Project Title: System Verification and Validation Plan for SFWRENG 4G06

Team #12, Team 12.0
Kanugalawattage, Anton
Subedi, Dipendra
Rizkalla, Youssef
Leung, Tamas
Zhao, Zhiming

April 4, 2023

1 Revision History

Date	Version	Notes
April 3, 2023	1.1	Updated test cases, removed unnecessary test cases. Added new test cases for new requirements. Added usability section and questions. Added Unit testing section. Added Tracability matrix for functional requirements.
November 1, 2022	1.0	Added initial version.

Contents

List of Tables

2 Symbols, Abbreviations and Acronyms

Symbol	Description
Data Structures and Algorithms	A topic of study for Computer Scientists.
CodeChamp	The system being built and tested.
Angular	A web framework for building web applications.
JavaScript	A programming language that can be executed by browsers.
TypeScript	A syntactical superset of JavaScript.
WebSocket	A communications protocol used for two-way interaction.
Client	A device used to connect to a CodeChamp instance.
DSA	Abbreviation for Data Structures and Algorithms.
CI	Abbreviation for Continuous Integration.
SRS	Abbreviation for Software Requirements Specification.
MIS	Abbreviation for Module Interface Specification.
MG	Abbreviation for Module Guide.
TestCafe	An end-to-end testing framework for web applications.
VSCode	A light weight code editor.
Mocha	A JavaScript testing framework.
Jasmine	A JavaScript testing framework.
Git	A version control system for software.
Husky	A tool used to setup Git hooks.

Table 1: Symbols, Abbreviations and Acronyms

3 General Information

3.1 Summary

This document's purpose is to show the detailed testing plan for the game CodeChamp. This document contains sections regarding information on the test plans, system test descriptions (functional and non-function requirements) and the unit testing plans. It explains how tests will be performed with details about initial state, input and output. This document will include descriptions of all the testing, validating and verification procedures.

3.2 Objectives

The objective of the test plan is to test the functionality of CodeChamp. The final goal of the test plan is to show that all the functional requirements and the non functional requirements from the software requirements specification are met. This way, we can demonstrate adequate usability and build confidence in the correctness of our software.

3.3 Relevant Documentation

1. Development Plan
2. System Requirements Specification
3. Hazard Analysis
4. Module Interface Specification
5. Module Guide

4 Plan

This section outlines the verification and validation team for CodeChamp alongside their roles. Additionally, it describes how different components will be tested, including the SRS, design and implementation. Finally, it specifies the testing and verification tools that will be used to accomplish that.

4.1 Verification and Validation Team

Member	Role
Anton Kanugalawattage	Automatic verification of front-end code; Manual code review
Dipendra Subedi	End-to-End testing; Manual SRS verification
Youssef Rizkalla	Automatic verification of back-end code; Manual code review
Tamas Leung	Performance testing; End-to-End testing
Zhiming Zhao	Integration of CI actions; Manual SRS verification
Spencer Smith	Manual review of documents and system
Chris Schankula	Manual review of documents and system
Classmate Review Groups	Manual review of documents and system

Table 2: Verification and Validation Team Members and Roles

All team members will participate in testing features all around the tech stack. The roles are intended for each member to have a focus area, which can later change depending on the team's needs. If a team member finishes early, they can help other team members with another part of the testing. Likewise, if a testing area is more labour-intensive than initially expected, other team members can spend additional time helping with that testing area.

4.2 SRS Verification Plan

The SRS checklist will be utilized to ensure that all requirements are verifiable. In turn, system tests will be derived to ensure that all functional and non-functional requirements are verified. Automated tests will be done when possible, using the end-to-end testing framework TestCafe. This will simulate the experience of a real user interacting with the system and will allow developers to easily identify regressions. Parts of the system that may be difficult to test this way will be tested and/or reviewed manually by the project developers as well as members of the testing team.

4.3 Design Verification Plan

The MG and MIS checklists will ensure that the modules specified in the MIS and MG are designed with high quality. Additionally, the designs individually architected by each engineer will be reviewed by the other engineers on the team. Finally, the design will be peer-reviewed by engineers belonging to other capstone groups, as well as the instructors. Feedback from peers and instructors will be considered in the final iteration of the design.

4.4 Implementation Verification Plan

Outlined system tests will be used to verify that the implementation of the system meets the requirements. Unit tests will be expected alongside each code change to the main branch of the repository. Each pull request will be reviewed by at least 2 other engineers. During each code walk-through/review the implemented changes will be verified by the reviewers. Additionally, the CI will automatically run tests against each new pull request to verify that it passes all unit tests and to help developers identify regressions in code coverage.

4.5 Automated Testing and Verification Tools

- Programming Language: JavaScript
- Testing Frameworks:
  - Backend (Unit-Tests): Mocha
  - Frontend (Unit-Tests): Jasmine
  - End-to-End: TestCafe
- Code Linting / Formatting / Style:
  - Linter: ESLint
  - Formatter: Prettier
  - The Airbnb style guide will be enforced during code review for all back-end code
  - The official Angular style guide will be enforced during code review for all front-end code

  - Husky pre-commit hooks will be used to automatically apply linters and formatters before pushing to the remote repository

- GitHub Actions CI
  - Auto build and test new pull requests using aforementioned testing frameworks
  - Test new pull requests for linting
  - Test new pull requests for formatting
- Code Coverage
  - All testing frameworks support options for providing code coverage
  - GitHub Actions CI will state code coverage on new pull requests and report regressions
- Performance Testing:
  - Backend: Postman Performance Testing
  - Frontend: Chrome Dev Tools Performance Tester

4.6 Software Validation Plan

An external software validation plan is not planned because the requirements are set by the main developers of the project. An internal software validation will be done by the developers weekly by reviewing the changes that were made and by doing a walk-through of the system. This will allow the stakeholders (developers in the team) to validate that the implementation of the system meets the desired needs.

5 System Test Description

5.1 Tests for Functional Requirements

5.1.1 Match Making Tests

1. TC-MM-1: Join a random match

   Testing Type: Functional, Manual, Dynamic

   Initial State: On Main Menu Page

   Input: Press the "find a match" button

   Output: Screen changes to the lobby page

   Test Case Derivation: The system receives the "find a match" input and returns the user a lobby match id to join, which sends the user to the lobby page.

   How test will be performed: Tester will be used to find and press the "find a match" button, wait until the HTML page is loaded, and then checks the HTML of the page to ensure the lobby page is displayed.

2. TC-MM-2: Join an existing page with a code

   Testing Type: Functional, Manual, Dynamic

   Initial State: On Main Menu Page

   Input: Inputs an existing lobby's code and presses the "join match" button.

   Output: Screen changes to lobby page.

   Test Case Derivation: The system receives the "join match" input with a code and returns the user the lobby match that matches the inputted code, which sends the user to the lobby page.

   How test will be performed: Tester creates a lobby with one machine, receives the code from the screen, then uses the code to join the match. Tester then checks to see if the page changes to the lobby page and if the lobby code matches the inputted lobby code.

3. TC-MM-3: Create a match

   Testing Type: Functional, Manual, Dynamic

   Initial State: On Main Menu Page

   Input: Press the "create match" button.

   Output: Screen changes to lobby page.

Test Case Derivation: The system receives the "create match" input with a code and returns the user the lobby match with a new code, which sends the user to the lobby page.

How test will be performed: Tester will be used to press the "create match" button, wait until the HTML page is loaded, and then checks the HTML of the page to ensure the lobby page is displayed.

5.1.2 In-Game Tests

1. TC-IG-1: Complete a full match

   Testing Type: Functional, Manual, Dynamic

   Initial State: Start of In-game Page

   Input: Enter the correct code and submit the code in NUMBER_OF_ROUNDS rounds.

   Output: After every successful submission code input, a new problem is displayed. After the last round's successful submission, a win-end game page is displayed.

   Test Case Derivation: The system receives the submitted code input with user-written code. The system then validates and allows the user to move to the next round. There are only NUMBER_OF_ROUNDS rounds per game, on the last round, the game ends and gives the user a win.

   How test will be performed: A user will start a game. The user will manually find out the answers for each question and inputs them. The user will submit every round and ensure a new problem is displayed when the next round starts or the win end game screen appears if it was the last round. The user also counts to ensure the number of rounds played is equal to NUMBER_OF_ROUNDS.

2. TC-IG-2: Compiles code

   Testing Type: Functional, Manual, Dynamic

   Initial State: In-game Page

Input: The following code snippet with the JavaScript language option:

```
console.log(‘Hello, World!’);
```

Output: Returns success status and presents the user with the number of passed test cases.

Test Case Derivation: The system receives the submitted code input with user-written code. The system then validates and compiles the code. The system then returns the number of passed test cases.

How test will be performed: Tester will type the input code into a predetermined Hello World problem and input it into the code editor. Tester will choose the JavaScript language option from a dropdown menu. The tester waits until the HTML page is loaded and checks the HTML to ensure that all test cases were passed.

5.1.3 Problem Maintenance Tests

1. TC-PM-1: Add a new problem

   Testing Type: Functional, Manual, Dynamic

   Initial State: problem maintenance page

   Input: Input a new problem with the following data and click "Add Problem"

   ```
   {
       name: "Hello World Test Problem",
       description: "Data"
   }
   ```

   Output: List of problems on the problem maintenance page shows the new problem with the new name and description

   Test Case Derivation: The system receives the new problem and updates the database. The system then updates the display to show the new problem.

   How test will be performed: Tester will type the test problem data and click "Add Problem". Tester will check the list of problems displayed including the new problem.

2. TC-PM-2: Modify a problem

   Testing Type: Functional, Manual, Dynamic

   Initial State: problem maintenance page

Input: Clicks “Edit” button onto a problem and edits the problem name to “Test Problem Name”.

Output: A list of problems on the problem maintenance page shows the problem with the name “Test Problem Name”.

Test Case Derivation: The system receives the new problem name and updates the database. The system then updates the display to show the updated problem name.

How test will be performed: Tester will click edit on a test problem and edit the problem name. Tester will check the list of problems displayed including the new problem name.

3. TC-PM-3: Delete a problem

   Testing Type: Functional, Tester, Dynamic

   Initial State: problem maintenance page

   Input: Clicks “Delete” button onto a problem

   Output: The list of problems on the problem maintenance page does not show the problem

   Test Case Derivation: The system receives the delete problem and updates the database. The system then updates the display to show the problem is gone from the list.

   How test will be performed: Tester will click delete on a test problem. Tester will check the list of problems displayed does not include the deleted test problem.

5.1.4 Profile View Tests

1. TC-PV-1: Profile View Displays Win Percentage

   Testing Type: Functional, Manual, Dynamic

   Initial State: Profile View Page

   Input: Clicks a button to view their match statistics.

   Output: Page displays the number of matches the player has played, the number they have won as well as their win percentage.

Test Case Derivation: The system retrieves the match statistics from the database and sends them to the client. The system then calculates and displays the appropriate statistics for the player.

How test will be performed: Tester will run in a special environment with mock data which includes several matches. Tester will check the HTML for an element which contains the win percentage number and ensure it matches the expected value in accordance with the mock data.

2. TC-PV-2: Profile View Displays Match History

   Testing Type: Functional, Manual, Dynamic

   Initial State: Profile View Page

   Input: Clicks a match on the page.

   Output: Match history displays the match result, the result for each round in that match and the coding problem for each match.

   Test Case Derivation: The system retrieves the match statistics from the database and sends it to the client. The system then displays the appropriate statistics for a match which the user specified.

   How test will be performed: Tester will run in a special environment with mock data which includes several matches. Tester will click on the first match on the list and check the HTML to ensure that the match data is displayed correctly in accordance with the provided mock data.

5.1.5 Leader-board Tests

1. TC-LB-1 Testing Type: Functional, Manual, Dynamic

   Initial State: Home Page

   Input: Clicks the leader-board tab.

   Output: The system displays up to 100 users, sorted by their number of wins in descending order.

   Test Case Derivation: The system receives the leader board input and queries the database for the top 100 users by based on the number of wins of each user. The system then displays the results from the query.

   How test will be performed: Tester will run in a special environment with mock data which includes multiple users with a different number

of wins. Tester will click on the leaderboard tab. Tester will check HTML for each leader-board placement element to ensure that it is in the correct location in accordance with the mock data.

5.1.6 Authentication Tests

1. TC-A-1 Testing Type: Functional, Manual, Dynamic

   Initial State: Login Page

   Input: Clicks the login button. Select an account to login with and put in the correct credentials.

   Output: The system switch's to Home Page signalling a successful login.

   Test Case Derivation: The System receives the login information then checks the authentication service and when successful redirects the user to the Home Page.

   How test will be performed: Tester will click on login button. Tester will input their correct credentials.

2. TC-A-2 Testing Type: Functional, Manual, Dynamic

   Initial State: Login Page

   Input: Clicks the login button. Select an account to login with and put in the wrong credentials.

   Output: The system outputs an error message saying the wrong credentials preventing them from logging in.

   Test Case Derivation: The System receives the login information then checks the authentication service and when failure, returns an error message.

   How test will be performed: Tester will click on the login button. Tester will input the wrong credentials. Tester checks to see if the error message appears.

5.2 Tests for Nonfunctional Requirements

5.2.1 Look and Feel

Interface Navigability

1. TC-LF-1: Interface should be easy to navigate and consistent

   Type: Non-Functional, Manual, Static

   Initial State: Initial state of the game

   Input/Condition: User actions

   Output/Result: Completed game

   How test will be performed: The test will be performed manually by a new user. They will have to navigate through the system to join a game and complete a game and will score the navigation on a scale out of 10 based on how easy they found it to navigate. A minimum of 8/10 should be achieved. If this is not achieved, we will implement the user's feedback and re-run it on a new user.

Screen Size Compatibility

1. TC-LF-2: Problem description and code editor should be fully visible on all screen sizes

   Type: Non-Functional, Manual, Dynamic

   Initial State: The game screen is presented to the user

   Input/Condition: The screen is viewed on devices (simulated or real) with different screen sizes

   Output/Result: The description and editor must be fully visible on each screen size and aspect ratio tested without the need for scrolling the view

   How test will be performed: The test will be performed manually by a member of the team. They will utilize Google Chrome's dev tools to simulate various aspect ratios and screen sizes.

5.2.2 Performance

Maximum Action Latency

1. TC-P-1: The server should synchronize the state of the game to all clients in no more than INPUT_RESPONSE_TIME seconds

   Type: Non-Functional, Performance, Manual, Dynamic,

   Initial State: The game is started and in its initial state

Input/Condition: The user sends actions to the server using various buttons and controls

Output/Result: The server should receive, process and synchronize those actions across all connected clients in no more than INPUT_RESPONSE_TIME seconds.

How test will be performed: The test will be performed manually by a member of the team. They will open Chrome's Dev Tools and record themselves performing the actions and getting the results back. They will then profile the recorded data to verify the time constraint was met.

Maximum Compiling Latency

1. TC-P-2: The server should compile a solution and return results in no more than BACKEND_COMPILE_TIME seconds

   Type: Non-Functional, Performance, Manual, Dynamic,

   Initial State: The game is started and in its first round

   Input/Condition: The user sends a solution to the server.

   Output/Result: Compiled results of the solution returned BACKEND_COMPILE_TIME seconds.

   How test will be performed: The test will be performed manually by a member of the team. They will open Chrome's Dev Tools and record themselves performing the actions and getting the results back. They will then profile the recorded data to verify the time constraint was met.

5.2.3 Operational and Environmental

Browser and Device Compatibility

1. TC-OE-1: The game should be able to be viewed and run on any modern browser

   Type: Compatibility, Manual, Dynamic

   Initial State: The game is not running or being displayed on the browser window

Input/Condition: The game website is accessed on various modern browsers

Output/Result: The game should load all dependencies correctly and render the game properly

How test will be performed: The test will be performed manually by a team member. They will run the game site on the most commonly used modern browsers on multiple devices. They will note if any errors occur.

5.2.4 Security

User authenticity

1. TC-S-1: The server must only accept input from verified clients to update the game

   Type: Security, Manual, Dynamic

   Initial State: The game is started and is in any valid state of play

   Input/Condition: The server receives an action from a simulated WebSocket connection from a client that is not logged in or not part of the game

   Output/Result: The server blocks the connection and ignores any requests sent from it

   How test will be performed: The test will be performed manually by a team member. They will create a WebSocket session outside of the game client and send events to the server. They will note whether or not those actions affected the game.

5.2.5 Cultural

Content Moderation

1. TC-C-1: The system must not have any culturally inappropriate content

   Type: Manual, Static

   Initial State: Current state of the codebase

   Input/Condition: Snapshot of the current codebase

Output/Result: True or False that content is verified not to be culturally inappropriate.

How test will be performed: The test will be performed manually by a team member. They will review the code snapshot and will find any cultural references. If there is at least one cultural reference the test fails. Based on the findings, cultural references are to be removed.

5.2.6 Legal

Legal protection

1. TC-L-1: The system repository is protected by the GNU General Public License

   Type: Manual, Static

   Initial State: Current state of the codebase

   Input/Condition: System repository

   Output/Result: True or False that the repository is protected by the GNU General Public License.

   How test will be performed: The test will be performed manually by a team member. They will review the repository and ensure that legal protection is present.

5.2.7 Health and Safety

Epilepsy shock test

1. TC-HS-1: The system has no flashing lights that could cause harm to the user

   Type: Manual

   Initial State: The game is started and is in its initial state

   Input/Condition: System interface

   Output/Result: True or false that the system did not cause epilepsy.

   How test will be performed: The test will be performed manually by a team member. The member will navigate the site and ensure that there are no flashing lights that could result in an epilepsy shock.

5.2.8 Usability Testing

A semi-structured interview will be conducted with users in the target demographic. To achieve this, a CodeChamp lobby will be setup with a user group. Each session should target around 8-20 users participating in a CodeChamp game. After each user participates in a game, they will be asked several questions. The questions that will be asked can be found in the Appendix Section 7.2. As our platform's goal is to help users code, this survey will be used to evaluate the user's experience using the software, to find out if it's helpful in its current state and to determine potential new features that could be valuable to the users.

5.3 Dependencies Among Tests

Some tests cannot be performed until other parts of the testing plans are completed. Figure 1 demonstrates such dependencies, with an arrow from a test to another representing that the test cannot be performed until the test it points to has been performed.

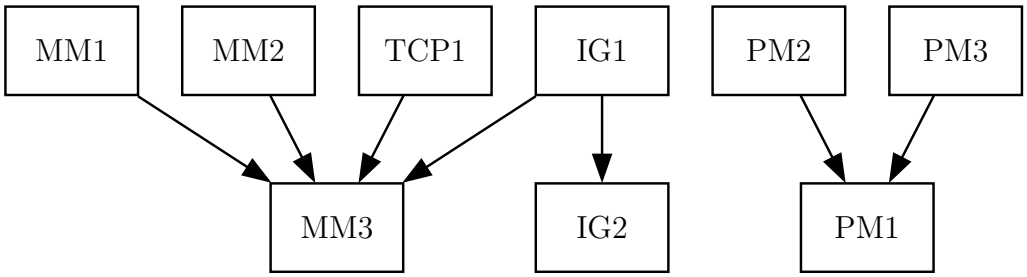

Figure 1: A dependency graph among tests in the CodeChamp system

5.4 Traceability Between Test Cases and Requirements

Test Case ID	Requirement ID	Requirement Description
TC-MM-1	FR.1	Should join a random match.
TC-MM-2	FR.19	Should join an existing page with a code.
TC-MM-3	FR.18	Should create a match.

TC-IG-1	FR.2, FR.3, FR.4, FR.5, FR.8, FR.9, FR.10	Should complete a match.
TC-IG-2	FR.6, FR.7, FR.15, FR.16, FR.17	Should compile solutions from the input.
TC-PM-1	FR.11, FR.12, FR.16, FR.17	Should be able to add new problems (admin).
TC-PM-2	FR.11, FR.12, FR.16, FR.17	Should be able to modify problems (admin).
TC-PM-3	FR.11, FR.12, FR.16, FR.17	Should be able to delete problems (admin).
TC-PV-1	FR.21	Profile should view win percentage.
TC-PV-2	FR.20, FR.22, FR.23	Profile should view match history.
TC-LB-1	FR.24	Leaderboard view should show the players sorted by their scores.
TC-A-1	FR.13	Users should be able to login
TC-A-2	FR.14	System should error when login credentials are wrong
TC-LF-1	NFR.1, NFR.2, NFR.3	Ease of navigation.
TC-LF-2	NFR.2, NFR.3	Responsive display among devices.
TC-P-1	NFR.4	Performance of user actions.
TC-P-2	NFR.10	Performance of user solution compilation.
TC-OE-1	NFR.6	Should run on any modern browser on any device.
TC-S-1	NFR.7	Only requests from authenticated users should be accepted.
TC-C-1	NFR.10	Content of the system should not contain any cultural references.

TC-L-1	NFR.11	System should be protected by GNU License.

Table 3: Traceability Table for Test Cases and Requirements

	MM-1	MM-2	MM-3	IG-1	IG-2	PM-1	PM-2	PM-3	PV-1	PV-2	LB-1	A-1	A-2	LF-1	LF-2	P-1	P-2	OE-1	S-1	C-1	L-1
FR.1	X																				
FR.2				X																	
FR.3				X																	
FR.4				X																	
FR.5				X																	
FR.6					X																
FR.7					X																
FR.8				X																	
FR.9				X																	
FR.10				X																	
FR.11						X	X	X													
FR.12						X	X	X													
FR.13												X									
FR.14													X								
FR.15					X																
FR.16					X	X	X	X													
FR.17					X	X	X	X													
FR.18			X																		
FR.19		X																			
FR.20										X											
FR.21									X												
FR.22										X											
FR.23										X											
FR.24											X										

Table 4: Traceability Matrix for Requirements and Tests Cases

6 Unit Test Summary

Unit testing will be done for all applicable modules, as defined in the Module Interface Specification. For back-end modules, Mocha will be used as the unit testing framework. For front-end modules, Jasmine will be used. In particular, for back-end testing, all API endpoints were tested. This covered operations of creating, reading, updating, and deleting data. The data we used to verify these operations were mocked data that was used by a mock database for the purposes of testing. As our project contains two distinct components (the backend and the frontend), the tests themselves will be placed alongside the code for the back-end in the tests directory tests directory as a separation of concerns. For front-end testing, components will be tested for each page. This involves testing the usability and functionality of menus, buttons, interactions, events and other GUI elements visible to the user using a Chrome runner. In accordance with the Angular convention, the test for each component will be placed in the same directory as the component itself as a '.spec.ts' file. The files for the frontend can be found here.

6.1 Unit Testing Scope

Modules M32, M33, M34 are out of scope as they are not developed by the CodeChamp team. In particular, M32 deals with the environment used to compile and execute code, which will be handled by an external service. Likewise, M33, the database module, is out of scope as it's handled by an external service. Finally, the router module is out of scope as it is provided as an abstraction by the Angular framework and ultimately implemented by the browser used. As shown in the Module Guide, many of the modules which are in scope depend on these modules. In these cases, a mock will be used for the modules that are out of scope. For instance, if a module relies on some database operations such as insertions or retrieval of data, the mock will be used to fake those interactions instead.

7 Appendix

7.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

Symbolic Constant	Value
INPUT_RESPONSE_TIME	2 seconds
BACKEND_COMPILE_TIME	10 seconds
NUMBER_OF_ROUNDS	3

Table 5: Symbolic Constants

7.2 Usability Survey Questions

- From 1 - 10, How easy was the interface to navigate? 1 being unnavigatable, 10 being no issues navigating.
- From 1 - 10, How consistent was the visual theming of the website? 1 being not consistent at all, 10 being super consistent. If users answered less than 8, testers should follow up and ask for feedback.
- Would you prefer the copy code button to copy the whole URL or just the game id? The team will conduct an A/B test in which half the users are tested on copying the URL then trying copying the game id and vice versa for the other group.
- What features do you feel are missing from the game?
- Which programming language should the platform support?

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

1. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage etc. You should look to identify at least one item for each team member.

   (a) **Jasmine (Frontend Unit Tests)**: Frontend of the system, implemented in Angular is to be tested using Jasmine. Jasmine allows modular testing allowing tests to be written for each component individually. Jasmine also allows state manipulation and monitoring of the systems outputs as the state changes. This will be a crucial part of the verification plan to verify the correctness of responsive user interface based on it's state.

   (b) **TestCafe (End-to-End)**: End to End testing is important to ensure the end user experiences correctly corresponds with the requirements. TestCafe works well as it is able to simulate user inputs onto a web page, mimicking a real user. This is a important as it allows building test plans for user scenarios.

   (c) **Jest (Backend Unit Tests) - Youssef Rizkalla**: The backend of the system will be implemented in Node.js and will be tested using Jest. This will help ensure the correctness of the codebase and help identify regressions and weak-points for developers and reviewers, as it can track code coverage across the back-end. Ultimately, this will give the team a high-degree of confidence that the implementation matches the design and requirements needed for the project to succeed.

   (d) **CI testing - Zhiming Zhao**: Continuous integration testing focuses on execution during the continuous integration process. Testing in the CI process allows for rapid feedback and stops the progression of the artifact if minimum quality is not met. CI testing has the benefit of repeatability, it's also able to run builds

or tests in parallel with other team members. To summarize, CI testing allows for iteration and rapid feedback, it also saves the team a lot of time since a failing build is much less severe than a failing deployment.

(e) **Performance testing (Postman/Chrome dev tools) - Tamas Leung**: Performance testing is important to ensure website is to common web standards. Skills to identify common performance issues such as large CSS files or back end algorithms are important to ensure websites are snappy and are a smooth experience for users.

2. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

   (a) **Jasmine (Frontend Unit Tests) - Anton Kanugalawattage** This skill could be acquired by reading and following examples of existing implementation of tests using this technology. Another approach to acquire this skill is to read the documentation or blogs of how this testing technology is used in systems (software testing classes, technology company's engineering blogs). Anton will pursue this skill as he has worked on a project which has utilized a similar testing technology before. Also, as he is the front-end lead this verifying components and changes to the front-end will be a crucial part as he will also we reviewing these changes.

   (b) **Performance testing - Tamas Leung**: In order to learn performance testing, Tamas will study the common issues faced when doing performance testing. For front-end, the google developer tools can generate a lighthouse report in which shows the best practices for web developoment. Tamas will use these reports to learn and understand how to improve the software to run at the webs standards. For back-end, Tamas will study common issues with building heavy load backend systems and will watch videos on methods to improve the servers such as load balancing.

   (c) **Jest (Backend Unit Tests) - Youssef Rizkalla**: This can be developed by observing and critiquing open-source projects and

identifying the testing techniques utilized. Youssef will work on this as he is the back-end lead and main code-reviewer, so it is important to understand testing patterns that will maintain good code health. Observing open-source projects allows us to identify issues that have occurred in other projects and develop a design to get around them. Additionally, testing patterns can be learned from official documentation or trustworthy blog posts that are published by professional Software Engineers. This allows us to learn from engineers who developed the tooling in the first place, which will allow us to understand and set best practices in our code-base. When reviewing code, Youssef can ensure that others are submitting unit tests according to the standards set, and that no regressions are occurring from new pull requests.

(d) **TestCafe (End-to-End) - Dipendra Subedi**: This skill could be developed by reading web tutorials, video examples, and seeing the use of TestCafe in existing open source projects. Web tutorials run through specific examples of how to use TestCafe, and similar code structure shall be utilized when developing end to end tests for CodeChamp. Open source projects will also be useful since previously identified issues can be mitigated when implemented for CodeChamp. As the design lead, Dipendra is knowledge-able in the design of the entire web application, so developing this skill will ensure that CodeChamp simulates what a real use case scenario looks like from start to finish.

(e) **CI Testing - Zhiming Zhao**: To start learning CI testing, it's better to start small and catch easiest errors first. Some of the simplest and easiest to fix errors can end up causing the biggest problems if they make it into production workloads. It's a good choice to configure some automatic testing to take place on developer machines before code is committed. The second approach would be built on top of first, and it would be making security a part of CI testing. Catching security issues early in the software development life-cycle means they are much easier, cheaper and safer to fix. Adding some basic static code analysis tools and dependency checkers can vastly improve the security posture of an application by providing fast feedback and early detection of common security problems and potential vulnerabilities. Since ap-

proach 2 builds on approach 1, it's fair to start learning by starting small and then builds on to approach 2 by implementing security into the CI testing.