Team 13 (DISMS13) - Test Plan Document

1 Front-end Testing

1.1 Front-end – Student

The strategy chosen to test the validity of the HTML and CSS code is by using the W3C validation service. All the files were imported into the service to find errors. The diagram below illustrates an example of executing testing results before and after amending the error code. Some details of the errors will be listed below.
HTML: https://validator.w3.org/
CSS: https://jigsaw.w3.org/css-validator/#validate_by_upload
JavaScript: https://codebeautify.org/jsvalidate

HTML Error Detected:

1. All the image elements do not consist of alt attributes.
2. All the forms' action must be non-empty.
3. Existence of unnecessary backslashes.
4. Misusage of backslash when importing images.
5. Duplicate ID name in different element.
6. Table borders should be implemented in CSS instead of HTML.
7. Stray start and end tags for table, td, and tr tags.
8. The value attribute for input image (button) is not allowed.
9. The first child option element of a select element must have an empty value or no text content.

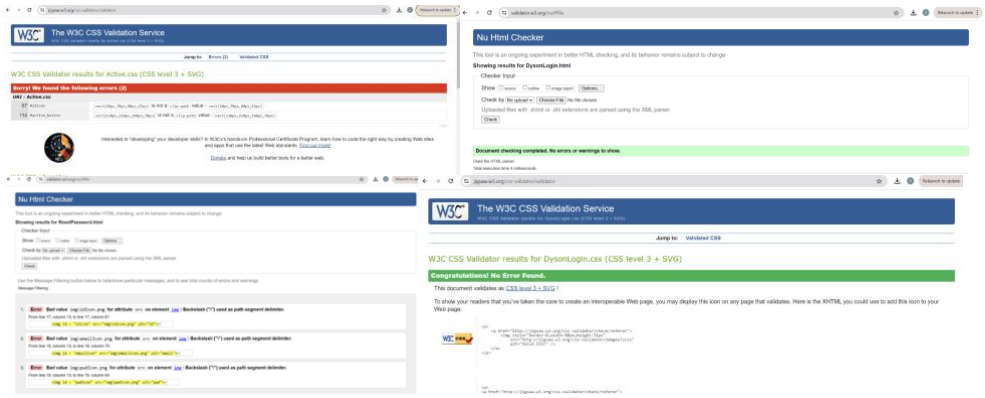

Figure 6.1.1 Validation checking

CSS Error Detected:

1. Misusage of keyword 'rect()'.

Java Script validation results:

Pages	Errors	Status
Active.js	No errors	
Dyson_Login.js	No errors	
Examination.js	No errors	
Feedback.js	No errors	
Main.js	No errors	
Module1.js	No errors	
Module2.js	No errors	
Module3.js	No errors	
Module4.js	No errors	

PersonalTutor.js	No errors	
RestANDActive.js	No errors	
StudentTimetable.js	No errors	
Subjects.js	No errors	

*Some pages were detected with errors, but after searching for information to verify that they were written correctly, they were marked as No Error.

1.2 Front-end – Staff

Validate Tool:

The strategy chosen to test the validity of the HTML and CSS code is by using the W3C validation service. All the files were imported into the service to find errors. The diagram below illustrates an example of executing testing results before and after amending the error code. Also, using the website codebeautify to validate Java script. Some details of the errors will be listed below.

HTML: https://validator.w3.org/
CSS: https://jigsaw.w3.org/css-validator/#validate_by_upload
JavaScript: https://codebeautify.org/jsvalidate

HTML validation results

Pages	Errors	Status
CheckSTuDetail.html	<img> needs to have alt arrtibute	Modified
	The value of the for attribute of the< label > element must be the ID of a non-hidden form control	Haven't Modified
Coursework.html	<option> must not be empty	Modified
	Element<option>without attribute label must not be empty.	Haven't Modified
ModuleManagement.html	not allowed as child of <h3> in this context.	Haven't Modified
StaffDashboard.html	Stray end tag <section>	Modified
StaffTimetable.html	End tag <h3> seen, but there were open elements.	Modified
	The value of the for attribute of the label element must be the ID of a non-hidden form control.	Haven't Modified
Undertaken.html	No errors	
StaffLogin.html	No errors	
Active.html	No errors	
ResetPassword.html	No errors	

*Some pages were detected with errors, but after searching for information to verify that they were written correctly, they were marked as No Error.

CSS validation results

Pages	Errors	Status
CheckSTuDetail.css	No errors	
Coursework.css	No errors	
ModuleManagement.css	No errors	
StaffDashboard.css	No errors	
StaffTimetable.css	No errors	
Undertaken.css	No errors	
StaffLogin.css	Misusage of 'rect'	Modified
Active.css	Misusage of 'rect'	Modified
ResetPassword.css	Misusage of 'rect'	Modified

Java Script validation results

Pages	Errors	Status
CheckSTuDetail.js	No errors	
Coursework.js	No errors	
ModuleManagement.js	No errors	
StaffDashboard.js	No errors	
StaffTimetable.js	No errors	
Undertaken.js	No errors	
StaffLogin.js	No errors	
Active.js	No errors	
ResetPassword.js	No errors	

*Some pages were detected with errors, but after searching for information to verify that they were written correctly, they were marked as No Error.

Defects in Front-End Web Page Design

1. Screen Adaptation Issues
The webpage does not properly adjust to fit all computer screens, resulting in elements being misaligned or displayed incorrectly on different devices.
2. Text Overflow Issues
After connecting to backend data, the text displayed exceeds the bounds of the designated text boxes.
3. Inconsistent Font Sizes
The font size appears differently on various computers, leading to a lack of uniformity across different viewing environments.
4. Element Overlapping on Narrow Screens
When the page width is reduced, elements on the page overlap with each other.

1.3 JavaScript Unit Tests

Overall JavaScript functionality should be unit tested using the Jest testing framework to ensure each function works as expected.

2. Back-end Testing

2.1 Integration Testing:

I. Testing the collaboration between different components. In this project, the main focus is on testing the collaboration of the code under the MVC architecture in the backend.
(1) The collaboration scenario where a message initiated from the frontend is received by the Controller layer and then processed by calling the Service layer.
(2) The collaboration scenario where the Controller layer calls the processing functions in the Service layer.
(3) The collaboration scenario where the Service layer uses the data interaction layer functions to operate on the database data.
The following video links demonstrate the testing of code collaboration within the backend framework using simple GET/DELETE/POST/UPDATE functionalities:
https://youtu.be/ppSz1r8kK00

II. Simulating end-to-end processes in real-world scenarios to verify the correct operation of the entire system.
The following video links demonstrate the testing of simulated real-world scenarios that users would encounter:
https://youtu.be/zmQOo5yWRK4

2.2 Usability Testing:

The main focus is on testing the usability and maintainability of the system, including the ease of deployment on different devices and the scalability of configuration files. The test results show:

I. Deploying the backend framework on different devices:
The backend framework primarily utilizes SpringBoot, and in the design of the configuration file pom.xml, we only selected commonly used dependencies and performed version control, making it compatible with any Java JDK version. This greatly simplified the debugging difficulty in deployment scenarios. Additionally, in the configuration of the database and backend program, we used application.xml to specify parameters such as port and database name, ensuring that every team member could successfully pass the tests through the test files. We chose the code for the Login functionality as the test file to check if the information is displayed on the page and if the backend data is called correctly for normal login.

II. Software deployment:
We encountered some difficulties in software deployment, as each team member's device had different hardware and software versions. Installing different versions of software, such as Node.js, Java JDK, and SpringBoot, meant that members responsible for different modules would only configure the corresponding software. This posed a challenge when selecting a device as the terminal for program execution. We should have employed cloud technologies or container technologies like Docker to simplify the overall deployment difficulty of our project, allowing all members to develop in the same environment without worrying about downloading and switching between different software versions. This is an area where we could have done better.

2.3 Database Testing

Prepare a database instance for testing, and populate it with sample data at the start of the test. Verify the correctness of database operations, such as insertions, updates, deletions, and queries.
We created a local MySQL instance with DBEaver, and insert about 10 columns of sample data in each table for testing. In Springboot framework, we also test and perform SQL operations in Repository class using Mybatis. An example is that in test package, we use .insert(), .select(), .updateByPrimaryKeySelective () methods to test if the database and Java back-end are working well together.

2.4 Unit Testing

Unit testing involves writing test cases for each Repository, Service, and Controller class, covering various edge cases and exception handling. The aim is to test the database operation functions in the DAO layer to ensure the accuracy and integrity of the data. The logic in the Service layer is tested to verify the correctness of various operations. The HTTP request handling and response generation in the Controller layer are also tested.
In practice, we can use Junit as the unit test framework, use Mockito framework to mock dependent objects, use AssertJ to assert test results. For example, in StudentServiceTest class we perform a test using framework and methods mentioned above to ensure the methods in the StudentService class work as expected under the conditions simulated by the tests. This is a crucial part of maintaining the reliability and robustness of the application. When we need to modify this part, we can perform the test after modification to make sure the function is working as expected as it was before modification.
It’s important to note that comprehensive unit tests should cover all possible edge cases and handle potential exceptions to ensure the robustness of the code. We plan to do so but unfortunately we don’t have enough time.

3. Final Testing

Finally, some usability testing between group members should be undertaken using the think aloud method to try and come up with any issues in usability. Also, if there is time this could be sent out as a closed beta test for selected people to test and provide feedback on.