Software Testing Plan

April 5, 2024

Version 1.0

Sponsored by: Dr. Elise Donovan, Sneha Vissa, Adonna Rometo

Mentor: Tayyaba Shaheen

Team members: Dayra Quinonez, Nicole Sylvester,
Rino De Guzman, Bailey Rosato

Table of Contents

1. Introduction

The Diversified Anatomy and Physiology Lab Resource Application is a web application designed specifically for the BIO201: Anatomy and Physiology course at Northern Arizona University (NAU). Our clients, Elise Donovan, Adonna Rometo, and Sneha Vissa have taken notice of the lack of diversity within the course material that the students are learning from. This concern was the motivation behind the creation of this application. Many issues are created through the lack of diversity in medical textbooks that can foster feelings of underrepresentation among the students. For example, the lack of knowledge of different body types and skin tones can lead to misdiagnosis. The primary objective of this application is to be a supplement to the course material, providing students with the opportunity to explore a more diverse range.

The main feature of this application is the ability for students to customize anatomical models according to their preferences. They will be able to alter the skin color, body size, and biological sex of the model. Once the students have customized their models they will be able to export the model for further study. They can use a built-in whiteboard feature to take notes through the web application. The whiteboard will allow the students to import their images and annotate by drawing or typing notes on top. The application renders subunit pages based on the units and subunits of the course. Each page features a rendering of a model and the course content corresponding to the subunit. As well as ensuring that all pages are tailored to the corresponding unit and subunit. There are specific subunits that the users will be able to further interact with the three-dimensional model and apply some of the content within the model scene. This dynamic approach ensures an engaging learning experience for all students.

In this step of implementation, we will be testing our implemented software. Software testing is the process of validating that the software product aligns with the requirements. Its primary purpose is to check that the functionality is reliable and it is at its optimal performance. Ensuring that the software is not lacking nor is coming across any problems that will be a risk to the efficiency and performance of the software. The motivation behind software testing is to identify any bugs within the software. The prevention of bugs will enhance the performance of the software.

The testing strategy for this application will cover several layers. The application will go through unit testing, integration testing, as well as usability testing. Unit testing will allow us to detect errors in specific components of the code. Integration testing aims to identify issues in the interaction between different units integrated. These two types of methods ensure that the components of the application are functioning in isolation and integrated. Users will also be testing the product to ensure that it is user-friendly and that they do not experience issues or confusion as they run through the web application.

2. Unit Testing

Unit testing is a software testing technique where units of a system are tested in isolation to ensure they function correctly. The primary goal of unit testing is to validate that each unit of the software performs as expected and to catch any bugs that may appear. For unit testing of our React project, we will use Jest as the testing framework along with the React Testing Library. Jest will provide our team with an efficient way to write and execute tests and the React Testing Library will help simulate user interactions with components and provides a user-centric approach to testing. We will be using Jest's built-in test coverage which indicates the percentage of code covered by tests to assess the effectiveness of our tests. We plan to test the CourseEntry, MainMenu, Navigation, SubunitPage, ModelPage, Interface, Experience, AnatomyModel, SettingsConfigurator, and UserGuide components. These are the main components of our system and make up most of the functionality of our web application.

2.1 CourseEntry Component

In order to test the CourseEntry component, we have identified two tests; input field handling and form submission. The first input field handling test verifies that the 'handleInputChange' function updates the 'courseCode' state correctly in response to user input changes. Equivalence partitions for this test include a valid input of "BIO201" or "bio201" and an invalid input of "123456". The boundary values are similar, with the valid course code being "BIO201" and "bio201" and the invalid course code being "123456". For this test, the selected inputs are "BIO201" for the valid case and "123456" for the invalid case. By testing these scenarios, we ensure that the CourseEntry component handles input changes accurately and maintains its state management.

The test for form submission evaluates the behavior of the CourseEntry component when the form is submitted, specifically focusing on handling incorrect course codes. Equivalence partitions for this test include scenarios where the submitted course code is correct or incorrect. In this case, the incorrect input is "123456". The test aims to verify that upon submission of an incorrect course code, the component displays an error alert with the message "Incorrect course code. Please try again." Boundary values are implicit in this test, with the valid course code input not being explicitly tested here. By testing this scenario, we ensure that the CourseEntry component communicates errors to the user.

2.2 MainMenu Component

In order to test that the MainMenu component works as expected, we have identified three tests; fetching course data correctly from the database, rendering menu items, and the expansion and collapse functionality of the menu items.

First, the test for fetching the course data from the database utilizes mocked Firebase data to cover the equivalence partitions of empty course data and non-empty course data. The boundary values for this test include no course data, a single unit with one subunit, and multiple units, each with multiple subunits. Next, we examine the rendering of the menu items, focusing

on both units and subunits. This test utilizes similar equivalence partitions and boundary values as the test mentioned above. Last, we assess the expansion and collapse functionality of the menu items. By considering cases with no expanded units, a single expanded unit, and multiple expanded units, we ensure that the MainMenu component handles these interactions seamlessly. Together, these tests provide thorough coverage of the MainMenu component's functionality, ensuring its reliability and accuracy in displaying course titles.

2.3 Navigation Component

The tests for Navigation are similar to the MainMenu component tests in that we will be testing that the component fetches course data correctly from the database, the rendering of menu items, and the expansion and collapse functionality of the units. We mock the course data from Firebase so that we can control the test environment and the data returned by Firebase calls. This enables the simulation of various test cases and the tests can be run independently of the Firebase backend. The Navigation component also has similar boundary values to the MainMenu component, including no course data, a single unit and subunit, and multiple units and subunits.

2.4 SubunitPage Component

To effectively test the SubunitPage component, we will break the testing process into three main areas; data fetching, rendering, and interaction. In order to test data fetching, we will test that the component fetches the correct subunit data based on route parameters. The equivalence partitions include empty and non-empty subunit data. The boundary values are no subunit data, subunits with only a description, and subunits with both a description and image. The test cases will test each boundary value identified.

The second test will evaluate the rendering of the subunit descriptions and images. This test will have similar equivalence partitions, boundary values, and test cases as the previous test. Lastly, we will test the functionality of displaying and closing the full image overlay. The equivalence partitions include clicking to show and close the full image and the boundary values are straightforward in that either there is or is not a full image displayed. The test cases we will include are clicking to show the full image, clicking to close the image, and clicking when there is no image available. By addressing these areas with the respective tests, we can ensure coverage of the SubunitPage component's functionality without duplicating integration testing by testing interactions between the Subunit Page and other components it relies on.

2.5 ModelPage Component

In order to unit test the ModelPage component, we focus on testing aspects specific to its functionality, while trying to avoid overlap tests for the Experience and Interface components that are rendered within the ModelPage. We have identified one test that meets this requirement described above. First, we test that the canvas component is rendered with correct camera settings and equivalence partitions of canvas properties like camera position, shadows, and FOV (field of view). The boundary values are the different camera positions, field of views, and

enabling or disabling shadows. The test cases we include are tests for default camera position and FOV as well as enabling shadows.

2.6 Interface Component

In order to test the Interface component, we first test the toggling of model settings. Equivalence partitions for this test include the button click event, which should toggle the state of the configurator being opened or closed. Boundary values verify that the state toggles between true and false as expected. Next, we test the reset camera button to ensure that clicking the button resets the camera mode to free. Equivalence partitions include the button click event and checking if the camera mode is set. As for the stretch goal test, we set the subunit to “Directional Terms” and then to “Planes of Sectioning” to assert that the correct buttons are rendered. By testing each aspect of the Interface component we ensure its functionality and conditional rendering behavior are correct across various scenarios.

2.7 Experience Component

In order to test the Experience component, we focus our tests specifically on camera controls and lighting. These are broken down into 3 tests; camera controls are rendered, ambient light is rendered, and directional light is rendered. The equivalence partitions and boundary values include camera controls rendered vs. not rendered, ambient lighting rendered vs. not rendered, and directional lighting rendered vs. not rendered in the correct position. These test cases cover the major aspects of the camera controls and lighting in the Experience component, ensuring they are correctly rendered with expected properties.

2.8 AnatomyModel Component

In order to test the AnatomyModel component, we utilize three tests; load in the correct model based on user-selected options, set the material color according to the skin tone, and render the meshes properly.

First, for loading the correct model file test, the equivalence partitions include different combinations of sex (female and male) and body size (1-8). The boundary values are the extremes of these partitions, such as the smallest body size female and the largest body size male. Test cases involve variations of the model parameters such as loading in a female with the largest body size (8) and a male with the smallest body size (1).

Next, in order to test the material color being set based on skin tone, we utilize the equivalence partitions of different skin tone color values. The boundary values are the endpoints of the hexadecimal color spectrum. Test cases include verifying that the material color matches the selected skin color for various skin tone hexadecimal values.

Lastly, to test the rendering of meshes, the equivalence partitions include different biological sex selections (male and female). The boundary values in this case are the extremes of each group of meshes associated with a biological sex. Test cases involve checking the correct meshes are rendered for both male and female. These tests for the AnatomyModel component ensure the accuracy of the customized models being displayed based on the user’s selections.

2.9 SettingsConfigurator Component

In order to test the SettingsConfigurator component, we identified three tests to ensure the skin color, selected biological sex, and body size are updated correctly when a new color is selected, the switch is clicked, or a new drop-down option is selected respectively.

First, the skin tone test's equivalence partitions are valid color inputs of hexadecimal values and invalid color inputs of non-hexadecimal characters or out of range values. The boundary values are the minimum and maximum valid color values ranging from white to black. The test cases include a valid hexadecimal color input, an invalid hexadecimal input, and the minimum and maximum color values.

Next, for the selected biological sex, the equivalence partitions are sex toggled to female or male and there are no boundary values as the input can only be male or female. For this reason, the test cases include toggling from male to female and then from female to male.

Last, for the body size options, the equivalence partitions include valid body size options (integers representing body types from 1 to 8) and invalid body size options (non-integer values or out of range values). The boundary values are the minimum and maximum valid body size values, 1 and 8. To test the updating of body size, we test selecting a valid body size, selecting an invalid body size, and testing the minimum and maximum values. These test cases cover various models and ensure thorough testing of the SettingsConfigurator component's functionality, handling both valid and invalid inputs.

2.10 UserGuide Component

In order to test the CourseEntry component, we have identified one test that provides 100% coverage of the component; testing that the user guide content is rendered correctly. The equivalence partitions for this test include a valid input of mocked course data that is available and correctly fetched, and an invalid input of missing course data. The boundary values include an empty course data array, course data with one unit and one subunit, and course data with multiple units and subunits.

3. Integration Testing

Integration testing is the process of software testing that combines all the separate modules of a product to demonstrate how the combination of services work together to create a smooth user experience. For our purposes, integration testing will focus on the flow of the application's pages and extensions to help identify any missing functionality and dependencies before next stages of project development. For the integration testing of this application, we will be using the same tools that were used for unit testing, including the React Testing Library with the Jest framework. To ensure that the interfaces of the application are correctly displayed, we will be simulating user interactions to authenticate communication between the front-end and back-end and then validate the behavior of the component.

In the following sections, the integration testing will be broken down into the components based on the flow of user interaction in the web application.

3.1 Course Entry Component

Beginning with the course entry component of the web application, one of the key functions of this component is to limit user traffic of the application by only allowing users with the correct code to access the page. In terms of cases regarding the correct or incorrect code to access the application have been tested with unit testing. The goal of integration testing for this component is to ensure that once the user has entered the correct code, the application correctly navigates the user to the main menu component.

The integration points that will be tested are the handleSubmit function that is triggered when a user submits the course code and the navigate function which is responsible for redirecting the user. To verify the integration of the two modules, the user interaction of entering the correct course code will be observed. The expected outcome of this test case is that the application will correctly navigate to the main menu page.

3.2 Main Menu Component

Next, once directed to the main menu, there are three separate integration points that will be tested. The main menu is responsible for displaying the correctly retrieved units and subunits from the database and then allows users to easily navigate to the subunit pages.

For purposes of integration testing, we will first be testing that once a user clicks on the arrow, the menu item will expand into a list of its subunits, and clicking the arrow once expanded will collapse the subunits. The initial state of the unit items is collapsed, only displaying the unit title as a menu item. All units have an arrow to expand the subunits except for the user guide menu item, which will be tested separately. Again, for this case we will simulate user interaction and verify the actions of the expansion or collapse of the subunits by observing the behavior. The expected outcome for clicking the arrow to expand a unit is that it will reveal the subunits and the arrow icon will then be in the expanded state. Conversely, the expected outcome for clicking the arrow to collapse a unit is that it will hide the subunits and the arrow icon will revert to its original state.

Once we have tested that all subunits are visible for users, we will now test that the subunit menu items correctly navigate to the correct subunit page. As a note, the testing for this case assumes that the user has already expanded the unit into subunits. To test the navigation of each subunit, each subunit's navigation to the corresponding subunit page will be tested. User action will be simulated by testing the Link action of the subunit item and will be checked by asserting the outcome of clicking the subunit.

Finally, the last integration point that we will test is the correct navigation of the user guide page. Unlike the other menu items, the user guide does not have an expandable state. To assert that the correct page is shown when the user guide menu item is clicked, the Link action of the menu item will be tested by simulating user interaction. The navigation can be verified by confirming that the correct interface is displayed.

3.3 Subunit Component

Once a user has navigated to the desired subunit, they will be able to interact with the model and use the whiteboard feature. For this component, there are five total integration points that will be tested to ensure that the user experience within the subunit page is seamless.

First, we will test that a user is able to download the PNG file of the model. For this test, we are wanting to check that the download of the exported model is triggered by clicking on the export button and that the browser is appropriately handling the download. For this case, it should be noted that this testing does not encompass all possible browsers and for our purposes only works for local hosts. This test requires that a mock function is created to replicate the actions of the exportToPNG function to test that the exportToPNG function is appropriately being called to trigger the download. To verify that the user will be able to view the PNG, we will observe the expected behavior of the download to confirm that a downloaded PNG can be opened on the user's device. The next integration point for the subunit component is the display of the whiteboard feature that users are able to annotate with. To simulate the integration of the whiteboard within the subunit page, we will test that when the subunit component is rendered, the whiteboard subcomponent can be found in the subunit component by validating with a test identification string. Another whiteboard subcomponent functionality that we want to test is the prompting of a user to upload a file from their device onto the whiteboard. For this integration test, the functionality of uploading the file is within the 'tldraw' package that is responsible for rendering the whiteboard. First, to test the integration between the application and the 'tldraw' package we will render the whiteboard and then simulate user action by clicking the upload button. A testing file will be created and once the upload file action is triggered, it will be checked if the file is present in the DOM to ensure that the file was uploaded.

Then, we will test the display of the 3D model on the subunit page by first checking that the correct model is rendered and then testing that the ModelPage component is correctly integrated with the Interface and Experience component. They are responsible for the rendering and user interactivity of the model. To begin, the ModelPage component will be rendered to verify that the 3D model is correctly rendered for the user from the Experience component. The

Experience component can also be verified by observing the outcome of the rendering of the model, checking that the model is visible. Next, to ensure that the Interface functionalities are reflected correctly on the model, we will simulate user interactions by asserting changes in the skin tone, body type, and biological sex. When changes within the Interface are made, it must also be checked that those values are passed to the ModelPage. This integration point will confirm that the Experience and Interface components are communicating with the ModelPage component that is displayed within the Subunit component.

3.4 Navigation Component

On the subunit page, there is also a sidebar menu that allows the user to navigate to the other pages of the web application, including the main menu. The functionality of this menu is the same as the main menu for the units, subunit, and user guide page. The steps from the main menu portion of the integration testing will be replicated here for this navigation component.

The two unique integration points for this component are to test that the main menu is correctly being navigated to and that the side navigation closes when a new subunit is chosen. First, to test that the navigation component correctly navigates the user to the main menu page, we will take a similar approach to how we tested the user guide page. To assert that the correct page is shown when the main menu item is clicked, the Link action of the menu item will be tested by simulating user interaction. The navigation can be verified by confirming that the correct interface is displayed. Next, we will test that the subunit component and navigation component are correctly integrated by confirming that once a subunit menu item is clicked, the navigation menu is closed. The closing of the menu is controlled by the onClick action with the toggleMenu function. When a subunit item is clicked, we will assert that the menu closed by testing the state of the menu.

4. Usability Testing

Usability testing is one of the software testing methods that we will be employing on the application. By utilizing this method, we will be evaluating the functionality of our product using real end users. To begin the process, users are expected to complete tasks that have been predetermined for them. In terms of this project, the list of tasks will be based on the application's features to ensure that everything is running smoothly. While the users are testing the application, researchers will observe and take notes of the entire process. If any problems arise during the testing process then it will be noted. Any comments or suggestions will be taken into consideration to improve the application after thorough testing and review.

As this project's core functionality is applying course content and interactivity to diverse models, it is important to have people thoroughly test the product. The application is meant to be a study tool, targeting the students at NAU who are taking Anatomy and Physiology. Gathering data and feedback from end users would be vital to ensure that the application is functioning and that we are providing a high-quality product before it is launched. We determined the best end users to test this application would be NAU students who have taken this course before. However, as the group of students is specific, there is a limit to gathering students who attend NAU and have taken BIO201. Fortunately, the process of gathering the students needed to test the software will not be difficult as our clients were able to connect us to a group of students in a program, EITS. The program's main goal is to explore teaching design to build an inclusive learning environment. We contacted the group of students to schedule a time and day that worked for all of us. Due to the limited EITS students that have taken BIO201, we also have decided to have users who have not taken the course part take in the usability session to observe how much knowledge a student will take from utilizing the application. The goal of these sessions is to ensure that the application is running smoothly and that no problems are encountered.

The next steps are to create a list of tasks that involve all of our features ensuring that we are meeting the functional requirements expected by our clients. The functionality that we will be testing will be the navigation of the application, the interactions with the three-dimensional model, the rendering of the model, the exportation of content, and the functionality of the whiteboard: importing, drawing, typing, and exporting notes. We are testing that they are functioning well on the user interface and that students will be able to complete these tasks without coming across any issues throughout the observation.

Regarding the Course Entry page, tasks for the usability testers ensure that the page loads with the Anatomia logo and a form field that asks the user for a unique course code. Submitting an incorrect passcode appropriately responds with an error message, and submitting a correct passcode redirects the user to the BIO201 Menu.

For testing the user's ability to navigate through the application, tasks encompass checking if each page category can be accessed and displayed correctly to the user. Within the application, there are three main page categories: Main Manu, Subunit, and User Guide. The Main Menu page should appropriately display the different units and User Guide sections. The

user will click on “Unit 1” and will be able to see the drop-down of the subunits available. Clicking on the subunit link shall redirect the user to the corresponding subunit page. Once the corresponding subunit page is loaded. The unit and subunit name shall be displayed correctly at the top of the page. Clicking on the hamburger button on the top left side will display a sidebar menu that allows the user to navigate through the other subunit pages. Clicking the exit button returns the user to the subunit page. The User Guide page should correctly display a download button to the user. The user will click on the download button to download the pdf. Once the pdf is downloaded, the user will open the pdf file on their computer to verify that the file is indeed the User Guide and is accessible. As the navigation capability is now verified and tested, the next step is to test the usability of the 3D model controls

These controls enable users to interact with and manipulate the 3D models within the application. The first control feature is rotation, the user shall rotate the 3D model around each axis to verify that the user can have a comprehensive view from different angles. The user will then use the zoom feature to be able to view the body from different distances. The user will continue testing the camera functionalities by panning the camera (which shifts the viewpoint horizontally or vertically) and pressing the “reset camera” button which should bring the camera to its default position/orientation. The user will also test the functionality of the skin color by choosing from the color presets and trying a color of their choice. The user will change the biological sex between biological male and biological female by clicking on the switch, the appropriate biological sex should be shown. Lastly, the user shall be able to export the 3D model by clicking the “Export Model” button. The user should be able to see the downloaded 3D model file as a PNG image. The last main functionality of the application is the whiteboard, which is the last step in usability testing.

The user will conduct several tasks to assess the performance and usability of the whiteboard feature. First, the user will import an image they choose into the whiteboard, i.e. (the exported 3D model image) and the appropriate image should be displayed. Next, the user will be prompted to utilize the drawing and text tools. This allows the user to create their own annotation and notes to facilitate an interactive learning experience. In addition, the user will be able to zoom in and out of the whiteboard area which enables users to enlarge content for better visibility or to get a birds eye view of the notes they have written on the whiteboard. Finally, the user will be able to export the whiteboard screen by pressing the “export” button and the downloaded PDF shall be accessible and viewed by the user.

5. Conclusion

The development and testing of the Diversified Anatomy and Physiology Lab Resource Application have been thorough and meticulous. The rigorous testing process encompassing unit testing, integration testing, and usability testing has been vital in ensuring the reliability, functionality, and user-friendliness of the application. Unit testing was conducted extensively across various components of the application to validate the functionality of each unit in isolation. Integration testing focuses on assessing the seamless interaction between different components of the application. Lastly, usability testing involves the interaction with real end-users to create a feedback loop that would provide valuable insights into the user experience and to highlight areas of improvement. Furthermore, the integration of feedback from student testers is incredibly important to refine and enhance the application's functionality and usability. By incorporating feedback and addressing identified issues, the application will be able to evolve into a comprehensive and effective tool for NAU's BIO 201 curriculum.

In summary, the Diversified Anatomy and Physiology Lab Resource Application represents a significant step toward promoting diversity, inclusivity, and engagement within the field of Anatomy and Physiology education. By providing students with exposure to diverse body types and skin tones, the application not only enriches the educational experience but also contributes to a more inclusive environment. As the application continues to be refined and expanded, it holds the potential to make a meaningful impact at NAU and beyond.