview to communicate with each other through text.	Must	Section 5.5	socketController.js sendChat()	socketControllerTest.js 4.7.1
F6.5	The systems should allow both users in a technical interview to communicate with each other through voice message.	Wish	Not implemented		

F6.6	The systems should allow both users in a technical interview to communicate with each other through video conferencing.	Wish	Not implemented
-------------	---	------	-----------------

3.2 Non-Functional Requirements

We have categorised our non-functional requirements as such and have listed them in order of decreasing priority:

1. Security
2. Performance
3. Usability

Firstly, security is ranked first as more often than not, software engineers overlook the aspect of security. The scale of the application does not matter when it comes to security as it is important to defend our application from potential attacks and uphold the privacy of our users' data. Only when users know that their data is well-protected will they be willing to use our application, which requires them to provide personal data, such as their emails. Hence, security is ranked above both performance and usability.

Secondly, as SHReK Tech is a collaborative, multi-user application, performance is crucial and hence ranked second. It is important to minimise delays, most notably in the interaction between users, so as to provide a seamless user experience and a real-time environment for collaboration.

Lastly, usability is ranked third as our product is still in its early stages. Since we do not have overly complex features, we do not foresee that users will face difficulty in using our application effectively.

NF1. Security

S/N	Requirement	Reason	Priority	Design Reference	Code Reference	Test Case Reference
NF1.1	Passwords should be hashed when stored in the database.	Provides an extra layer of security in the event that the database is compromised.	Must	Section 5.1	authController.js createUser()	N.A.
NF1.2	Session tokens should be created with a secret key.	Prevents an attacker from generating a valid session token or modifying an existing session token such that it is still valid.	Must	Section 5.2	authController.js userLogin()	authController Test.js 1.5.4
NF1.3	Session tokens should have an expiry of three hours.	In the event that an attacker gains access to a valid session token, the attacker will only have access to the short-lived token.	Must	Section 5.2	authController.js userLogin()	authController Test.js 1.5.5
NF1.4	Sensitive requests (i.e., deleting accounts and changing passwords) require users to re-authenticate.	Provides an additional layer of security based on the Time-of-Check-Time-of-Use (TOCTOU) principle, ensuring that the user is still the same user that	Must	Section 5.3	authController.js deleteUser() updatePassword() resetPassword()	authController Test.js 1.4.2 - 1.4.3

		is authenticated to the system.				
NF1.5	Restricted views should only be visible to users with valid session tokens.	Prevents unauthorised access to SHReKTech.	Must	Section 5.2	client/src/router/index.js beforeEach hook	authControllerTest.js 1.5.2 - 1.5.4
NF1.6	The system should only allow the technical interview to be accessed by its two matched users.	Prevents unauthorised read and write access to the code and chats of the two matched users.	Must	Section 5.7	socketController.js matchUsers()	Passed acceptance test

NF2. Performance

S/N	Requirement	Reason	Priority	Design Reference	Test Case Reference
NF2.1	The delay between the time a user starts looking for a match to the time another user sees that there is a waiting user should be no more than three seconds.	To reduce time spent by another user waiting on the home page to find a match, which may affect user experience.	Must	N.A.	Performance Test Cases Demo (0:00 - 0:05) Passed (~2s)
NF2.2	The delay between one user sending a message and the other user receiving the message should be no more than two seconds.	To allow for “real-time” communication between both users in a session, this delay should be kept to a minimum.	Must	Section 5.5	Performance Test Cases Demo (0:32 - 0:48) Passed (<1s)
NF2.3	The delay between one user writing code and the other user seeing the written code should be no more than two seconds.	To allow for “real-time” collaboration between both users in a session, this delay should be kept to a minimum.	Must	Section 5.5	Performance Test Cases Demo (0:20 - 0:27) Passed (~1s)
NF2.4	The delay between one user leaving the session and the other user seeing the left	To notify the remaining user that the matched user has left the session in the shortest time	Must	N.A.	Performance Test Cases Demo (0:50 - 0:55) Passed (~1s)

	message should be no more than two seconds.	possible.			
--	--	-----------	--	--	--

NF3. Usability

S/N	Requirement	Reason	Priority	Design Reference	Code Reference	Test Case Reference
NF3.1	The application should be usable by a novice who has never attended any technical interview.	As the main objective of our application is to create a platform for users to brush up on their technical interview skills, our target users also include novices who have never attended any technical interview.	Should	Section 7.4	CodingRoom.vue Tooltip component	Passed acceptance test
NF3.2	The system should authenticate a user without them having to enter their username and password if they already have a valid access token.	By persisting access tokens across page reloads, it prevents the need to re-authenticate on every reload. This hence provides a better and more efficient user experience for a frequent user.	Should	Section 5.2	client/src/router/index.js beforeEach hook	Passed acceptance test
NF3.3	The system should allow a	The application should be	Should	Section 5.1	server/src/	authController

	user to recover their account if they forget their password.	usable even for an occasional user who is more likely to forget their password.			controllers/ authController.js resetPasswordEmail	Test.js 1.3.1 - 1.3.3
NF3.4	The system should notify users when there is a waiting user for a difficulty level.	This allows users to find matches more efficiently.	Should	Section 5.4	Home.vue hasWaitingUser icon	Passed acceptance test

4 Architecture Design of SHReK Tech

4.1 4-Tier Architecture

4.1.1 4-Tier Architecture Design

For our project, we adopted the 4-tier architecture design with four closed layers — User Interface, Application, Domain Model and Data, as illustrated in Figure 1. On the client side is a Vue Single Page Application which makes REST API calls to communicate between the User Interface and Application tiers. Where realtime and bidirectional communication is required between these two layers, Socket.IO connections (discussed in [Sections 5.4](#) and [5.5](#)) are also used.

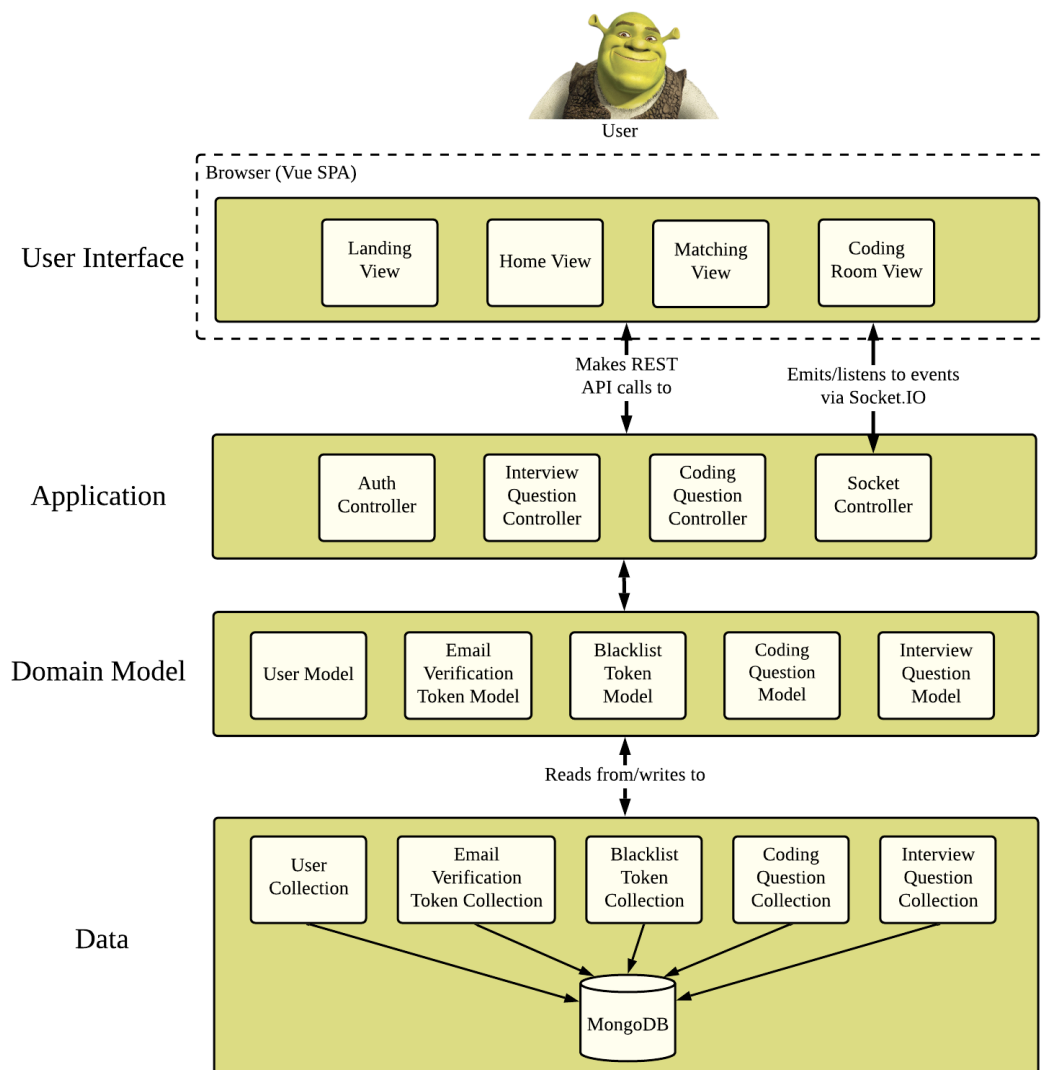


Figure 1: 4-Tier Architecture

4.1.2 Rationale for 4-Tier Architecture Design

A layered architecture brings about horizontally sliced components grouped by functionality. This provides loose coupling, maintainability, extensibility, and efficiency during development.

Firstly, since each layer has its own responsibilities, this allows our components to have a clean separation of concerns and are loosely coupled. Furthermore, having closed layers, whereby control strictly moves from upper layers to lower layers, and vice versa, without skipping any layer, provides even more decoupling.

Secondly, with loose coupling, we can also add new features or modify existing ones easily without affecting other components, hence increasing extensibility. Maintainability which arises from loose coupling is another advantage. By having loosely coupled components and a clear control flow between layers, this makes things clearer and easier to manage and understand when we perform maintenance of our application.

Lastly, this design is simple, efficient and friendly for development. Since we only have limited time to work on this project, the tiered architecture allows us to develop the application and handle the relevant tasks more efficiently.

4.2 Vue Single Page Application

4.2.1 Rationale for Vue SPA

The main consideration for our decision to use the Vue framework is its performance. As mentioned in [Section 3.2](#), performance is crucial to our project due to its collaborative nature. Being an extremely lightweight framework, Vue has proven to be an “exceptionally fast tool” that is recommended for “applications that necessitate speed”⁵. Therefore, we chose to develop a Vue SPA which would help us in fulfilling [NF2](#).

⁵ <https://blog.logrocket.com/angular-vs-react-vs-vue-a-performance-comparison/>

5 Design of Key Components of SHReK Tech

5.1 Authentication — User Account Creation

5.1.1 Design of User Account Creation

Upon every user's first encounter with SHReK Tech, they will have to create an account, as illustrated in Figure 2 with a unique email and a unique username. Before an account is created, the Auth Controller first checks for duplicate emails and usernames. If any of this exists, an error would be returned to the user, informing them that either the email or username exists.

If no duplicate is found, the password will then be hashed with a salt using the JavaScript bcrypt⁶ library. The email, username and hashed password will then be used to create a new User Model. This model is then stored in the User Collection of our database. An email consisting of a verification link with a random token will then be sent to the user's email for account verification. The user will have to verify the account using their email before logging in.

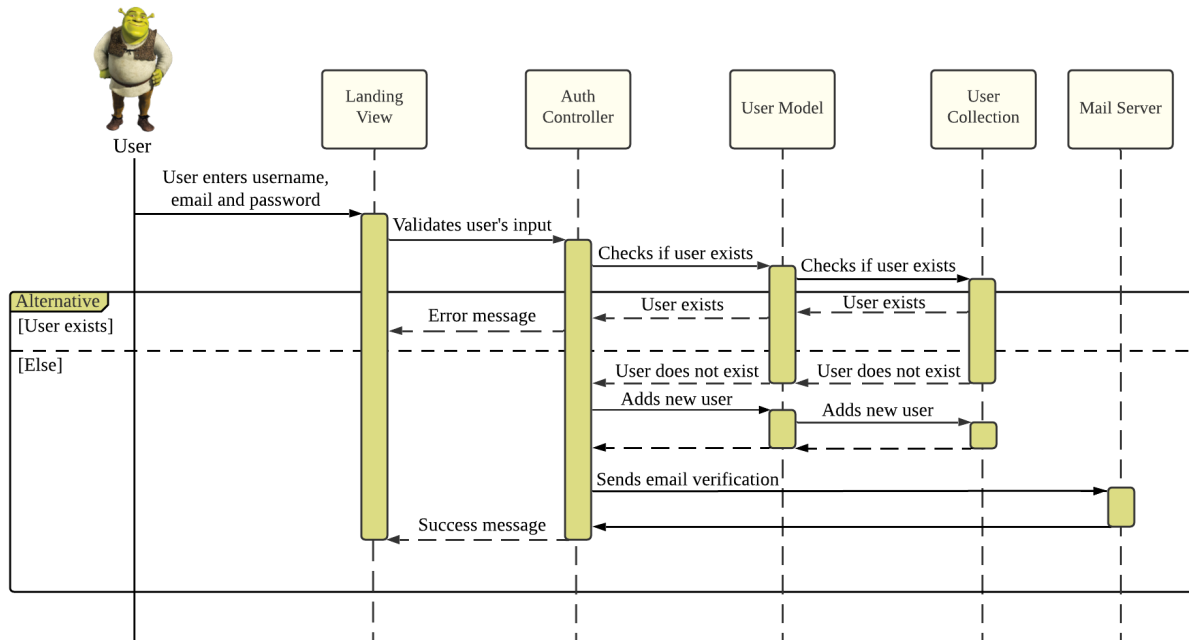


Figure 2: Sequence diagram illustrating how user account creation works

⁶ <https://www.npmjs.com/package/bcrypt>

5.1.2 Rationale for Design of User Account Creation

SHReK Tech requires all user emails to be unique because this email will be used to reset the user's password when necessary and hence, should be able to uniquely identify a user account. The email should also be verified before the user's first login to prevent unauthenticated use of an email address and registration spam.

Although account verification before use may inconvenience users and negatively impact SHReK Tech's usability, this is a necessary tradeoff between security and usability, and we have chosen to prioritise security, as explained in [Section 3.2](#). In addition to that, a verified email address is often the easiest method to recover an account if a user forgets their password, increasing the usability of SHReK Tech in that scenario ([NF3.3](#)).

Hashing passwords with a salt aims to provide an additional layer of security in the event that the database is compromised, which is in line with [NF1.1](#).

5.2 Authentication — Session Token Management

5.2.1 Design of Session Token Management

Upon a user's successful login with their username and password, a JWT, signed with a secret key and created with an expiry of three hours, is stored in the user's session storage. This session token will be attached to all subsequent requests made by the user and validated before any request is served. If the session token is valid, the request will be served. However, in the case where this token is invalid, the request will be rejected and the user is required to login again.

When a user logs out, the user's current session token is blacklisted, such that it can no longer be used as a valid token to perform requests.

5.2.2 Rationale for Design of Session Management

As illustrated in Figure 3, storing a session token allows the user to skip the step of entering their username and password whenever the user attempts to access a restricted page or makes a request which requires authentication. This increases the efficiency of a frequent user of SHReK Tech, in accordance with [NF3.2](#). However, with increased convenience and usability, some precautions are also required to maintain the security of SHReK Tech.

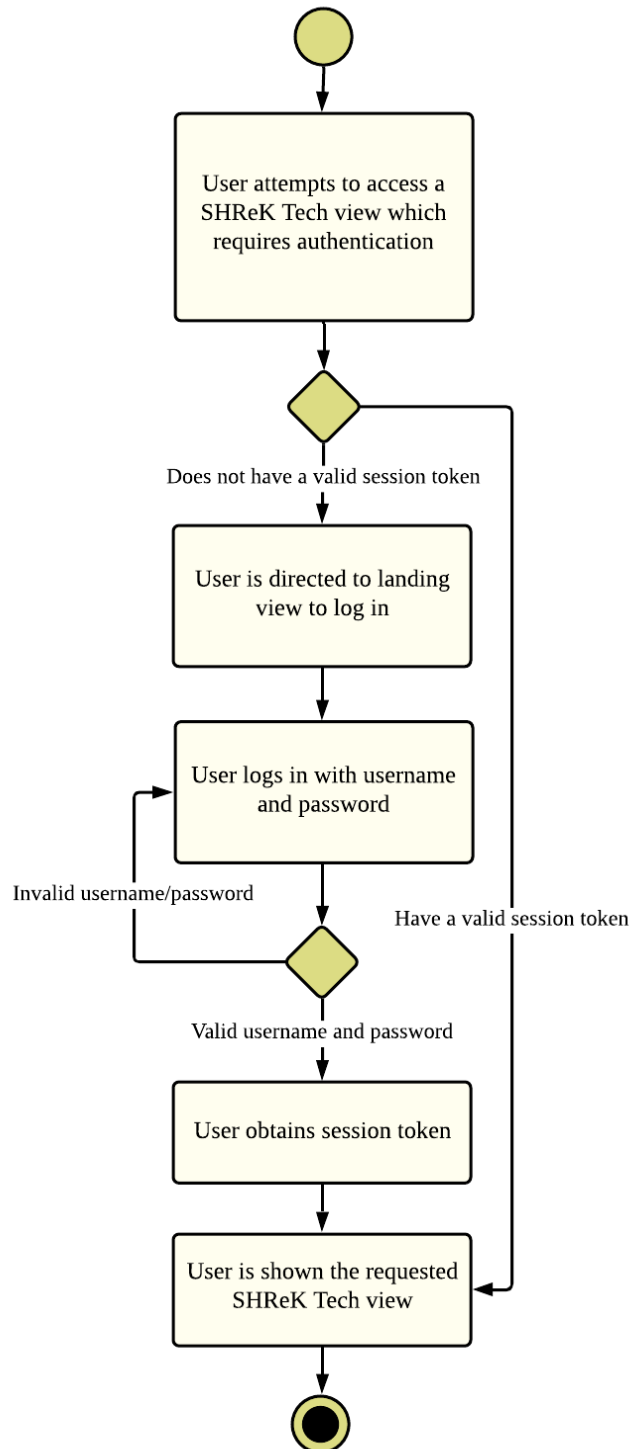


Figure 3: Activity diagram illustrating how login works

In fulfilment of [NF1.2](#), signing session tokens with a secret key prevents an unauthenticated attacker from generating a JWT that can be used to make requests on SHReK Tech or modifying the contents of a valid JWT.

Furthermore, in accordance with [NF1.3](#), if an attacker gains access to a valid session token, a token with an expiry means that the attacker will only have short-lived access to the

compromised account. The attacker will lose access to the compromised account after three hours, mitigating the potential negative consequences.

Blacklisting tokens after a user logs out ensures that the token would not be misused for further authentication.

5.3 Authentication — Sensitive Request Management

5.3.1 Design of Sensitive Request Management

When making sensitive requests such as password change or account deletion, users will have to re-enter their password in order to perform the request, as given in [NF1.4](#).


5.3.2 Rationale for Design of Sensitive Request Management

The session token is no longer sufficient when it comes to authenticating sensitive requests due to the possibility of Time-of-Check-Time-of-Use (TOCTOU) attacks. It is necessary to ensure that the user making this sensitive request is the same user that is authenticated in the system. Re-authentication may inconvenience users but we have chosen to prioritise security over usability, even more so for sensitive requests, based on [Section 3.2](#).

5.4 Waiting User Status

5.4.1 Design of Waiting User Status

Before selecting a difficulty level for a technical interview in the Home View ([Section 7.2](#)), users are able to view which difficulty levels already have a waiting user. This feature is an implementation of the Pub-Sub pattern using Socket.IO where all users subscribe to the waiting-users channel upon loading the Home view (see Figure 4).



```
controllers > JS socketController.js > handleJoinWaitingUsersEvent
function handleJoinWaitingUsersEvent (socket) {
  socket.join('waiting-users')
  socket.emit('update-waiting-users', waitingUsers)
}
```

Figure 4: Code snippet illustrating subscribing to a channel

Subsequently, whenever a waiting user is added or removed, an update-waiting-users event is published to the waiting-users channel (see Figure 5). All subscribed users will

receive this event and the attached data, and the client-side event handler of each subscribed user then uses this data to update the Home view UI.

```
controllers > JS socketController.js > ...  
  
function addWaitingUser (matchBy, socket) {  
  userMatchingPreferences.set(socket.id, matchBy)  
  waitingUsers[matchBy] = socket.id  
  socket.to('waiting-users').emit('update-waiting-users', waitingUsers)  
}  
  
function removeWaitingUser (matchBy, waitingUserId, io) {  
  waitingUsers[matchBy] = null  
  io.to('waiting-users').emit('update-waiting-users', waitingUsers)  
  userMatchingPreferences.delete(waitingUserId)  
}
```

Figure 5: Code snippet illustrating publishing to a channel

5.4.2 Rationale for Design of Waiting User Status

The Pub-Sub pattern is used as there can be multiple users interested in the same information about the waiting users. With the Pub-Sub pattern, the same update-waiting-users event can easily be broadcasted to all interested users who have subscribed to the waiting-users channel.

Additionally, emitting events over Socket.IO connections is chosen over the typical HTTP requests here as there are ongoing updates to the waiting users and the client cannot anticipate when these changes will take place. To receive these updates with HTTP requests, the client would have to make multiple regular polling calls to check for changes, which is rather inefficient. In contrast, Socket.IO provides bidirectional communication which allows the server to push changes in the waiting users to the clients as and when they happen.

This UI design also allows users to find matches more efficiently, increasing the usability of SHReK Tech ([NF3.4](#)).

5.5 Real-Time Communication Components

5.5.1 Design of Real-Time Communication Components

Similar to the design of the waiting user status feature ([Section 5.4](#)), real-time communication of two matched users in a technical interview through the code editor and chatbox are implemented by emitting events over Socket.IO connections. On the server side, a Socket.IO Server instance is created, whereas on the client side, each user is assigned a client socket

when they start a technical interview. Upon matching two users, they subscribe to a common channel and in the case of Socket.IO, the sockets join the same room.

In the coding room, users send and receive code updates and chat through their client sockets. Instead of client sockets directly communicating with one another, client sockets are unaware of one another and use the Server instance as the Mediator. This communication between client sockets follows the Mediator pattern, whereby client sockets delegate the routing of data and messages that they exchange with other client sockets to the Mediator.

For example, with reference to Figure 6, when user A updates the code or sends a chat, an update-code or send-chat event is published to the room within the Server instance. Since the server-side socket of user B has joined the same room, a new-code or new-chat event is emitted to client socket B from the Server. The respective event handlers on the client side are then triggered to update the view of the code editor or chatbox accordingly.

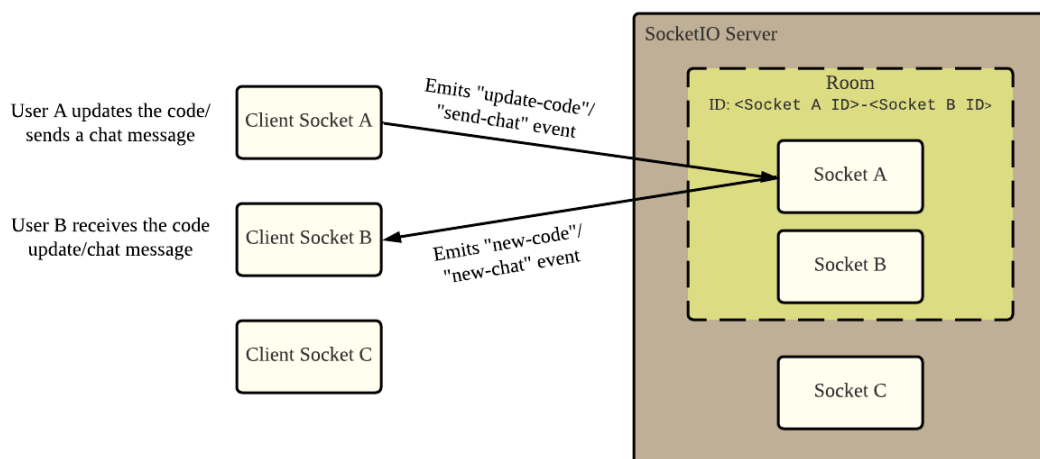


Figure 6: Diagram illustrating update-code and send-chat events

5.5.2 Rationale for Design of Real-Time Communication Components

Real-time communication components such as the code editor and chatbox are implemented using Socket.IO instead of HTTP requests. Since an event is emitted for every code change and chat sent, a high frequency of event emittance with small payloads is expected. As such, the overhead incurred by HTTP requests and responses, which are sent with headers, will be significant. On the other hand, sockets provide persistent connections for users to exchange data and messages without incurring additional overheads when transporting. This greatly reduces the communication delay and improves performance, which is especially crucial for

real-time collaboration. Thus, we chose to use Socket.IO over HTTP requests, in alignment with [NF2.2](#) and [NF2.3](#).

In addition, the Mediator pattern allows for loose coupling between the clients by having the Server instance handle the interactions between clients.

5.6 Code Editor

5.6.1 Design of Code Editor

Building upon [Section 5.5](#), any updated code is not directly sent as text data with the emitted update-code event. Rather, we incorporated Automerge⁷, a library for Conflict-Free Replicated Data Types (CRDTs). In the technical interview, each user maintains an Automerge state which should be equal. If, for example, user A makes an edit to the code as seen in Figure 7, Automerge gets the changes between the states before and after the edit. These changes are then passed to user B over the Socket.IO connections, following the process described in [Section 5.5](#). Automerge then applies the changes to user B's Automerge state such that the resultant state is equal to that of user A's.

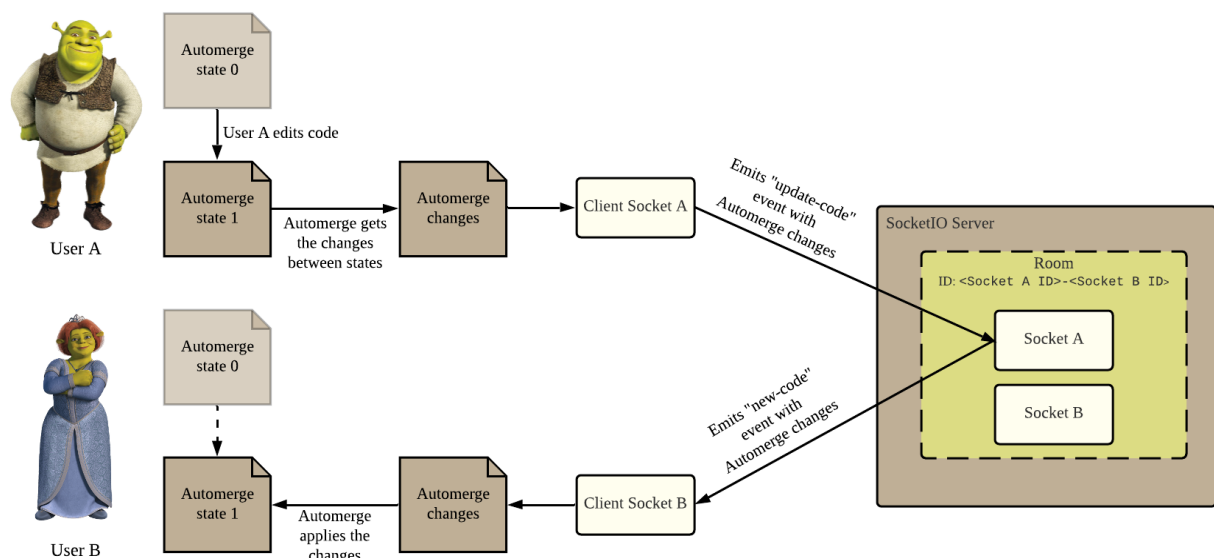


Figure 7: Diagram illustrating how CRDT works

5.6.2 Rationale for Design of Code Editor

To provide a truly collaborative experience during the technical interview, both matched users should be allowed to concurrently edit the code. The resultant code should also be consistent

⁷ <https://github.com/automerge/automerge>

for both users. Automerge is a tool that manages merge conflicts and ensures that two documents are equal after applying the same set of changes, even if the changes were applied in a different order. Therefore, we utilised Automerge, allowing us to fulfil [F6.2](#) and [F6.3](#).

5.7 Coding Room

5.7.1 Design of Coding Room

As mentioned in [Section 5.5](#), a user is able to send and receive code updates and chats in a coding room if the user's socket is subscribed to the coding room's channel, known as the coding room ID in this project.

If two users are matched for the technical interview, the coding room ID is a concatenation of the socket IDs of both users. If a user is unable to find a match for the technical interview, the coding room ID is simply the user's socket ID.

5.7.2 Rationale for Design of Coding Room

Random and unique Socket.IO IDs are dynamically assigned to each socket upon a new connection. Since SHReK Tech establishes a new Socket.IO connection for each user for each technical interview, it would be extremely difficult for an unauthorised third party to guess and subscribe to a coding room's channel. Therefore, this ensures that only the authorised users are able to access the code updates and chats of their coding room, which is in accordance with [NF1.6](#).

5.8 Coding Questions

5.8.1 Design of Coding Questions

The coding questions that SHReK Tech uses were scraped from Leetcode using a Python scraper. Apart from the question title and content, we also stored the difficulty level for each coding question.

Upon two users being matched by a common difficulty level, we randomly generate two coding questions of that difficulty level for both parts of the interview. From the data source of all coding questions, we first filter by the given difficulty level, then use the random index generated by the Coding Question Controller to obtain the object ID of the coding question at

that index, as illustrated in Figure 8. These question IDs are only generated once for each technical interview then passed on to the Socket Controller to communicate to the two users.

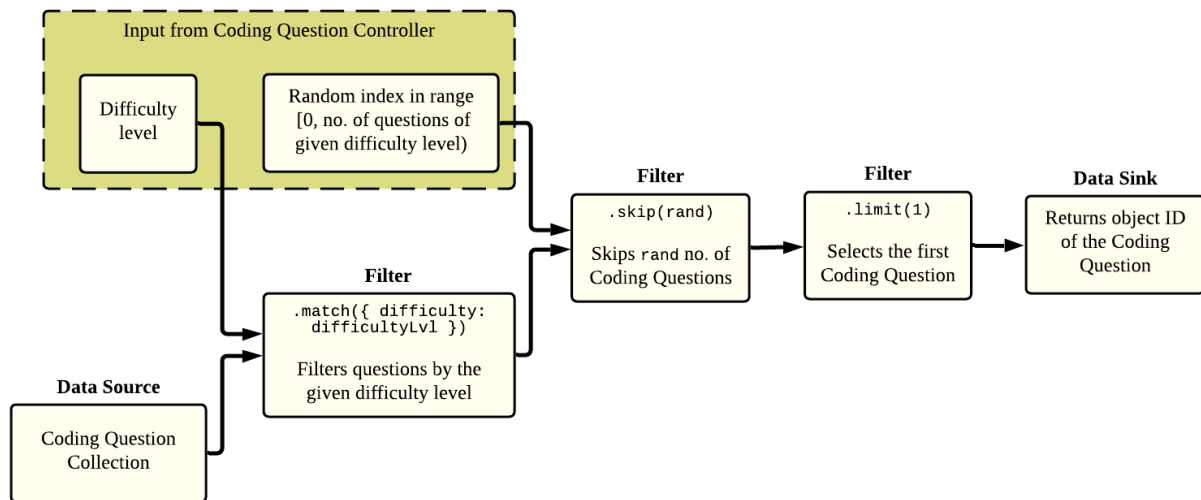


Figure 8: Pipe and Filter diagram illustrating how Coding Questions are rendered

5.8.2 Rationale for Design of Coding Questions

With this implementation, we can make sure that both users in the same Coding Room will see the same questions that are of their preferred difficulty level ([F4.1](#)). Furthermore, since the object ID is a unique identifier, the two coding questions generated are not the same question ([F4.3](#)).

6 DevOps

6.1 Agile Software Development Process

Our group followed the Agile Software Development Process (see Figure 9), where we had one-week sprints. Before each sprint, we conducted sprint planning which includes evaluating the items in our product backlog, selecting items for our sprint backlog based on priority and assigning each selected item to a member. Although we did not have daily stand-ups, we notified one another of our progress and any blockers we were facing through our Telegram group.



Figure 9: Sprint process

6.2 Continuous Integration/Continuous Deployment

6.2.1 Design and Rationale for CI/CD Pipeline

While a typical CI/CD pipeline first deploys to a staging environment before acceptance testing and deploying to production, our pipeline directly deploys to production after passing unit tests, as illustrated in Figure 10. This eliminates the extra steps of needing to set up a staging environment, and potentially having to run acceptance tests in both the staging and production environments. Given the short timeframe of this project and the fact that there are no real users actively using our application, we made this decision to speed up our development process.

We acknowledge that our current CI/CD pipeline will likely introduce bugs not caught by our unit tests into production. Hence, once our application starts serving real, active users, we should follow the typical CI/CD pipeline to minimise the possibility of such bugs, which would impact our users.

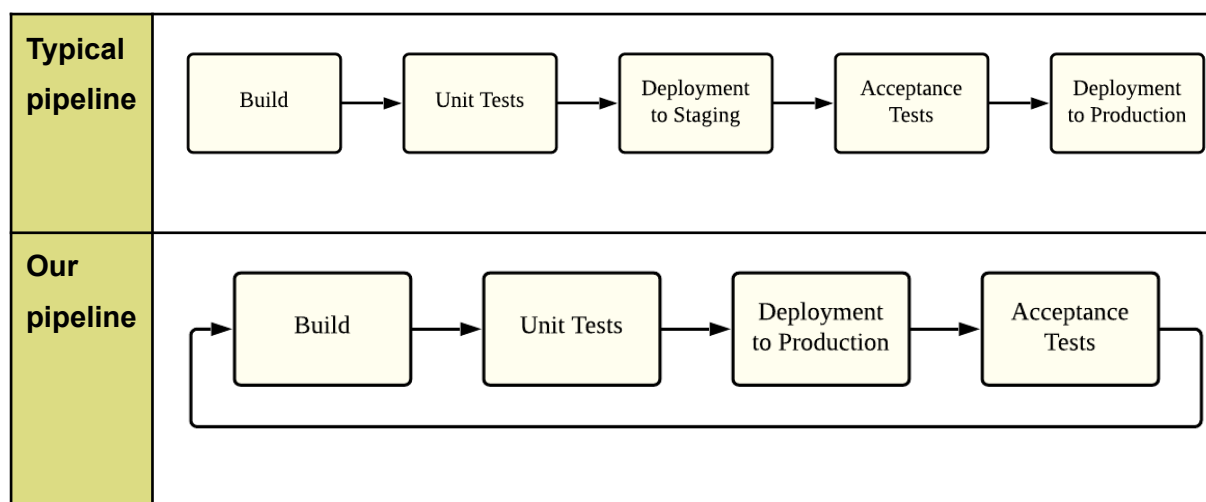


Figure 10: Typical CI/CD pipeline as compared to our CI/CD pipeline

6.2.2 Implementation of CI/CD Pipeline

Our project utilises GitHub Actions for CI/CD as illustrated in Figure 11. Upon a Pull Request made to the main branch of the SHReK Tech repository, the CI workflow is triggered to lint the code based on the JavaScript Standard Style⁸, as well as to test the code. Once the checks pass and the PR is approved by at least one reviewer, the PR can be merged, which triggers the CD workflow. This workflow builds the static files for frontend, which are copied to the server directory, and deploys the server to <https://shrektech.herokuapp.com/> as a Heroku application.

⁸ <https://standardjs.com/>

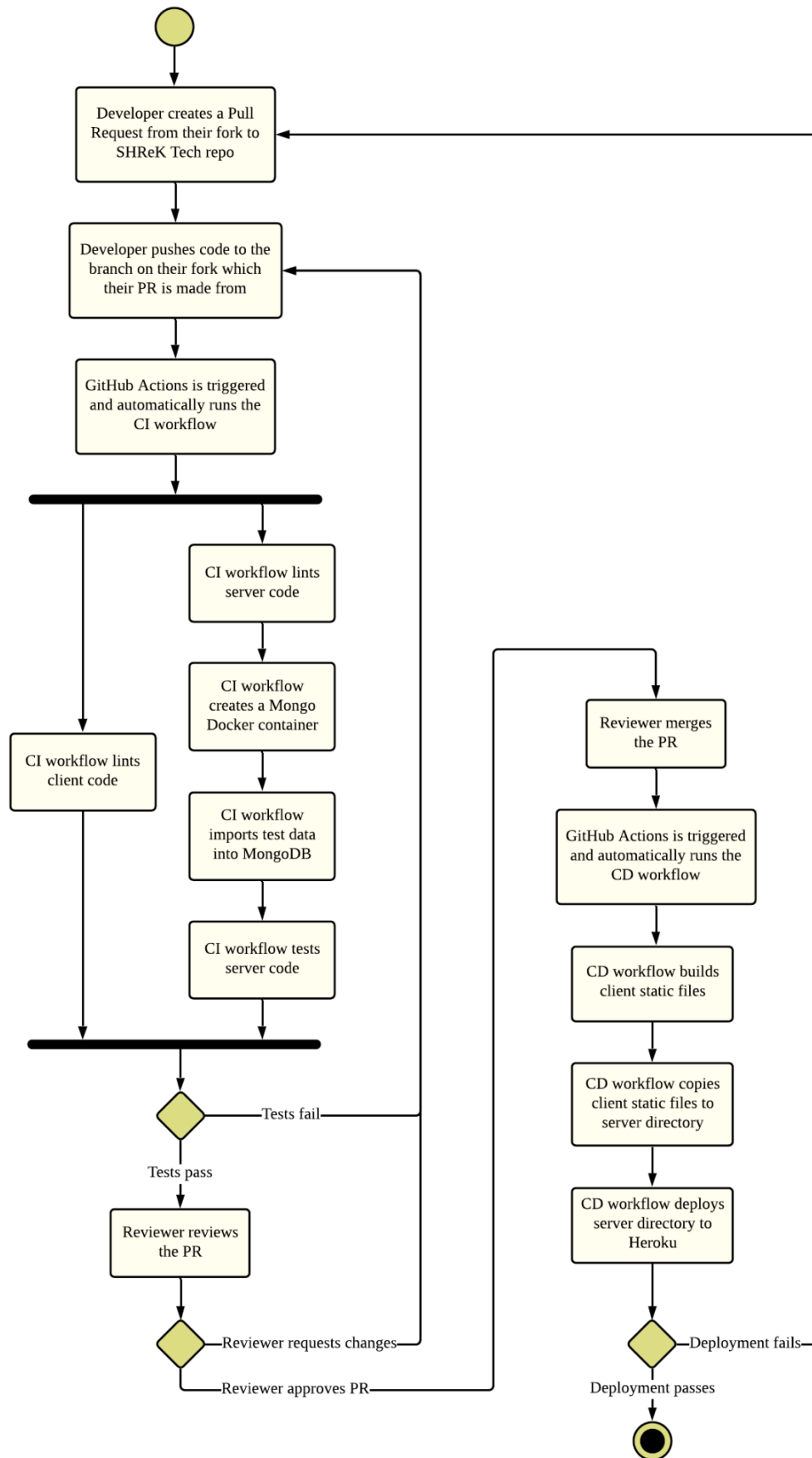
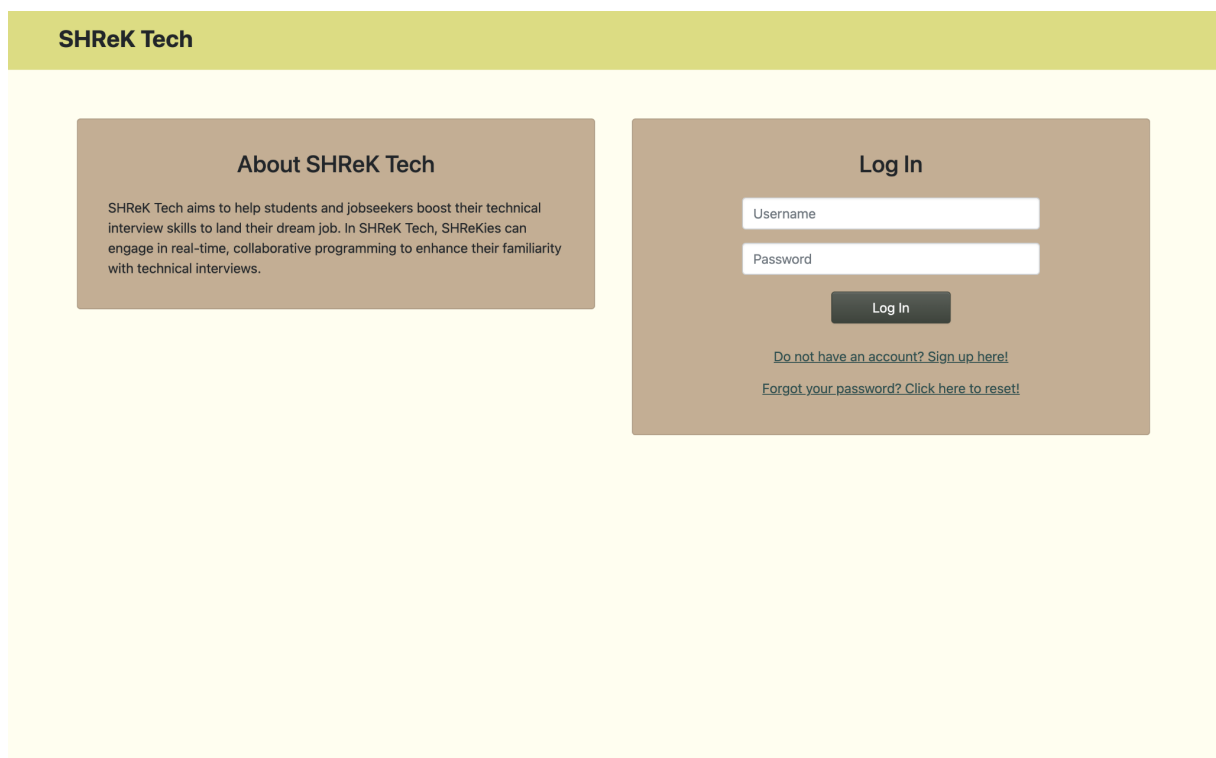


Figure 11: CI/CD workflow

7 User Interface of SHReK Tech

7.1 Landing View

This is the view that users first see upon entering SHReK Tech at <https://shrektech.herokuapp.com/>. Users are able to log in if they have existing accounts (see Figure 12), sign up for new accounts (see Figure 13) or reset their passwords if they have forgotten them (see Figure 14).



The screenshot displays the SHReK Tech landing page. At the top, there is a green header bar with the text "SHReK Tech". Below this, the page is divided into two main sections. On the left, there is a brown box titled "About SHReK Tech" containing the text: "SHReK Tech aims to help students and jobseekers boost their technical interview skills to land their dream job. In SHReK Tech, SHReKies can engage in real-time, collaborative programming to enhance their familiarity with technical interviews." On the right, there is a larger brown box titled "Log In". This box contains a "Username" input field, a "Password" input field, and a "Log In" button. Below the input fields, there are two links: "Do not have an account? Sign up here!" and "Forgot your password? Click here to reset!".

Figure 12: Landing View - Log in to existing accounts

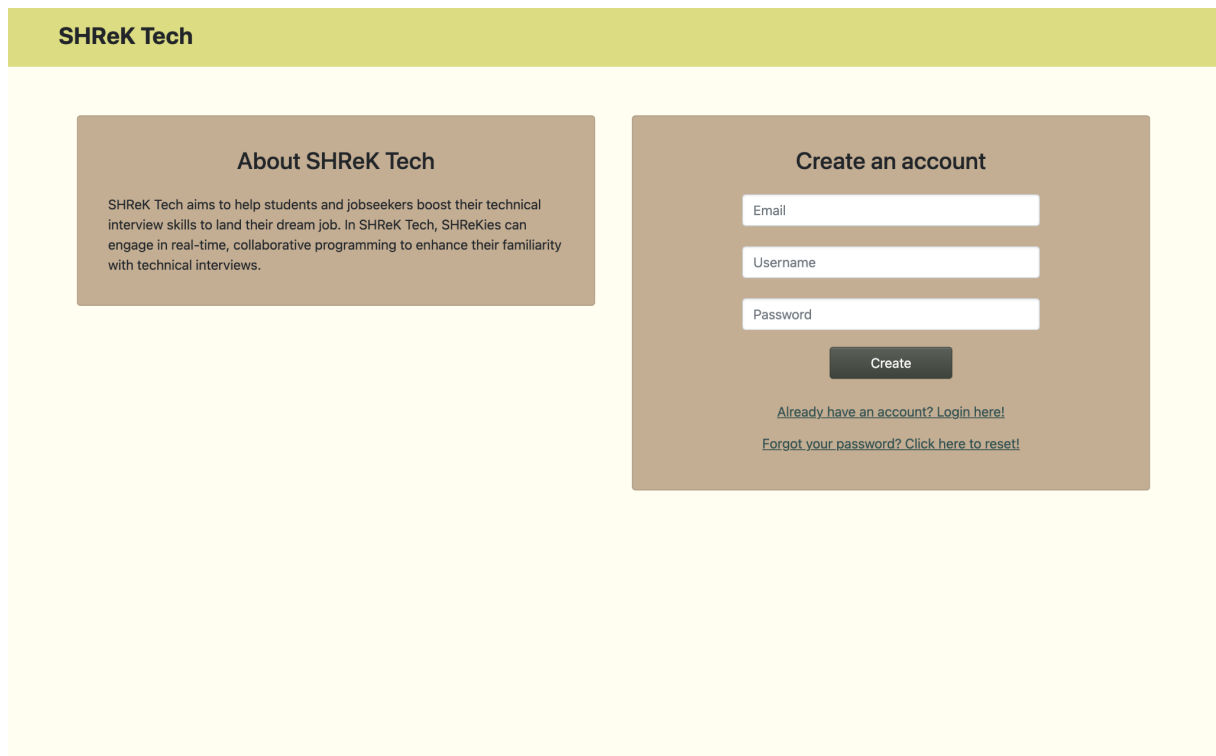


Figure 13: Landing View - Sign up for new accounts

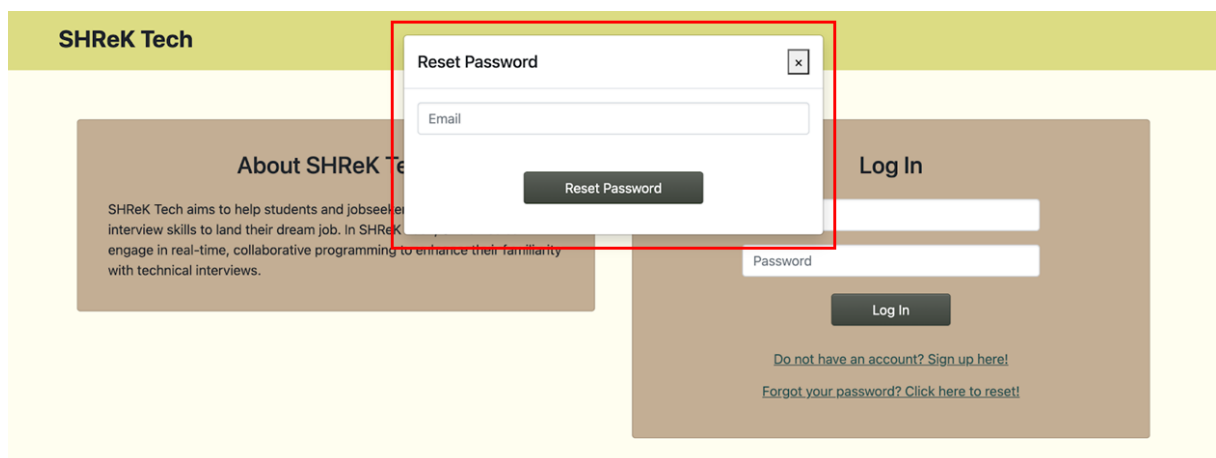


Figure 14: Landing View - Reset password using verified email address

7.2 Home View

Once users are logged in successfully, they are brought to the Home View. Users are able to select the difficulty levels which they want to attempt. Furthermore, to allow users to find matches more efficiently ([NF3.4](#)), the human icon will light up in green when there is a user waiting for a match for that difficulty level (see Figure 15). After selecting a difficulty level, users can click 'Find a Match' and will be brought to the Matching View.

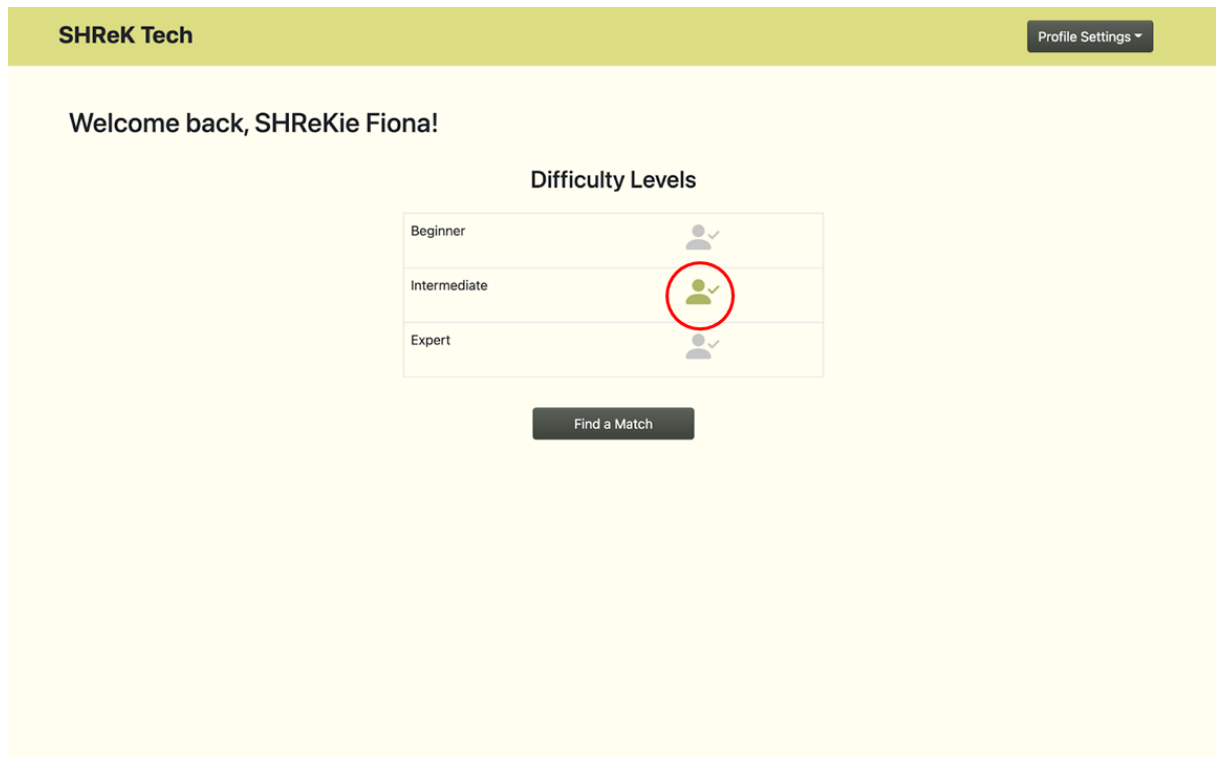


Figure 15: Home View - Green icon when there are users waiting to be matched

In addition to finding a match, users are also able to change their passwords, delete their accounts and log out (see Figure 16) in Home View.

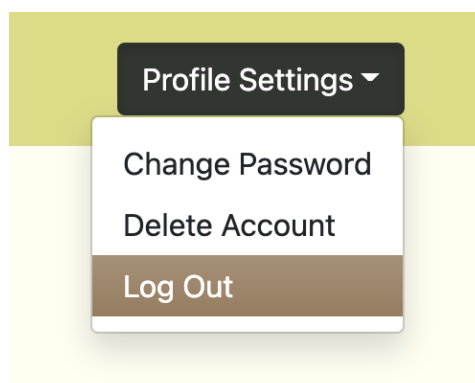


Figure 16: Home View - Profile Settings

7.3 Matching View

After selecting 'Find a Match', users are brought to the Matching View where a timer runs for 30 seconds while the application tries to find a match for the selected difficulty (see Figure 17).

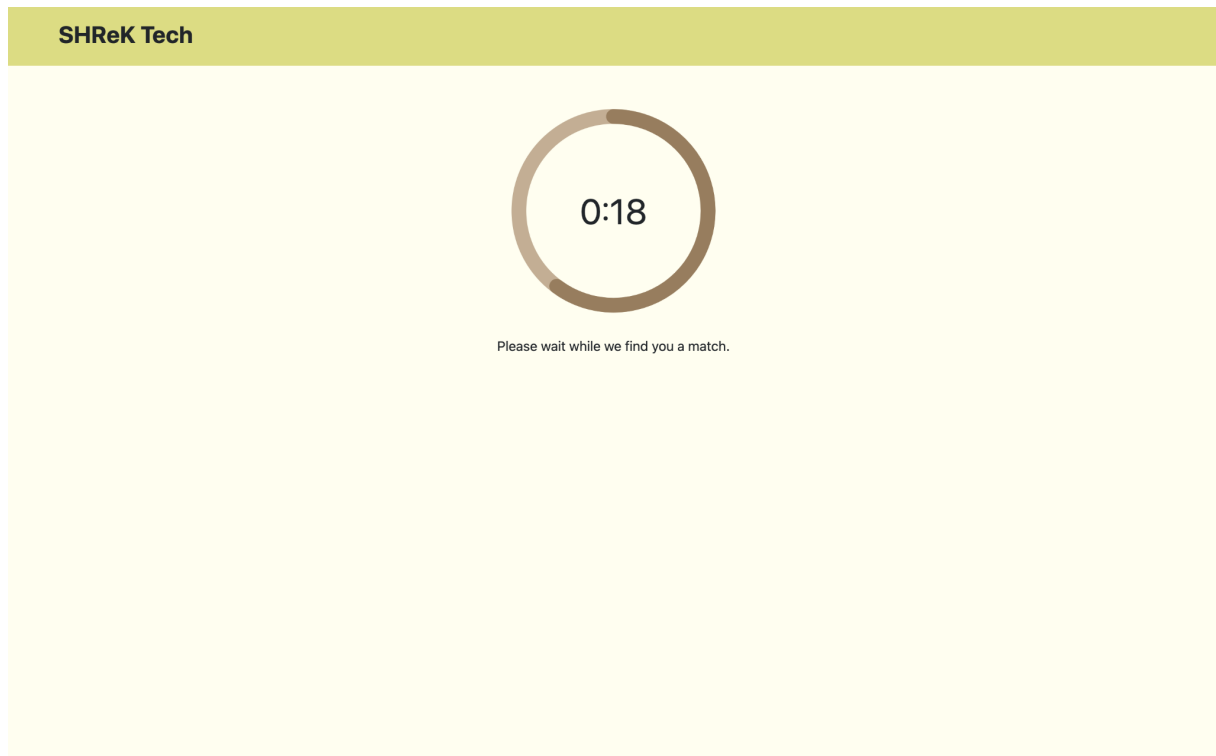


Figure 17: Matching View - Countdown timer

In the case whereby a match cannot be found after 30 seconds, users are able to

1. Wait for another 30 seconds
2. Select another difficulty level
3. Proceed without a match

See Figure 18 for screenshot.

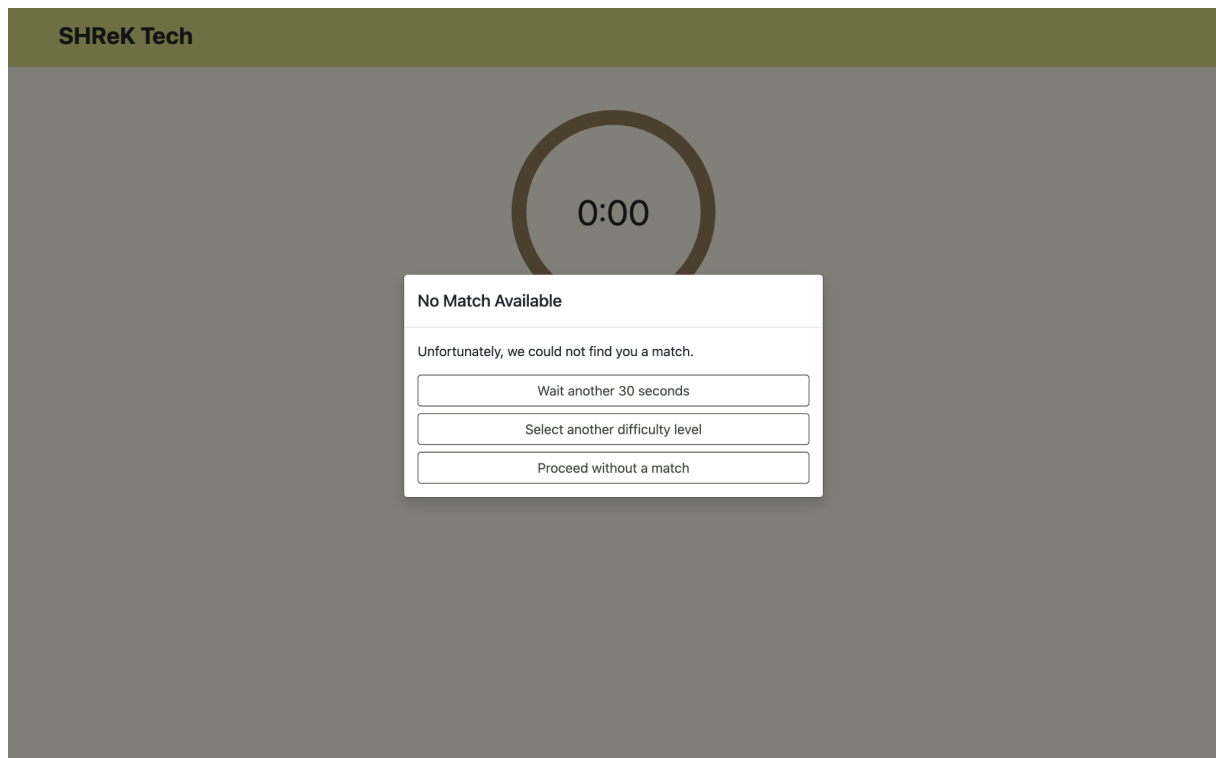


Figure 18: Matching View - 3 options to choose from if no match is found

7.4 Coding Room View

After successful matching or if users choose to proceed without a match, they are brought to the Coding Room View. Users are able to see the assigned coding question of the selected difficulty, a collaborative code editor and a chatbox for communication (see Figure 19).

SHReK Tech

Coding Question

Next Question

Code Editor

00:02:14
Recommended: 00:45:00

End Session

Peeking Iterator

Design an iterator that supports the peek operation on an existing iterator in addition to the hasNext and the next operations.
Implement the PeekingIterator class:

PeekingIterator(iterator<int> nums) Initializes the object with the given integer iterator iterator.
int next() Returns the next element in the array and moves the pointer to the next element.
boolean hasNext() Returns true if there are still elements in the array.
int peek() Returns the next element in the array without moving the pointer.

Note: Each language may have a different implementation of the constructor and Iterator, but they all support the int next() and boolean hasNext() functions.

Example 1:
Input
["PeekingIterator", "next", "peek", "next", "next", "hasNext"]
[[["1, 2, 3"], [], [], [], [], []]]
Output
[null, 1, 2, 2, 3, false]

Explanation
PeekingIterator peekingIterator = new

```
public static main(String[] args) {  
    // Type code here  
  
}
```

Chat

SHReK Tech Bot 12:20

Fiona joined this room

SHReK Tech Bot 12:20

Please note that you cannot return to this session after leaving or refreshing the page.

SHReK Tech Bot 12:20

Your role for this coding question is INTERVIEWEE

SHReK Tech Bot 12:20 PM

Shrek joined this room

Fiona Interviewee 12:20

Hi Shrek <3

Shrek Interviewer 12:20 PM

Hi Fiona my love~

Shrek Interviewer 12:22 PM

What is a naïve solution?

Chat here...

Send

Figure 19: Coding Room View

Users who assume the Interviewer role would be able to view sample interview questions to ask (see Figure 20). This is to guide users assigned the 'Interviewer' role who may be unsure of what interview questions they may ask their 'Interviewee'. This increases the usability of SHReK Tech for users who are new to technical interviews ([NF3.1](#)).

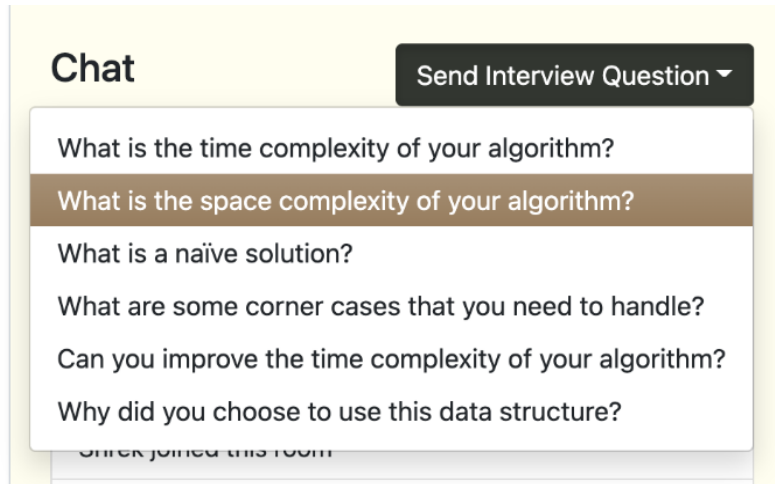


Figure 20: Coding Room View - Sample interview questions

Matched users are able to proceed to the next question once they have completed the first coding question. To allow users who have limited experience in technical interviews, we included a tooltip to provide information on the 'Next Question' button ([NF3.1](#)) (see Figure 21).

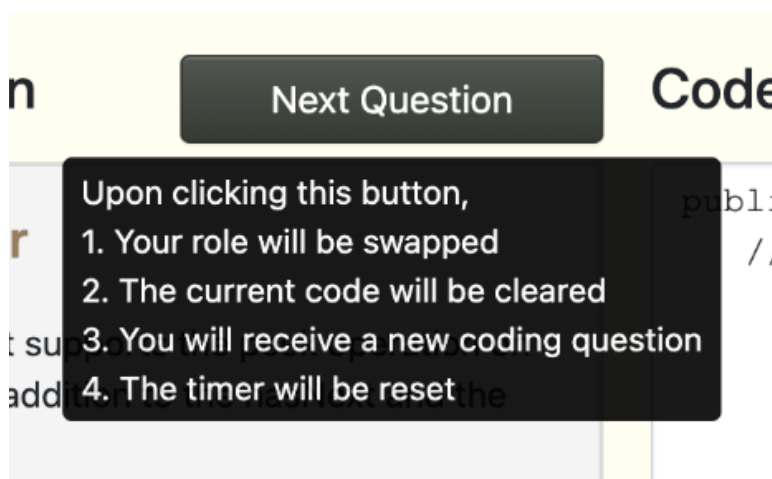


Figure 21: Coding Room View - Tooltips

8 Improvements and Enhancements

Due to the tight time frame of this project, we are unable to implement more features. However, our group has considered various improvements which would further enhance SHReK Tech.

8.1 Find a Match By Topic

In addition to difficulty levels, users can choose to start a technical interview based on the different topics of coding questions. This enhancement is more targeted at users who wish to practise topics which they are weaker in.

8.2 Join a Technical Interview via an Invite Link

Users can send their friends a unique invite link to invite them to a technical interview. Each user can assume either an interviewer or interviewee role. This feature might come in handy for users who are more comfortable practising with their friends, allowing them to have a more fruitful technical interview, as compared to if they were matched with a stranger.

8.3 In-Built Compiler

To better replicate a typical technical interview, a possible enhancement would be to replace the existing code editor, which is currently a simple textbox, into one with a compiler. This would allow the interviewee to run their code within SHReK Tech and verify their output against the suggested. One possible way is to make use of JDoodle⁹.

8.4 Voice Chat/Video Conferencing

As most typical technical interviews might have voice chat or video conferencing in place, by incorporating these two features would better simulate the setting. In addition, it will be more convenient for users to communicate with each other rather than typing in the chatbox.

⁹ <https://www.jdoodle.com/>

9 Reflections and Learning Points

Through this project, our group has learnt how to apply the content and principles taught during lectures into practice. Despite some smooth-sailing times, more often than not, we were faced with a lot of unknown bugs and problems. As our group is considered to be less experienced in web application developments, we had to learn things on the go. While there were more challenges we faced in this project, we would like to highlight the more distinct ones and share what we have learnt.

Firstly, most of the tools we used in our application were new to us, for example MongoDB, Express and Node, Socket.IO, Heroku Deployment, and Vue. Fortunately, we had the chance to learn some of them through the OTOT Assignments and hence it was less daunting when we started on our project.

Secondly, deployment to production with sockets was another challenge we faced. Initially we tried deploying to Google App Engine but the sockets could not work as intended. Afterwards, we tried deploying the server to Heroku and frontend to Netlify, but again, the sockets could not work. After multiple times of trying, we managed to deploy to Heroku successfully with the frontend served as static files.

This brings us to our next challenge which is Continuous Deployment. As we had tried to manually deploy from our local machine before implementing CD, Heroku recognised that multiple repositories are trying to push to the same code base. We had to trial and error various methods in order to push the latest version to Heroku.

Thirdly, unit testing was another challenge we faced. GitHub secrets are not passed to workflows that are triggered by a pull request from a fork. This also means that we are unable to access the secrets during the integration testing in GitHub Actions. Hence, we did not know where to store the secret key and test tokens for testing without compromising on the security of the application. In the end, we decided to come up with a secret key for the sole purpose of testing. This allows us to explicitly copy the key and tokens into an environment file during builds and use them for testing. Since the secret key used to generate the token is different from that used in deployment, the security of the application is not compromised.

Although there were many difficulties faced, our group felt that this project was a very good opportunity for us to learn more about the principles and patterns of software engineering. We had the chance to experience the whole software development process, from gathering user requirements to implementation and deployment. In addition, we had the chance to

experience a mini version of an Agile Software Development Process where we have sprints, which we believe would prepare us for future internships and jobs as Software Engineers.

Lastly, our group would like to express our sincerest gratitude to Professor Bimlesh, our mentor Shao Yi, as well as the rest of the teaching team, for the great learning opportunities and guidance that we have been provided with throughout this module.

10 Appendix

Appendix A. CI Test Cases

Test Case 1. Auth

- 1.1 Route POST /api/users/signup
 - 1.1.1 Should POST a new user
 - 1.1.2 Should not POST user if there a missing fields
 - 1.1.3 Should not POST user if email format is invalid
 - 1.1.4 Should not POST a user if username already exists
 - 1.1.5 Should not POST a user if email already exists
- 1.2 Route /api/users
 - 1.2.1 Should login if credentials are right
 - 1.2.2 Should not login if credentials are wrong
 - 1.2.3 Should not login if not verified
- 1.3 Route /api/users/reset
 - 1.3.1 Should generate reset email
 - 1.3.2 Should not generate reset email if email format is invalid
 - 1.3.3 Should not generate reset email if email does not exists
- 1.4 Route /api/users/
 - 1.4.1 Should PUT new password
 - 1.4.2 Should not PUT if credentials are wrong
 - 1.4.3 Should not DELETE user if credentials are wrong
 - 1.4.4 Should DELETE user
- 1.5 Route /users/verify/checkAuth
 - 1.5.1 Should authenticate user if session token is valid
 - 1.5.2 Should blacklist the token if user logs out
 - 1.5.3 Should not authenticate user if session token is blacklisted
 - 1.5.4 Should not authenticate user if session token is invalid
 - 1.5.5 Should not authenticate user if session token have expired

Test Case 2. Coding Questions

- 2.1 Route GET /api/coding-questions
 - 2.1.1 Should GET the coding questions if JWT is valid
 - 2.1.2 Should not GET coding questions if JWT is not provided

- 2.1.3 Should GET a coding questions of easy difficulty
- 2.1.4 Should GET a coding questions of intermediate difficulty
- 2.1.5 Should GET a coding questions of expert difficulty
- 2.1.6 Should GET 2 different coding questions of easy difficulty
- 2.1.7 Should GET 2 different coding questions of intermediate difficulty
- 2.1.8 Should GET 2 different coding questions of expert difficulty

Test Case 3. Interview Questions

- 3.1 Route GET /api/interview-questions
 - 3.1.1 Should GET all interview questions if JWT is valid
 - 3.1.2 Should not GET interview questions if JWT is not provided
 - 3.1.3 Should not GET interview questions if JWT is invalid

Test Case 4. Sockets

- 4.1 Event update-waiting-users
 - 4.1.1 Should send an empty object if there are no waiting users
 - 4.1.2 Should send an object with waiting users
- 4.2 Event match-found
 - 4.2.1 Should send the waiting user info about the coding room
 - 4.2.2 Should remove the waiting user from the waiting room
- 4.3 Event coding-room-ready
 - 4.3.1 Should send both matched users info about the coding room
- 4.4 Event coding-room-ready-solo
 - 4.4.1 Should send the user the coding question id
- 4.5 Event typing
 - 4.5.1 Should send the username of the typing user
- 4.6 Event stop-typing
 - 4.6.1 Should be emitted
- 4.7 Event new-chat
 - 4.7.1 Should send a non-private chat to both matched users
 - 4.7.2 Should send a private chat to only one user
 - 4.7.3 Should send the remaining user a chat if a user disconnects
- 4.8 Event new-code
 - 4.8.1 Should send the code changes to the other user
- 4.9 Event next-question
 - 4.9.1 Should be emitted to both matched users

Test Case 5. Sessions

- 5.1 Route GET /api/users/:username/session
 - 5.1.1 Should GET user and return true
 - 5.1.2 Should GET user and return false
 - 5.1.3 Should not GET user if username does not exist