Final Year Project
Testing Strategy and Test Plan

Student Name	**Lewis King**
University	**Aston University**
Module Code	**CS3270**
Module Name	**Testing and Reliable Software Engineering**
Student Number	**159025495**
Student Email	**kingl@aston.ac.uk**

Word Count	**2973***
Submission Date	**26/04/2019**

* Excluding the title page, contents, appendices and references.

Introduction

This report documents the task of creating a testing strategy and test plan for the creation of the CS3010 Final Year Project. The sections below contain the *testing strategy*, *test plan & test cases*, *future testing planned*, and an *evaluation & conclusion*.

Testing Strategy

The following testing strategy will provide an overview of the overall approach to testing that will be used for this project, including the types of testing to be used. This testing strategy could be suitable to use across different projects as it will not detail specific test cases.

Environment Requirements

For this project, the OS is Windows 10 (Home Edition). The software applications required are Unity (2018.3.0f2) and Visual Studio (Community 2017 15.9.4). In Visual Studio, the LightBDD and NUnit packages are required. *Figure 1* shows how to set up LightBDD and NUnit3 in Visual Studio (NuGet, 2019).

Figure 1: LightBDD setup.

STEP	DESCRIPTION
1	In Visual Studio, go to Tools > Extensions and Features. Install LightBDD for Visual Studio.
2	Right click the solution and select Manage NuGet Packages for Solution. Search and install LightBDD.NUnit3. This will install up to 4 packages. Then search and install NUnit3TestAdapter, otherwise the future tests will not run.
3	Unity will remove the .dll files from Visual Studio, unless they are manually dragged into the Plugins folder inside the Unity application.
4	Create a new folder called Editor, and inside this add a new item > LightBDD.NUnit3 Feature Test Class.

5	For each test, implement the following code and replace “AMethod” as appropriate.
	
	[Scenario]
	public void AScenario() {
	  Runner.RunScenario(
	  Given => AMethod(),
	  When => AMethod(),
	  Then => AMethod());
	}
	
	Multiple Runner.RunScenario() methods can be included in each AScenario().

Types of Testing

Quadrant 3 testing will be applied during the early creation stages of the project, and in the future stages it would be more appropriate to apply Quadrant 4 testing, as it is more suited to product nearer to completion. For now, a combination of scenario and exploratory testing, belonging to Quadrant 3, will be used.

Scenario testing is conducted by preparing a test script, and then going through this to check the system behaviour is expected. Exploratory testing doesn't have to be fully scripted, and involves cases where the system may not behave as expected or may fail. Exploratory testing will be used to supplement story/scenario tests (Esterle, 2019). The testing below is also an example of Behavioural Driven Development unit testing.

Black-box testing is a technique that doesn't consider the internal structure of a system or its component/s. Other names include functional, data-driven or behavioural testing. The goals for black-box testing include validating the delivery of client requested functionality, and the process can involve the client for unbiased testing. Whilst this technique doesn't require specialist coding knowledge, it doesn't test any hidden functions in the code (Esterle, 2019).

Contrasting with black-box testing, white-box testing does consider the internal structure of systems or components of a system, and can verify all code is reachable. Other names for this technique include structural or path testing. Whilst this technique can detect any redundant code, the focus is on the code and not the actual requirements (Esterle, 2019).

It's important to recognise that an effective strategy will incorporate both black/white-box testing to make use of their advantages. Whilst agile testing is reliant on the interactions between the testers and developers and stakeholders, specific testing will vary depending on the project and business requirements (Esterle, 2019).

The black-box test design technique used for this project is decision tables, and the white-box technique is condition coverage, as using both will assist in determining the minimum number of use cases/tests required. Given the type of testing shown below, decision tables are more suitable than boundary value analysis and equivalence partitioning. Condition coverage shares similarities with decision tables, and their simple representation enables easier interpretation (Esterle, 2019).

It is important to note that not all types of testing mentioned above will apply for each scenario below, as some scenarios only test one specific outcome, and time constraints.

Future testing will include User Acceptance Testing, performance and load testing, and security testing which will be explained under *Future Testing Planned*.

Test Plan & Test Cases

Acceptance Criteria

For this, user stories were collated and converted into functional/non-functional requirements, then converted into scenarios using the Gherkin format.

This template was used to create the user stories: *As a* [role], *I want* [feature], *so that* [benefit]. By using this template, it will be clear what feature is requested by which role and importantly the value of implementing the feature. *Figure 2* shows these user stories, and *Figure 3* shows the newly created requirements based on these user stories.

Figure 2: User stories

ID	User Story
1	*As someone who is busy* *I want to interact with a PT in my own time* *So that I can work out whenever I feel like.*
2	*As someone who has started going to the gym* *I want to see demonstrations of exercises* *So that I can use the correct form.*
3	*As someone who goes to the gym* *I want workout plans* *So that I can tailor my workouts based on my needs and abilities.*
4	*As a user* *I want to see visual demonstrations of workout exercises and text instructions* *So that I can correctly perform the exercises with additional guidance.*
5	*As a user* *I want to see realistic animations and real-time responses* *So that my experience is satisfactory.*
6	*As someone who is embarrassed to go to the gym* *I want to work out in my home* *So that I can lose weight.*

Figure 3: Functional/Non-functional requirements

ID	TITLE	FUNCTIONAL OR NON-FUNCTIONAL	DESCRIPTION
1	Open Application	Functional	Can I open the application?
1.1	-	Non-Functional	Can I open the application 24/7?
1.2	-	Non-Functional	Can the application open and load within 30 seconds?
2	General Communication	Functional	Can I communicate with JimBot?
2.1	-	Functional	Can JimBot communicate with me?
2.2	-	Non-Functional	Can communication occur in real-time?
2.4	-	Non-Functional	Does JimBot look realistic?
3	Workouts	Functional	Can I request workouts?
3.1	-	Functional	Can JimBot output workouts?
3.2	-	Non-Functional	Can JimBot output workouts in real-time?
4	Exercise Descriptions	Functional	Can I request exercise descriptions by interacting with the screen?
4.1	-	Functional	Can JimBot output text exercise descriptions?
4.2	-	Non-Functional	Can JimBot output descriptions in real time?
5	Muscles Affected	Functional	Can I request the muscles affected by the exercise by interacting with the screen?
5.1	-	Functional	Can JimBot output the relevant muscles diagram?

Acceptance Tests were then created based on the relevant requirements from above. Due to issues with Unity, which will be explained under *Evaluation*, and time constraints, it wasn't possible to use every requirement for scenario and exploratory testing. These Acceptance Tests are shown as behaviour driven development scenarios, which are explored below, and were written using the Gherkin format.

Feature: [Overall use of feature]

Scenario: [Specific use of feature]
Given [context / original state]
When [event / action to be tested]
Then [outcome / expected result]

Testing

Exercise Details

Scenario 1

This relates to handling an invalid input for requesting details about the exercises. If an invalid input is provided, then the application should be able to handle this and not throw any exceptions or crash.

It's necessary to handle invalid inputs for requesting exercise details because otherwise the application may crash or throw exceptions and this would confuse the user. *Figure 4* shows the details and result of this test case, with Appendix 7 providing the evidence.

Following the first test being successful, in order to experiment (for exploratory testing) with more invalid input values, a 'null' value was used as the input value. This was to check the application could handle this correctly. As a result of this exploratory testing, a bug was found in the code that the application failed to handle this 'null' value, so the code was amended to resolve this. The test cases in *figure 4* were updated to include these new unit tests.

Figure 4: Test cases.

ID	Description	Original State (Given)	Action Tested (When)	Expected Result (Then)	Result
1	The exercise details should not be found from an invalid input.	The "Exercises" array is not null.	The "Exercises" array is checked for the input value = "string.Empty".	Index = -1. The input value does not exist in the array.	Pass
2	The exercise details should not be found from an invalid input.	The "Exercises" array is not null.	The "Exercises" array is checked for the input value = "null".	Index = -1. The input value does not exist in the array.	Pass
3	The exercise details should not be found from an invalid input.	The "Exercises" array is not null.	The "Exercises" array is checked for the input value = "Fake input".	Index = -1. The input value does not exist in the array.	Pass

Scenario 2

This relates to handling an valid input for requesting details about the exercises. If a valid input is provided, then the application should be able to handle this and return the respective data.

It's necessary to ensure that the correct values from the Array are returned following a valid input as to not confuse the end user with the wrong information about the exercise. *Figure 5* shows the details and result of this test case, with Appendix 7 providing the evidence.

For exploratory testing, another valid input value was used, to ensure that this could also be found in the Exercises Array. The test cases in *figure 5* were updated to include this second unit test.

Figure 5: Test cases.

ID	Description	Original State (Given)	Action Tested (When)	Expected Result (Then)	Result
1	The exercise details should be found following a valid input.	The "Exercises" array is not null.	The "Exercises" array is checked for the input value = " Air Bikes".	Index != -1. string.IsNullOrWhiteSpace (exerciseDetails.GetArrayValue (index, 1)) == false	Pass
2	The exercise details should be found following a valid input.	The "Exercises" array is not null.	The "Exercises" array is checked for the input value = " Pullups".	Index != -1. string.IsNullOrWhiteSpace (exerciseDetails.GetArrayValue (index, 1)) == false	Pass

Response Handler

It is vital that this class is properly tested, because it affects the flow of the application. "ResponseHandler" will interpret the response from the external chatbot DialogFlow API. If the response isn't handled correctly, this would confuse the end user and may cause problems with the application. For example, if the user requests an arms workout and the application closes.

Scenario 1

This relates to handling an empty response from the API. The API can return an empty response when the user input does not match any of the training phrases assigned to the external chatbot, e.g. if the user provides an empty string or gibberish. If this empty response is returned from the API, then the application should play a specific audio clip named "DefaultErrorResponse", informing the user their input has not been matched to the chatbot training phrases.

Without this feature in the application, if the user did not provide a suitable input, then the application would have no way of relaying this information to the user. *Figure 6* shows the condition coverage table, *figure 7* shows the details and result of this test case, with Appendix 7 providing the evidence.

Originally the only input tested here was 'string.Empty'. For exploratory testing, to validate the application could handle more invalid values, different invalid inputs were provided similar to *Scenario 1* from *Exercise Details*. The test cases in *figure 7* were updated to include this second unit test.

Figure 6: Condition coverage table.

string.Empty	null	Fake Input	Outcome
T	F	F	audioToPlay = "DefaultErrorResponse"
F	T	F	audioToPlay = "DefaultErrorResponse"
F	F	T	audioToPlay = "ChatbotResponse"

Figure 7: Test cases table.

ID	Description	Original State (Given)	Action Tested (When)	Expected Result (Then)	Result
1	The returned value should be "DefaultErrorResponse" from an invalid input.	Input value = "string.Empty". audioToPlay = "".	The response handler is invoked with the input value.	audioToPlay = "DefaultErrorResponse".	Pass
2	The default error response audio string should be returned from an invalid input.	Input value = null. audioToPlay = "".	The response handler is invoked with the input value.	audioToPlay = "DefaultErrorResponse".	Pass
3	The returned value should be "ChatbotResponse" when the input is valid.	Input value = "Fake input". audioToPlay = "".	The response handler is invoked with the input value.	audioToPlay = "ChatbotResponse".	Pass

Scenario 2

This relates to the *"CheckDiagramToShow()"* method and how the chatbot response is interpreted and if a user has requested a specific type of workout, a value will be assigned to a string field named *"diagramToShow"*. This will then be used if the user wants to view the diagram of the muscles affected from the exercise.

It is important to implement this feature, because this will attain the acceptance criterion, that users want to see a diagram of the muscles affected from the exercise, created from the user stories, see *figures 2 and 3*.

Exploratory testing here involved using an invalid input to see what value the returned diagramToShow variable would be. In this case, the diagramToShow variable was returned with an empty value, which was expected and correct.

Figure 8 shows the decision table created for this scenario, *figure 9* shows the respective condition coverage table, then *figure 10* illustrates the details and result of this unit test case, with Appendix 7 providing the evidence.

Figure 8: Decision table.

		Rules					
		Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
Conditions	Arms	T	F	F	F	F	F
	Back	F	T	F	F	F	F
	Chest	F	F	T	F	F	F
	Core	F	F	F	T	F	F
	Legs	F	F	F	F	T	F
	Fake Input	F	F	F	F	F	T
Actions	diagramToShow is ArmsDiagram	X					
	diagramToShow is BackDiagram		X				
	diagramToShow is ChestDiagram			X			
	diagramToShow is CoreDiagram				X		
	diagramToShow is LegsDiagram					X	
	diagramToShow is empty						X

Figure 9: Condition coverage

Arms	Back	Chest	Core	Legs	None	Outcome
T	F	F	F	F	F	diagramToShow = armsDiagram
F	T	F	F	F	F	diagramToShow = backDiagram
F	F	T	F	F	F	diagramToShow = chestDiagram
F	F	F	T	F	F	diagramToShow = coreDiagram
F	F	F	F	T	F	diagramToShow = legsDiagram
F	F	F	F	F	T	diagramToShow = “”

Figure 10: Test cases.

ID	Description	Original State (Given)	Action Tested (When)	Expected Result (Then)	Pass/Fail
1	The returned value for diagramToShow should be "ArmsDiagram" when the input is "Arms workout".	Input value = "Arms workout". diagramToShow = "".	The response handler is invoked with the input value.	diagramToShow = "ArmsDiagram".	Pass
2	The returned value for diagramToShow should be "BackDiagram" when the input is "Back workout".	Input value = "Back workout". diagramToShow = "".	The response handler is invoked with the input value.	diagramToShow = "BackDiagram".	Pass
3	The returned value for diagramToShow should be "ChestDiagram" when the input is "Chest workout".	Input value = "Chest workout". diagramToShow = "".	The response handler is invoked with the input value.	diagramToShow = "ChestDiagram".	Pass
4	The returned value for diagramToShow should be "CoreDiagram" when the input is "Core workout".	Input value = "Core workout". diagramToShow = "".	The response handler is invoked with the input value.	diagramToShow = "CoreDiagram".	Pass
5	The returned value for diagramToShow should be "LegsDiagram" when the input is "Legs workout".	Input value = "Legs workout". diagramToShow = "".	The response handler is invoked with the input value.	diagramToShow = "LegsDiagram".	Pass
6	The returned value for diagramToShow should be "" (empty string) when the input is "Fake input".	Input value = "Fake input". diagramToShow = "".	The response handler is invoked with the input value.	diagramToShow = "".	Pass

Scenario 3

This relates to the *"CheckGoHome()"* method and how a *"Bye"* response from the chatbot is handled. This would occur when the user inputs one of the training phrases relating to *"Bye"*. When this response is returned from the API, a value of *"true"* will be assigned to the Boolean variable named *"goHome"*. Another class will be referencing this method, and will then redirect the user to the home screen if the value of "*goHome"* is *"true"*.

It is important to implement this method because then the application will be able to redirect the user to the home screen providing the response is *"Bye"*, otherwise nothing would happen and

this may confuse the user. *Figure 11* shows the condition coverage table and *figure 12* shows the details and result of this test case, with Appendix 7 providing the evidence.

Figure 11: Condition coverage table.

Bye	Fake Input	Outcome
T	F	goHome = true
F	T	goHome = false

Figure 12: Test cases table.

ID	Description	Original State (Given)	Action Tested (When)	Expected Result (Then)	Result
1	The returned value for goHome should be true when the input is “Bye”.	Input value = “Bye”. goHome = false.	The response handler is invoked with the input value.	goHome = true.	Pass
2	The returned value for goHome should be false when the input is “Fake input”.	Input value = “Fake input”. goHome = false.	The response handler is invoked with the input value.	goHome = false.	Pass

Scenario 4

This relates to how the chatbot API response is interpreted for the *“CheckPanelsToSetActive()”* method. If the response contains a specific phrase, a value will be added to an ArrayList named *“panelsToSetActive”*. These tests will ensure the correct panels are added to the ArrayList, and this ArrayList will then be interpreted by another class that will make the panels active in Unity.

It is important to implement this feature, because it is impossible to fit everything onto a single panel, therefore various panels are created in Unity to contain the information. It is vital to display the correct panel based on the user input, as to prevent the user from getting confused if their input doesn’t do anything. *Figure 13* shows the decision table created, *figure 14* shows the details and result of this test case, with Appendix 7 providing the evidence.

Exploratory testing here involved experimenting with an invalid input to see what value the response. In this case, the ArrayList returned had a count of 0, which was expected and correct, and as the count was 0, it wasn’t necessary to check if it contained the input value as with the previous test 1, 2 and 3. *Figure 13 and 14* were updated to include this new exploratory unit test.

Figure 13: Decision table.

		Rules			
		Rule 1	Rule 2	Rule 3	Rule 4
Conditions	which area would you like to train today	T	F	F	F
	-	F	T	F	F
	Ok. Let's do the workout	F	F	T	F
	Fake input	F	F	F	T
Actions	panelsToSetActive contains AreasToTrainPanel	X			
	panelsToSetActive contains ExercisesPanel		X		
	panelsToSetActive contains StartWorkoutPanel			X	
	panelsToSetActive is empty				X

Figure 14: Test cases table.

ID	Description	Original State (Given)	Action Tested (When)	Expected Result (Then)	Result
1	The returned value for the panel should be "AreasToTrainPanel" when the input is "which area would you like to train today".	Input value = "which area would you like to train today". ArrayList.Count = 0.	The response handler is invoked with the input value.	ArrayList.Count > 0. ArrayList.Contains ("AreasToTrainPanel")	Pass
2	The returned value for the panel should be "ExercisesPanel" when the input is "- ".	Input value = "-". ArrayList.Count = 0.	The response handler is invoked with the input value.	ArrayList.Count > 0. ArrayList.Contains ("ExercisesPanel").	Pass
3	The returned value for the panel should be "StartWorkoutPanel" when the input is "Ok. Let's do the workout".	Input value = "Ok. Let's do the workout". ArrayList.Count = 0.	The response handler is invoked with the input value.	ArrayList.Count > 0. ArrayList.Contains ("StartWorkoutPanel").	Pass
4	The returned value for the panel should be "" (empty string) when the input is "Fake input".	Input value = "Fake input". ArrayList.Count = 0.	The response handler is invoked with the input value.	ArrayList.Count = 0.	Pass

Scenario 5

This relates to the *"CheckStartWorkout()"* method and how the chatbot response is interpreted and if a user has requested to begin the workout. When the user requests a workout from the chatbot, a *"true"* value will be assigned to a Boolean field named *"startWorkout"*. This will then be referenced from another class and will handle the visual aspects for Unity.

It is important to implement this feature, because this will be used to commence the workout with a countdown timer, so the user can perform the exercises as if they would with a personal

trainer. *Figure 15* shows the condition coverage table and *figure 16* shows the details and result of this unit test case, with Appendix 7 providing the evidence.

Figure 15: Condition coverage table.

Let's do the workout	Fake Input	Outcome
T	F	startWorkout = true
F	T	startWorkout = false

Figure 16: Test cases table.

ID	Description	Original State (Given)	Action Tested (When)	Expected Result (Then)	Result
1	The returned value for startWorkout should be true and the panel to be made active should be "StartWorkoutPanel" when the input is "Ok. Let's do the workout".	Input value = "Ok. Let's do the workout." startWorkout = false.	The response handler is invoked with the input value.	startWorkout = true. panelsToSetActive.Contains ("startWorkoutPanel").	Pass
2	The returned value for startWorkout should be false when the input is "Fake input".	Input value = "Fake input". startWorkout = false.	The response handler is invoked with the input value.	startWorkout = false.	Pass

Future Testing Planned

User Acceptance Testing

This is a type of manual testing and is part of Quadrant 3. Real end users will test the software to ensure that it can handle the tasks defined by the requirements and user stories in real-world scenarios. UAT can be considered as a component of black-box testing, due to the fact that the end users will not see the internal components of the application.

UAT would involve solely testing the Unity project. This will naturally reference the C# scripts so would aid in ensuring these scripts function properly, however it would purely involve interacting with the UI, so the 10 users testing the product would not have direct access to the code.

For this project, the functional and non-functional requirements shown under *Test Plan & Test Cases > Acceptance Criteria* in *figure X* would be provided to the group of 10 users for them to validate these requirements have been fulfilled, based on their testing. UAT is extremely important because of this.

Performance & Load Testing

Performance testing is part of Quadrant 4, and can be used to confirm if the system can handle the required number of users, and monitoring average response times. The goal of performance testing is to locate and then eliminate any performance bottlenecks and proceed to identify a baseline for future performance testing (Esterle, 2019).

For this project, the Performance Testing Extension for Unity Test Runner extension from the Unity Asset Store would need to be installed. This provides an API and test case decorators to note measurements within the Unity Editor. Configuration metadata including build and player settings can be collected to assist in comparing data against different hardware and configurations.

Load testing is also part of Quadrant 4 and can be used to determine how the system performance changes when the number of users changes. It can show the load limit of the system and what happens at maximum load regarding the system and/or network.

For this project, the Web performance and load testing tools component would need to be installed from the Visual Studio Installer. Microsoft provide details about the initial setup, and this can produce detailed results for load testing the project (Microsoft, 2018).

Security Testing

Security testing also belongs in Quadrant 4 with performance and load testing. In contrast with performance and load testing, it's difficult to say how security testing would be as important for this application. The application doesn't require any user information, and there isn't a database directly linked to the application. The only data store is located on DialogFlow, and this purely contains chatbot data, for example the responses it returns, as shown in the tests above.

An intruder would have to intercept the personal Gmail credentials used for the chatbot, and also then figure out the semantics of generating the access token required for the DialogFlow API to be able to manipulate the existing data there. There is no personal data stored on the DialogFlow API and it is owned by Google, meaning there will be a high level of security. Also, emails are sent out when a Gmail account is used on an unfamiliar device, so it would be simple to change the password.

Evaluation & Conclusion

Software Architectural Issues

Unfortunately there were software/system architectural issues with the project. The application being built is a Unity project, involving Unity GameObjects, and any originally planned BDD tests involving the UI aren't currently possible, for example checking button clicks perform the expected functionality. This is because the Unity GameObjects derive from MonoBehaviour, which is entirely handled within Unity. Meaning in Visual Studio it wasn't possible to conduct BDD unit tests on classes deriving from MonoBehaviour, and using behavioural driven development would prove to be problematic.

There is a package on the Unity Asset Store called "BDD Extension Framework for Unity Test Tools", however this wasn't compatible with recent versions of Unity, as the Unity Test Tools package has been deprecated. Therefore it wasn't possible to use BDD for UI interactions.

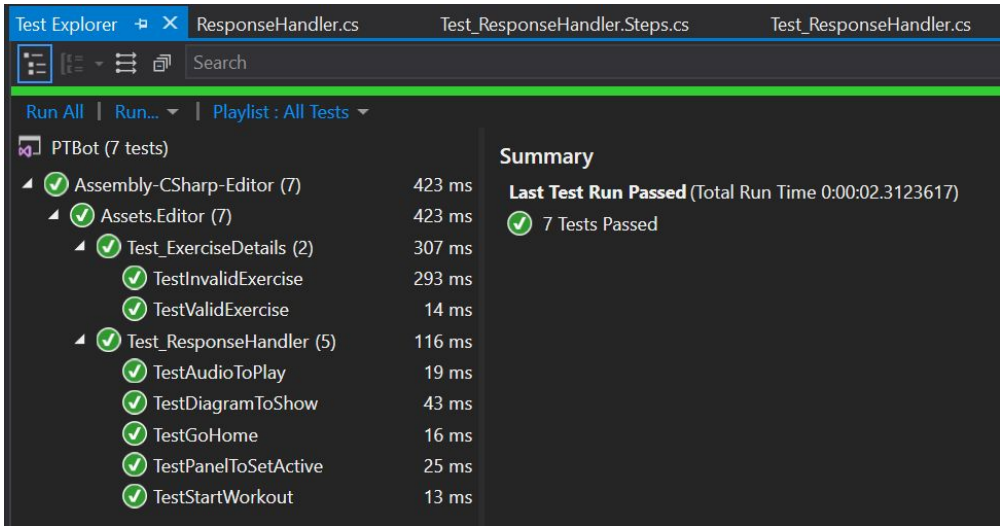
However, it is possible to use BDD for unit testing classes that don't involve any Unity objects, and by doing this the relevant logic can be encapsulated into its own class for specific testing, an example of this being the ResponseHandler class and test classes from *appendix 4, 5 and 6.* By separating logic into different classes, the code would become more manageable and maintainable rather than being in a single class.

Types of Testing

Different types of testing, combined when necessary, were conducted on the 2 classes shown above and in *Appendices 1,2,3,4,5 and 6*. The test cases were derived from Scenario Testing, and updated following Exploratory Testing when notes were created from experimenting with various input values for method parameters. Black-box testing was demonstrated through the use of decision tables, and was utilised with condition coverage, a technique for White-box Testing, and they were used for their suitability for these tests when compared with other black-box and white-box testing methods.

This testing strategy was beneficial to use, because through using and combining these types of tests, full coverage of both ExerciseDetails and ResponseHandler were achieved, ensuring all possible paths were testable and tested, with each test passing successfully, as shown from *figure 17 and Appendix 7*. Based on these results, it would be fair to say this test strategy was effective to use. Although, all projects are different, and the applied Q3 testing only covers a portion of the entire codebase, but it is a significant component.

Figure 17: Test explorer results.



There are further tests to be considered for the future, primarily from Quadrant 4, involving performance, load and security testing to see how the application can handle the maximum required number of users and review the security from external threats.

To conclude, this document has detailed the testing strategy, including evidence of different types of testing techniques and planned future testing, being applied to the Final Year Project created for the CS3010 module.

References

Esterle, L., 2019. *Black Box Test Design Strategy.* [Online]
Available at:
https://vle.aston.ac.uk/bbcswebdav/pid-1493840-dt-content-rid-9829259_1/courses/2018_CS3270/6_CS3270L04.pdf
[Accessed 01 04 2019].

Esterle, L., 2019. *Q3 Tests.* [Online]
Available at:
https://vle.aston.ac.uk/bbcswebdav/pid-1493855-dt-content-rid-9893221_1/courses/2018_CS3270/8_CS3270L06.pdf
[Accessed 01 04 2019].

Esterle, L., 2019. *Q4 Testing.* [Online]
Available at:
https://vle.aston.ac.uk/bbcswebdav/pid-1493845-dt-content-rid-9964359_1/courses/2018_CS3270/9_CS3270L07.pdf
[Accessed 01 04 2019].

Esterle, L., 2019. *Testing Strategy.* [Online]
Available at:
https://vle.aston.ac.uk/bbcswebdav/pid-1493824-dt-content-rid-10003244_1/courses/2018_CS3270/12_CS3270L10.pdf
[Accessed 01 04 2019].

Esterle, L., 2019. *White Box Test Design.* [Online]
Available at:
https://vle.aston.ac.uk/bbcswebdav/pid-1493798-dt-content-rid-9893223_1/courses/2018_CS3270/7_CS3270L05.pdf
[Accessed 01 04 2019].

Microsoft, 2018. Quickstart: Create a load test project
Available at:
https://docs.microsoft.com/en-us/visualstudio/test/quickstart-create-a-load-test-project?view=vs-2017
[Accessed 01 04 2019].

NuGet, 2019. LightBDD.Framework
Available at:
https://www.nuget.org/packages/LightBDD.Framework/
[Accessed 01 04 2019].

Appendices

Appendix 1: Test_ExerciseDetails.cs

```
using LightBDD.Framework;
using LightBDD.Framework.Scenarios;
using LightBDD.NUnit3;

[assembly: LightBddScopeAttribute]

namespace Assets.Editor
{
    public partial class Test_ExerciseDetails
    {
        [Label("Use an invalid exercise.")]
        [Scenario]
        public void TestInvalidExercise()
        {
            Runner.RunScenario(
                Given => CheckTheArrayIsNotNull(),
                When => WhenTheExerciseIsChecked(null),
                Then => TheItemDoesNotExistInTheArray());

            Runner.RunScenario(
                Given => CheckTheArrayIsNotNull(),
                When => WhenTheExerciseIsChecked(string.Empty),
                Then => TheItemDoesNotExistInTheArray());

            Runner.RunScenario(
                Given => CheckTheArrayIsNotNull(),
                When => WhenTheExerciseIsChecked("Fake input."),
                Then => TheItemDoesNotExistInTheArray());
        }

        [Label("Use a valid exercise.")]
        [Scenario]
        public void TestValidExercise()
        {
            Runner.RunScenario(
                Given => CheckTheArrayIsNotNull(),
                When => WhenTheExerciseIsChecked(" Air Bikes"),
                Then => TheItemExistsInTheArray(),
                And => TheExerciseDetailsAreFound());
```

```
            Runner.RunScenario(
                Given => CheckTheArrayIsNotNull(),
                When => WhenTheExerciseIsChecked(" Pullups"),
                Then => TheItemExistsInTheArray(),
                And => TheExerciseDetailsAreFound());
        }
    }
}
```

Appendix 2: Test_ExerciseDetails.steps.cs

```
using LightBDD.NUnit3;
using NUnit.Framework;

namespace Assets.Editor
{
    public partial class Test_ExerciseDetails : FeatureFixture
    {
        private ExerciseDetails exerciseDetails = new ExerciseDetails();
        private int indexToSearch;

        private void CheckTheArrayIsNotNull()
        {
            Assert.That(exerciseDetails.GetEntireArray() != null);
        }

        private void WhenTheExerciseIsChecked(string exerciseToCheck)
        {
            indexToSearch = exerciseDetails.GetArrayIndex(exerciseToCheck);
        }

        private void TheItemExistsInTheArray()
        {
            Assert.That(indexToSearch != -1);
        }

        private void TheExerciseDetailsAreFound()
        {
            Assert.That(!string.IsNullOrWhiteSpace(exerciseDetails.GetArrayValue(indexToSearch,
1)));
        }

        private void TheItemDoesNotExistInTheArray()
```

```
        {
            Assert.That(indexToSearch == -1);
        }
    }
}
```

Appendix 3: ExerciseDetails class.

```
public class ExerciseDetails
{
    private string[][] Exercises = new string[][]
    {
        // { name, url, description}
        // Full list of exercises to be added later
       new string[] { " Air Bikes", "https://videos.bodybuilding.com/video/mp4/28000/28901m.mp4",
"ToDo" },
       new string[] { " Pullups", "https://videos.bodybuilding.com/video/mp4/24000/25681m.mp4",
"ToDo" }
    };

    public string[][] GetEntireArray()
    {
        return Exercises;
    }

    public int GetArrayIndex(string value)
    {
        for (int i = 0; i < Exercises.GetLength(0); i++)
        {
            if (Exercises[i][0] == value) { return i; }
        }
        return -1;
    }

    public string GetArrayValue(int firstIndex, int secondIndex)
    {
        return Exercises[firstIndex][secondIndex];
    }
}
```

Appendix 4: Test_ResponseHandler.cs

```
using LightBDD.Framework;
using LightBDD.Framework.Scenarios;
```

```
using LightBDD.NUnit3;

namespace Assets.Editor
{
   [Label("FEAT-1")]
   public partial class Test_ResponseHandler
   {
      [Label("The string 'DefaultErrorResponse' should be returned from an invalid input, otherwise 'ChatbotResponse'")]
      [Scenario]
      public void TestAudioToPlay()
      {
         Runner.RunScenario(
            Given => TheResponseIsEmpty(string.Empty),
            When => TheResponseHandlerIsCalled(),
            Then => CheckTheAudioToPlay("DefaultErrorResponse"));

         Runner.RunScenario(
           Given => TheResponseIsEmpty(null),
           When => TheResponseHandlerIsCalled(),
           Then => CheckTheAudioToPlay("DefaultErrorResponse"));

         Runner.RunScenario(
            Given => TheResponseIsNotEmpty("Fake input."),
            When => TheResponseHandlerIsCalled(),
            Then => CheckTheAudioToPlay("ChatbotResponse"));
      }

      [Label("The assumes the user has chosen their equipment and the relevant panels will be returned and set active")]
      [Scenario]
      public void TestDiagramToShow()
      {
         Runner.RunScenario(
            Given => TheResponseIsNotEmpty("Arms workout"),
            When => TheResponseHandlerIsCalled(),
            Then => CheckTheDiagramToShow("ArmsDiagram"));

         Runner.RunScenario(
            Given => TheResponseIsNotEmpty("Arms workout"),
            When => TheResponseHandlerIsCalled(),
            Then => CheckTheDiagramToShow("ArmsDiagram"));
```

```
            Runner.RunScenario(
                Given => TheResponseIsNotEmpty("Back workout"),
                When => TheResponseHandlerIsCalled(),
                Then => CheckTheDiagramToShow("BackDiagram"));

            Runner.RunScenario(
                Given => TheResponseIsNotEmpty("Chest workout"),
                When => TheResponseHandlerIsCalled(),
                Then => CheckTheDiagramToShow("ChestDiagram"));

            Runner.RunScenario(
                Given => TheResponseIsNotEmpty("Core workout"),
                When => TheResponseHandlerIsCalled(),
                Then => CheckTheDiagramToShow("CoreDiagram"));

            Runner.RunScenario(
                Given => TheResponseIsNotEmpty("Legs workout"),
                When => TheResponseHandlerIsCalled(),
                Then => CheckTheDiagramToShow("LegsDiagram"));

            Runner.RunScenario(
                Given => TheResponseIsNotEmpty("Fake input."),
                When => TheResponseHandlerIsCalled(),
                Then => CheckTheDiagramToShow(string.Empty));
        }

        [Label("The returned value for goHome should be true when the response contains 'Bye',
and false if not.")]
        [Scenario]
        public void TestGoHome()
        {
            Runner.RunScenario(
                Given => TheResponseIsNotEmpty("Bye"),
                When => TheResponseHandlerIsCalled(),
                Then => CheckGoHome(true));

            Runner.RunScenario(
                Given => TheResponseIsNotEmpty("Fake input."),
                When => TheResponseHandlerIsCalled(),
                Then => CheckGoHome(false));
        }

        [Label("The correct panel should be displayed based on the chatbot response.")]
```

```
        [Scenario]
        public void TestPanelToSetActive()
        {
            Runner.RunScenario(
                Given => TheResponseIsNotEmpty("Fake input."),
                When => TheResponseHandlerIsCalled(),
                Then => CheckThePanelShouldBeSetActive(""));

            Runner.RunScenario(
                Given => TheResponseIsNotEmpty("which area would you like to train today"),
                When => TheResponseHandlerIsCalled(),
                Then => CheckThePanelShouldBeSetActive("AreasToTrainPanel"));

            Runner.RunScenario(
                Given => TheResponseIsNotEmpty("- "),
                When => TheResponseHandlerIsCalled(),
                Then => CheckThePanelShouldBeSetActive("ExercisesPanel"));

            Runner.RunScenario(
                Given => TheResponseIsNotEmpty("Ok. Let's do the workout."),
                When => TheResponseHandlerIsCalled(),
                Then => CheckThePanelShouldBeSetActive("StartWorkoutPanel"));
        }

        [Label("The returned value for startWorkout should be true when the response contains
'Ok. Let's do the workout.'")]
        [Scenario]
        public void TestStartWorkout()
        {
            Runner.RunScenario(
                Given => TheResponseIsNotEmpty("Fake input."),
                When => TheResponseHandlerIsCalled(),
                Then => CheckIfTheWorkoutShouldStart());

            Runner.RunScenario(
                Given => TheResponseIsNotEmpty("Ok. Let's do the workout."),
                When => TheResponseHandlerIsCalled(),
                Then => CheckIfTheWorkoutShouldStart());
        }
    }
}
```

Appendix 5: Test_ResponseHandler.steps.cs

```
using LightBDD.NUnit3;
using NUnit.Framework;

namespace Assets.Editor
{
    public partial class Test_ResponseHandler : FeatureFixture
    {
        private ResponseHandler responseHandler = new ResponseHandler();
        string chatbotResponse = "";

        private void TheResponseIsEmpty(string chatbotResponse)
        {
            this.chatbotResponse = chatbotResponse;
            Assert.That(string.IsNullOrWhiteSpace(chatbotResponse));
        }

        private void TheResponseIsNotEmpty(string chatbotResponse)
        {
            this.chatbotResponse = chatbotResponse;
            Assert.That(!string.IsNullOrWhiteSpace(chatbotResponse));
        }

        private void TheResponseHandlerIsCalled()
        {
            responseHandler.HandleResponse(chatbotResponse);
        }

        private void CheckTheAudioToPlay(string audioToPlay)
        {
            Assert.That(responseHandler.CheckAudioToPlay() == audioToPlay);
        }

        private void CheckGoHome(bool isTrue)
        {
            Assert.That(responseHandler.CheckGoHome() == isTrue);
        }

        private void CheckIfTheWorkoutShouldStart()
        {
            if (responseHandler.CheckStartWorkout() == true)
            {
```

```
Assert.That(responseHandler.CheckPanelsToSetActive().Contains("StartWorkoutPanel"));
            }
            else
            {
                Assert.That(responseHandler.CheckStartWorkout() == false);
            }
        }

        private void CheckThePanelShouldBeSetActive(string panel)
        {
            if (responseHandler.CheckPanelsToSetActive().Count > 0)
            {
                Assert.That(responseHandler.CheckPanelsToSetActive().Contains(panel));
            }
            else
            {
                Assert.That(responseHandler.CheckPanelsToSetActive().Count <= 0);
            }
        }

        private void CheckTheDiagramToShow(string diagramToShow)
        {
            Assert.That(diagramToShow == responseHandler.CheckDiagramToShow());
        }
    }
}
```

Appendix 6: ResponseHandler class.

```
using System.Collections;

public class ResponseHandler
{
    private string diagramToShow, audioToPlay;
    private ArrayList panelsToSetActive;
    private bool goHome, startWorkout, getExerciseDetails;

    public void HandleResponse(string chatbotResponse)
    {
        goHome = false; startWorkout = false; getExerciseDetails = false;
        diagramToShow = "";
        panelsToSetActive = new ArrayList();
```

```
        if (string.IsNullOrWhiteSpace(chatbotResponse))
        {
            audioToPlay = "DefaultErrorResponse";
        }
        else
        {
            if ((!chatbotResponse.Contains("Bye")) && (!chatbotResponse.Contains("Thanks for
your time"))
                && (!chatbotResponse.Contains("No worries, take care")) &&
(!chatbotResponse.Contains("Thanks for using JimBot")))
            {
                if (chatbotResponse.Contains("Ok. Let's do the workout."))
                {
                    panelsToSetActive.Add("StartWorkoutPanel");
                    startWorkout = true;
                }
                else
                {
                    if (chatbotResponse.ToLower().Contains("which area would you like to train
today"))
                    {
                        panelsToSetActive.Add("AreasToTrainPanel");
                        panelsToSetActive.Add("scrollArea");
                    }

                    if (chatbotResponse.Contains("Arms")) { diagramToShow = "ArmsDiagram"; }
                    else if (chatbotResponse.Contains("Back")) { diagramToShow = "BackDiagram"; }
                    else if (chatbotResponse.Contains("Chest")) { diagramToShow = "ChestDiagram"; }
                    else if (chatbotResponse.Contains("Core")) { diagramToShow = "CoreDiagram"; }
                    else if (chatbotResponse.Contains("Legs")) { diagramToShow = "LegsDiagram"; }
                    else if (chatbotResponse.Contains("No Equipment")) { diagramToShow =
"CoreDiagram"; }

                    if (chatbotResponse.ToLower().Contains("- "))
                    {
                        panelsToSetActive.Add("ExercisesPanel");
                        getExerciseDetails = true;
                    }
                }
            }
            else
            {
                goHome = true;
```

```
            }
            audioToPlay = "ChatbotResponse";
        }
    }

    public ArrayList CheckPanelsToSetActive()
    {
        return panelsToSetActive;
    }

    public bool CheckGetExerciseDetails()
    {
        return getExerciseDetails;
    }

    public bool CheckGoHome()
    {
        return goHome;
    }

    public bool CheckStartWorkout()
    {
        return startWorkout;
    }

    public string CheckAudioToPlay()
    {
        return audioToPlay;
    }

    public string CheckDiagramToShow()
    {
        return diagramToShow;
    }
}
```

Appendix 7: NUnit Unit Test output results.

```
1> SCENARIO: [Use an invalid exercise.] TestInvalidExercise
1>   STEP 1/3: GIVEN CheckTheArrayIsNotNull...
1>   STEP 1/3: GIVEN CheckTheArrayIsNotNull (Passed after 25ms)
1>   STEP 2/3: WHEN WhenTheExerciseIsChecked [exerciseToCheck: "<null>"]...
1>   STEP 2/3: WHEN WhenTheExerciseIsChecked [exerciseToCheck: "<null>"] (Passed after 1ms)
1>   STEP 3/3: THEN TheItemDoesNotExistInTheArray...
1>   STEP 3/3: THEN TheItemDoesNotExistInTheArray (Passed after <1ms)
1>   SCENARIO RESULT: Passed after 184ms
2> SCENARIO: [Use an invalid exercise.] TestInvalidExercise
2>   STEP 1/3: GIVEN CheckTheArrayIsNotNull...
2>   STEP 1/3: GIVEN CheckTheArrayIsNotNull (Passed after <1ms)
2>   STEP 2/3: WHEN WhenTheExerciseIsChecked [exerciseToCheck: ""]...
2>   STEP 2/3: WHEN WhenTheExerciseIsChecked [exerciseToCheck: ""] (Passed after <1ms)
2>   STEP 3/3: THEN TheItemDoesNotExistInTheArray...
2>   STEP 3/3: THEN TheItemDoesNotExistInTheArray (Passed after <1ms)
2>   SCENARIO RESULT: Passed after 12ms
3> SCENARIO: [Use an invalid exercise.] TestInvalidExercise
3>   STEP 1/3: GIVEN CheckTheArrayIsNotNull...
3>   STEP 1/3: GIVEN CheckTheArrayIsNotNull (Passed after <1ms)
3>   STEP 2/3: WHEN WhenTheExerciseIsChecked [exerciseToCheck: "Fake input."]...
3>   STEP 2/3: WHEN WhenTheExerciseIsChecked [exerciseToCheck: "Fake input."] (Passed after <1ms)
3>   STEP 3/3: THEN TheItemDoesNotExistInTheArray...
3>   STEP 3/3: THEN TheItemDoesNotExistInTheArray (Passed after <1ms)
```

```
3>   SCENARIO RESULT: Passed after 3ms
4> SCENARIO: [Use a valid exercise.] TestValidExercise
4>   STEP 1/4: GIVEN CheckTheArrayIsNotNull...
4>   STEP 1/4: GIVEN CheckTheArrayIsNotNull (Passed after <1ms)
4>   STEP 2/4: WHEN WhenTheExerciseIsChecked [exerciseToCheck: " Air Bikes"]...
4>   STEP 2/4: WHEN WhenTheExerciseIsChecked [exerciseToCheck: " Air Bikes"] (Passed after <1ms)
4>   STEP 3/4: THEN TheItemExistsInTheArray...
4>   STEP 3/4: THEN TheItemExistsInTheArray (Passed after <1ms)
4>   STEP 4/4: AND TheExerciseDetailsAreFound...
4>   STEP 4/4: AND TheExerciseDetailsAreFound (Passed after <1ms)
4>   SCENARIO RESULT: Passed after 6ms
5> SCENARIO: [Use a valid exercise.] TestValidExercise
5>   STEP 1/4: GIVEN CheckTheArrayIsNotNull...
5>   STEP 1/4: GIVEN CheckTheArrayIsNotNull (Passed after <1ms)
5>   STEP 2/4: WHEN WhenTheExerciseIsChecked [exerciseToCheck: " Pullups"]...
5>   STEP 2/4: WHEN WhenTheExerciseIsChecked [exerciseToCheck: " Pullups"] (Passed after <1ms)
5>   STEP 3/4: THEN TheItemExistsInTheArray...
5>   STEP 3/4: THEN TheItemExistsInTheArray (Passed after <1ms)
5>   STEP 4/4: AND TheExerciseDetailsAreFound...
5>   STEP 4/4: AND TheExerciseDetailsAreFound (Passed after <1ms)
5>   SCENARIO RESULT: Passed after 4ms
 > FEATURE FINISHED: Test ExerciseDetails
 > FEATURE: [FEAT-1] Test ResponseHandler
```

```
6> SCENARIO: [The string 'DefaultErrorResponse' should be returned from an invalid input, otherwise 'ChatbotResponse'] TestAudioToPlay
6>   STEP 1/3: GIVEN TheResponseIsEmpty [chatbotResponse: ""]...
6>   STEP 1/3: GIVEN TheResponseIsEmpty [chatbotResponse: ""] (Passed after <1ms)
6>   STEP 2/3: WHEN TheResponseHandlerIsCalled...
6>   STEP 2/3: WHEN TheResponseHandlerIsCalled (Passed after 1ms)
6>   STEP 3/3: THEN CheckTheAudioToPlay [audioToPlay: "DefaultErrorResponse"]...
6>   STEP 3/3: THEN CheckTheAudioToPlay [audioToPlay: "DefaultErrorResponse"] (Passed after <1ms)
6>   SCENARIO RESULT: Passed after 9ms
7> SCENARIO: [The string 'DefaultErrorResponse' should be returned from an invalid input, otherwise 'ChatbotResponse'] TestAudioToPlay
7>   STEP 1/3: GIVEN TheResponseIsEmpty [chatbotResponse: "<null>"]...
7>   STEP 1/3: GIVEN TheResponseIsEmpty [chatbotResponse: "<null>"] (Passed after <1ms)
7>   STEP 2/3: WHEN TheResponseHandlerIsCalled...
7>   STEP 2/3: WHEN TheResponseHandlerIsCalled (Passed after <1ms)
7>   STEP 3/3: THEN CheckTheAudioToPlay [audioToPlay: "DefaultErrorResponse"]...
7>   STEP 3/3: THEN CheckTheAudioToPlay [audioToPlay: "DefaultErrorResponse"] (Passed after <1ms)
7>   SCENARIO RESULT: Passed after 3ms
8> SCENARIO: [The string 'DefaultErrorResponse' should be returned from an invalid input, otherwise 'ChatbotResponse'] TestAudioToPlay
8>   STEP 1/3: GIVEN TheResponseIsNotEmpty [chatbotResponse: "Fake input."]...
8>   STEP 1/3: GIVEN TheResponseIsNotEmpty [chatbotResponse: "Fake input."] (Passed after <1ms)
8>   STEP 2/3: WHEN TheResponseHandlerIsCalled...
8>   STEP 2/3: WHEN TheResponseHandlerIsCalled (Passed after <1ms)
8>   STEP 3/3: THEN CheckTheAudioToPlay [audioToPlay: "ChatbotResponse"]...
8>   STEP 3/3: THEN CheckTheAudioToPlay [audioToPlay: "ChatbotResponse"] (Passed after <1ms)
```

```
14>   STEP 3/3: THEN CheckTheDiagramToShow [diagramToShow: "LegsDiagram"]...
14>   STEP 3/3: THEN CheckTheDiagramToShow [diagramToShow: "LegsDiagram"] (Passed after <1ms)
14>   SCENARIO RESULT: Passed after 3ms
15> SCENARIO: [The assumes the user has chosen their equipment and the relevant panels will be returned and set active] TestDiagramToShow
15>   STEP 1/3: GIVEN TheResponseIsNotEmpty [chatbotResponse: "Fake input."]...
15>   STEP 1/3: GIVEN TheResponseIsNotEmpty [chatbotResponse: "Fake input."] (Passed after <1ms)
15>   STEP 2/3: WHEN TheResponseHandlerIsCalled...
15>   STEP 2/3: WHEN TheResponseHandlerIsCalled (Passed after <1ms)
15>   STEP 3/3: THEN CheckTheDiagramToShow [diagramToShow: ""]...
15>   STEP 3/3: THEN CheckTheDiagramToShow [diagramToShow: ""] (Passed after <1ms)
15>   SCENARIO RESULT: Passed after 6ms
16> SCENARIO: [The returned value for goHome should be true when the response contains 'Bye', and false if not.] TestGoHome
16>   STEP 1/3: GIVEN TheResponseIsNotEmpty [chatbotResponse: "Bye"]...
16>   STEP 1/3: GIVEN TheResponseIsNotEmpty [chatbotResponse: "Bye"] (Passed after <1ms)
16>   STEP 2/3: WHEN TheResponseHandlerIsCalled...
16>   STEP 2/3: WHEN TheResponseHandlerIsCalled (Passed after <1ms)
16>   STEP 3/3: THEN CheckGoHome [isTrue: "True"]...
16>   STEP 3/3: THEN CheckGoHome [isTrue: "True"] (Passed after <1ms)
16>   SCENARIO RESULT: Passed after 8ms
17> SCENARIO: [The returned value for goHome should be true when the response contains 'Bye', and false if not.] TestGoHome
17>   STEP 1/3: GIVEN TheResponseIsNotEmpty [chatbotResponse: "Fake input."]...
17>   STEP 1/3: GIVEN TheResponseIsNotEmpty [chatbotResponse: "Fake input."] (Passed after <1ms)
17>   STEP 2/3: WHEN TheResponseHandlerIsCalled...
```

```
17>   STEP 2/3: WHEN TheResponseHandlerIsCalled (Passed after <1ms)
17>   STEP 3/3: THEN CheckGoHome [isTrue: "False"]...
17>   STEP 3/3: THEN CheckGoHome [isTrue: "False"] (Passed after <1ms)
17>   SCENARIO RESULT: Passed after 5ms
18> SCENARIO: [The correct panel should be displayed based on the chatbot response.] TestPanelToSetActive
18>   STEP 1/3: GIVEN TheResponseIsNotEmpty [chatbotResponse: "Fake input."]...
18>   STEP 1/3: GIVEN TheResponseIsNotEmpty [chatbotResponse: "Fake input."] (Passed after <1ms)
18>   STEP 2/3: WHEN TheResponseHandlerIsCalled...
18>   STEP 2/3: WHEN TheResponseHandlerIsCalled (Passed after <1ms)
18>   STEP 3/3: THEN CheckThePanelShouldBeSetActive [panel: ""]...
18>   STEP 3/3: THEN CheckThePanelShouldBeSetActive [panel: ""] (Passed after <1ms)
18>   SCENARIO RESULT: Passed after 5ms
19> SCENARIO: [The correct panel should be displayed based on the chatbot response.] TestPanelToSetActive
19>   STEP 1/3: GIVEN TheResponseIsNotEmpty [chatbotResponse: "which area would you like to train today"]...
19>   STEP 1/3: GIVEN TheResponseIsNotEmpty [chatbotResponse: "which area would you like to train today"] (Passed after <1ms)
19>   STEP 2/3: WHEN TheResponseHandlerIsCalled...
19>   STEP 2/3: WHEN TheResponseHandlerIsCalled (Passed after <1ms)
19>   STEP 3/3: THEN CheckThePanelShouldBeSetActive [panel: "AreasToTrainPanel"]...
19>   STEP 3/3: THEN CheckThePanelShouldBeSetActive [panel: "AreasToTrainPanel"] (Passed after <1ms)
19>   SCENARIO RESULT: Passed after 5ms
20> SCENARIO: [The correct panel should be displayed based on the chatbot response.] TestPanelToSetActive
20>   STEP 1/3: GIVEN TheResponseIsNotEmpty [chatbotResponse: "- "]...
20>   STEP 1/3: GIVEN TheResponseIsNotEmpty [chatbotResponse: "- "] (Passed after <1ms)
```

```
20>   STEP 2/3: WHEN TheResponseHandlerIsCalled...
20>   STEP 2/3: WHEN TheResponseHandlerIsCalled (Passed after <1ms)
20>   STEP 3/3: THEN CheckThePanelShouldBeSetActive [panel: "ExercisesPanel"]...
20>   STEP 3/3: THEN CheckThePanelShouldBeSetActive [panel: "ExercisesPanel"] (Passed after <1ms)
20>   SCENARIO RESULT: Passed after 5ms
21> SCENARIO: [The correct panel should be displayed based on the chatbot response.] TestPanelToSetActive
21>   STEP 1/3: GIVEN TheResponseIsNotEmpty [chatbotResponse: "Ok. Let's do the workout."]...
21>   STEP 1/3: GIVEN TheResponseIsNotEmpty [chatbotResponse: "Ok. Let's do the workout."] (Passed after <1ms)
21>   STEP 2/3: WHEN TheResponseHandlerIsCalled...
21>   STEP 2/3: WHEN TheResponseHandlerIsCalled (Passed after <1ms)
21>   STEP 3/3: THEN CheckThePanelShouldBeSetActive [panel: "StartWorkoutPanel"]...
21>   STEP 3/3: THEN CheckThePanelShouldBeSetActive [panel: "StartWorkoutPanel"] (Passed after <1ms)
21>   SCENARIO RESULT: Passed after 5ms
22> SCENARIO: [The returned value for startWorkout should be true when the response contains 'Ok. Let's do the workout.'] TestStartWorkout
22>   STEP 1/3: GIVEN TheResponseIsNotEmpty [chatbotResponse: "Fake input."]...
22>   STEP 1/3: GIVEN TheResponseIsNotEmpty [chatbotResponse: "Fake input."] (Passed after <1ms)
22>   STEP 2/3: WHEN TheResponseHandlerIsCalled...
22>   STEP 2/3: WHEN TheResponseHandlerIsCalled (Passed after <1ms)
22>   STEP 3/3: THEN CheckIfTheWorkoutShouldStart...
22>   STEP 3/3: THEN CheckIfTheWorkoutShouldStart (Passed after <1ms)
22>   SCENARIO RESULT: Passed after 5ms
23> SCENARIO: [The returned value for startWorkout should be true when the response contains 'Ok. Let's do the workout.'] TestStartWorkout
23>   STEP 1/3: GIVEN TheResponseIsNotEmpty [chatbotResponse: "Ok. Let's do the workout."]...
```

```
23>   STEP 1/3: GIVEN TheResponseIsNotEmpty [chatbotResponse: "Ok. Let's do the workout."] (Passed after <1ms)
23>   STEP 2/3: WHEN TheResponseHandlerIsCalled...
23>   STEP 2/3: WHEN TheResponseHandlerIsCalled (Passed after <1ms)
23>   STEP 3/3: THEN CheckIfTheWorkoutShouldStart...
23>   STEP 3/3: THEN CheckIfTheWorkoutShouldStart (Passed after <1ms)
23>   SCENARIO RESULT: Passed after 5ms
  > FEATURE FINISHED: Test ResponseHandler
13.0.0: Test execution complete
```