



# Testing Documentation

Project Title: Intendi

Karl Duignan: 16105982

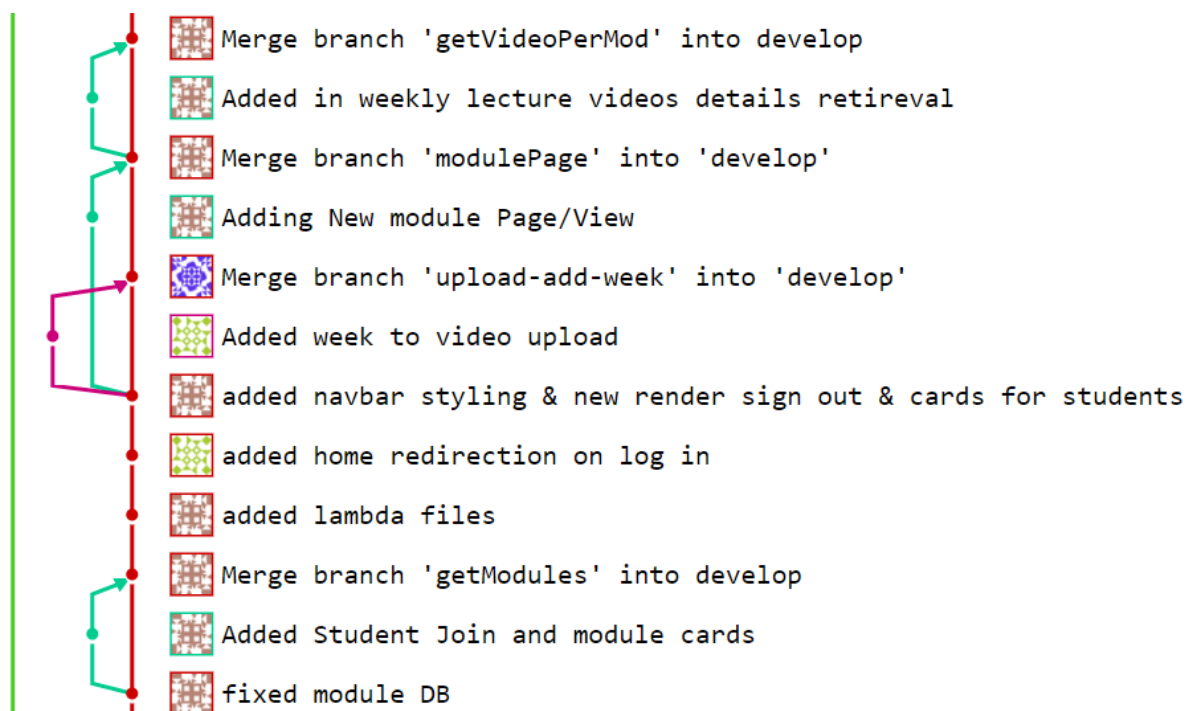
Luke Hebblethwaite: 17425212

Supervisor: Dr Michael Scriney

Date created: 02/05/2021

## 1.0 Git + Ci/CD usage

At the beginning of our development, we decided that the best workflow to use would be a feature branch workflow. This would ensure that we could work on a particular feature without disrupting the main codebase. Our Git workflow was as follows: when development began, we created the develop branch from the master branch. This would be our main development branch and from here, we would further branch out for each feature. When the feature was completed, a pull request would be opened and we would review it together. If there were no issues with the code, the branch would then be merged with develop and deleted. In the screenshot below, you can see that feature branches were made, commits were made to them and then when the feature was completed, the branch was merged into develop.



We also decided it would be good to have a CI/CD pipeline to make sure any merges/commits are fully working as expected. The CI/CD pipeline involved 3 phases, build, frontend tests and backend tests.

The build phase simply involved running an npm build and awaiting a successful build response. Once this job succeeded it then moved onto running all the frontend Jest tests. This again ran all our pre-defined frontend component Jest tests and will only pass onto the final phase once this has succeeded. This made sure no components within the React application were not working as expected when a new commit was made. The final phase is to run the backend Pytest to ensure all the Lambda functions being pushed are running as expected and that none of them are broken when the changes are deployed.

```


! .gitlab-ci.yml
1  stages:
2    - build
3    - frontendtest
4    - backendtest
5
6  build:
7    stage: build
8    script:
9      - cd ./src/intendi
10     - echo "Building deploy package"
11     - npm install
12     - npm build
13     - echo "Build successful"
14  frontendtest:
15    stage: frontendtest
16    image: node
17    script:
18      - cd ./src/intendi
19      - echo "Testing App"
20      - npm install
21      - CI=true npm test a
22      - echo "Test successfully!"
23
24  backendtest:
25    stage: backendtest
26    image: python
27    script:
28      - cd ./src/intendi/backendTests
29      - echo "Testing Backend"
30      - pip install -U pytest
31      - pip install moto
32      - CI=true pytest
33      - echo "Test successfully!"

```

Once these are all passed we can confirm the whole pipeline has succeeded.


For example, below, we can see when the branch “deleteModule” was being merged, the pipeline was run and as we can see that all phases passed therefore leading to a successful push.




passed

Pipeline #28108 triggered 36 minutes ago by  Karl Duignan

## Added delete Module Lambda

3 jobs for `deleteModule` in 6 minutes and 9 seconds (queued for 1 second)

 latest


 8d9e9487  


Pipeline Jobs 3


Build


Frontendtest


Backendtest


 build



 frontendtest



 backendtest



## 1.1 Unit Testing

For unit testing the backend Lambda function, we used a combination of Moto and Pytest. Moto is a library that allows your tests to mock AWS services easily. This was essential for us as almost all of our Lambda functions involve working with AWS DynamoDB. We, therefore, needed a way to test these without actually making any changes to the live tables.

Pytest is a full-featured Python testing tool and seeing as our Lambda functions were written in Python, this was exactly what we needed. The tests involved firstly creating any mock version of the DynamoDB tables that we needed for the Lambda function we were testing.

Once created, we also put any data/items into the mock table to be able to treat the test as close to real-world conditions as possible. Once this was all done, we were then able to import the functions from within the Lambda that we wanted to test. We then could pass the mock table and any other arguments needed by the Lambda function.

We then performed asserts to check that all the expected output was returned. This included ensuring the whole function ran and returned a response to also scanning and checking the mock table to ensure the required data updated, deleted or was inserted etc.

An example of this is testing the `ratingSubmit` Lambda function. This function requires updating values within the `VideoDetails` Table, in particular the video rating and total votes.

We firstly mocked the table and input fake data:

```

table_name = 'VideoDetails-intendinew'
dynamodb = boto3.resource('dynamodb', 'eu-west-1')

# create DB
dynamodb.create_table(
    TableName=table_name,
    KeySchema=[{'AttributeName': 'VideoID', 'KeyType': 'HASH'}],
    AttributeDefinitions=[{'AttributeName': 'VideoID', 'AttributeType': 'S'}],
    ProvisionedThroughput={'ReadCapacityUnits': 5, 'WriteCapacityUnits': 5}
)

video_details_table = dynamodb.Table(table_name)
video_details_table.put_item(
    Item={
        'VideoID': 'Video1234',
        'AverageRating': 0,
        'FeedbackComments': [],
        'TotalVotes': 0
    })

```

We could then test the function with different test case scenarios, particularly making sure the comments add to the FeedbackComments list successfully and that if there are two ratings, the average is calculated correctly. As seen below, we see one call to the Lambda and the checking of if it was all updated correctly. We then perform another call to the Lambda function to ensure that it updates the table as expected.

```

getsubmitratingresponse = feedbacksubmit(video_details_table, "Video1234",3.5,"Great Video")

response = video_details_table.scan()

assert getsubmitratingresponse == 'Successfully submitted Video Rating'
assert response['Items'][0]['AverageRating'] == 3.5
assert response['Items'][0]['TotalVotes'] == 1
assert response['Items'][0]['FeedbackComments'] == ['Great Video']

getsubmitratingresponse = feedbacksubmit(video_details_table, "Video1234",1,"bad Video")

response = video_details_table.scan()

assert getsubmitratingresponse == 'Successfully submitted Video Rating'
assert response['Items'][0]['AverageRating'] == 2.25
assert response['Items'][0]['TotalVotes'] == 2
assert response['Items'][0]['FeedbackComments'] == ['Great Video','bad Video']

```

If it does, all these tests will pass or else it will return an error message displaying which assert was different than expected and how it was different.

```

PS C:\Users\karld\Desktop\2021-ca400-duinak2-hebblel2\src\intendi\backendTests> pytest ratingsubmit_test.py
===== test session starts =====
platform win32 -- Python 3.8.0, pytest-6.2.3, py-1.10.0, pluggy-0.13.1
rootdir: C:\Users\karld\Desktop\2021-ca400-duinak2-hebblel2\src\intendi\backendTests
collected 1 item

ratingsubmit_test.py . [100%]

===== 1 passed in 0.78s =====

```

Once all the Lambda functions tests were covered, we were able to run all the tests collectively rather than separately, and this then covered all the backend unit testing in terms of Lambda functions.

```
PS C:\Users\karld\Desktop\2021-ca400-duignak2-hebble12\src\intendi\backendTests> pytest
===== test session starts =====
platform win32 -- Python 3.8.0, pytest-6.2.3, py-1.10.0, pluggy-0.13.1
rootdir: C:\Users\karld\Desktop\2021-ca400-duignak2-hebble12\src\intendi\backendTests
collected 14 items

addmodulepassword_test.py . [ 7%]
createmodule_test.py . [ 14%]
getusermodule_test.py . [ 21%]
moduledeletemodifiable_test.py . [ 28%]
moduledeleteuser_test.py . [ 35%]
moduledeletevideotable_test.py . [ 42%]
ratingsubmit_test.py . [ 50%]
sendwatchtime_test.py .. [ 64%]
studentjoin_test.py . [ 71%]
studentwatchcount_test.py .. [ 85%]
upload_test.py . [ 92%]
videodelete_test.py . [100%]

===== 14 passed in 2.19s =====
```

## 1.2 UI Testing

After doing some research on React testing, we decided to use Jest and Enzyme. Jest is a JavaScript testing framework. Jest acts as a test runner, assertion library and mocking library. Enzyme adds additional methods for rendering a component, finding elements, and interacting with elements. We performed 38 tests over 13 components.

```
Test Suites: 13 passed, 13 total
Tests:       38 passed, 38 total
Snapshots:   13 passed, 13 total
Time:        11.08 s
Ran all test suites.
```

Below is a snippet of code from CreateModule.test.js which is the test file for CreateModule.js. “it” is an alias of test, what is written in the parentheses is the name of the test. In the first test, we perform a shallow rendering test on the CreateModule component. Shallow rendering is useful to test a component as a unit and to ensure that our tests aren't indirectly asserting the behaviour of child components. In the second test, we are ensuring that our states are set correctly when the component is rendered. In the third test, we are performing a snapshot test. Snapshot tests are a very useful tool whenever you want to make sure your UI does not change unexpectedly. A snapshot test renders a UI component, takes a snapshot, then compares it to a reference snapshot file stored alongside the test. The test will fail if the two snapshots do not match: either the change is unexpected, or the reference snapshot needs to be updated to the new version of the UI component.

```

it("CreateModule.js renders without crashing", () => {
  shallow(<CreateModule/>);
});

it('states should be undefined', () => {
  const wrapper = shallow(<CreateModule/>);
  expect(wrapper.state('moduleCde')).toEqual(undefined);
  expect(wrapper.state('modulePass')).toEqual(undefined);
});

it("Snapshot test", () => {
  const wrapper = shallow(<CreateModule/>);
  expect(wrapper).toMatchSnapshot();
});

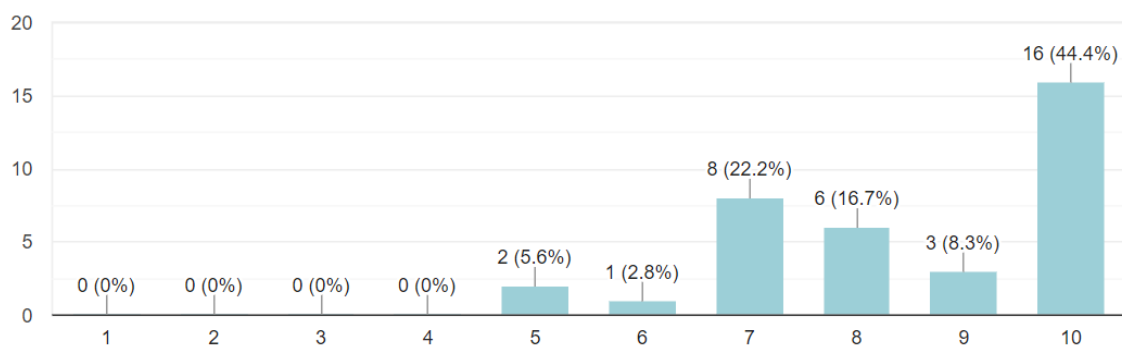
```

## 1.3 User Testing

We conducted user testing in order to ensure usability. We decided to create two user feedback forms. One that contained screenshots of our application and questions about those screenshots and another that was to be taken by members of our household that got to actually use our application. First, we will discuss the results of the form that contained screenshots and was taken by people who did not live in our household. Overall, we received 36 responses.

How beneficial do you think this would be for lecturers?

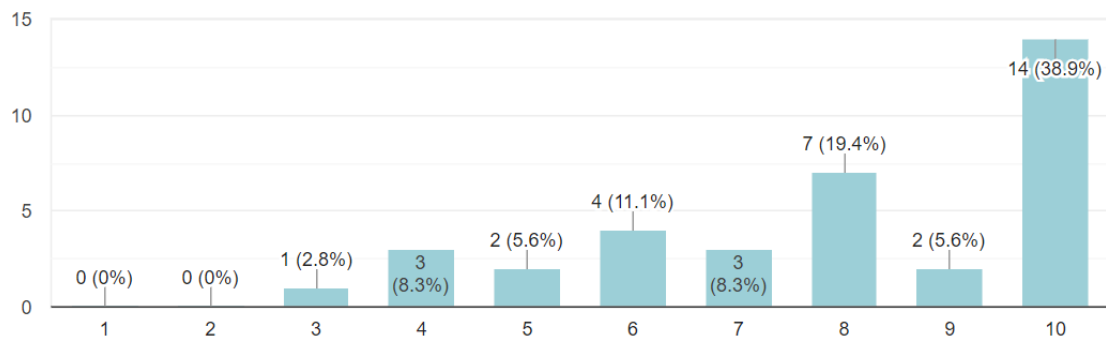
36 responses



As you can see from the above screenshot, the majority of people felt that our application would be very beneficial for lecturers.

How beneficial do you think this would be for students?

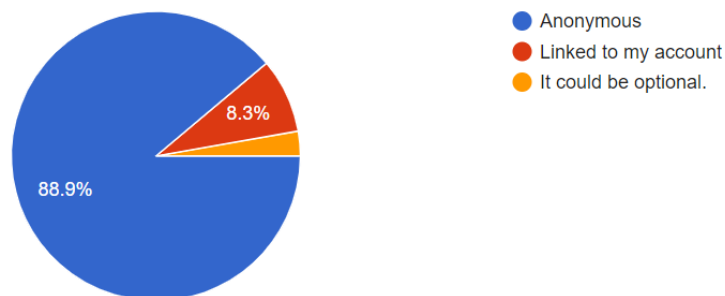
36 responses



The majority of respondents also thought that our application would be highly beneficial to students.

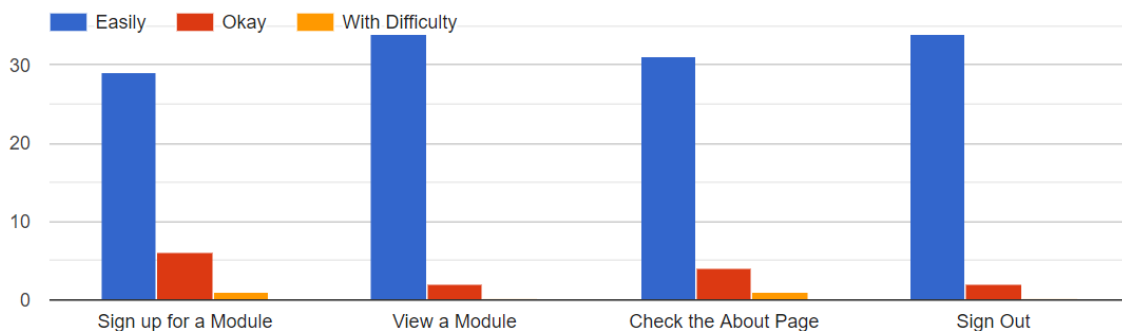
Would you prefer for the data to be anonymous or would you like it to be linked to your student account?

36 responses



As you can see from the screenshot above, the majority of students would prefer that the facial analysis data was kept anonymous which was great for us as that was our goal with the project at the very beginning.

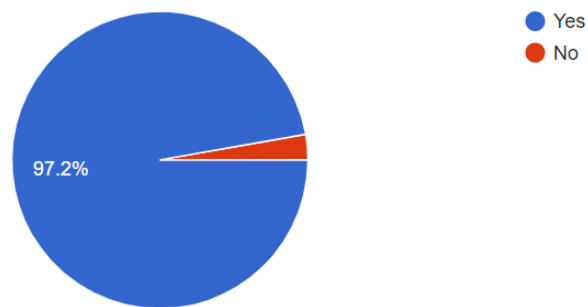
Judging from the image, how easy do you think it would be to...





Does the Module Home Page seem like it's easy to find a video you wish to watch?

36 responses

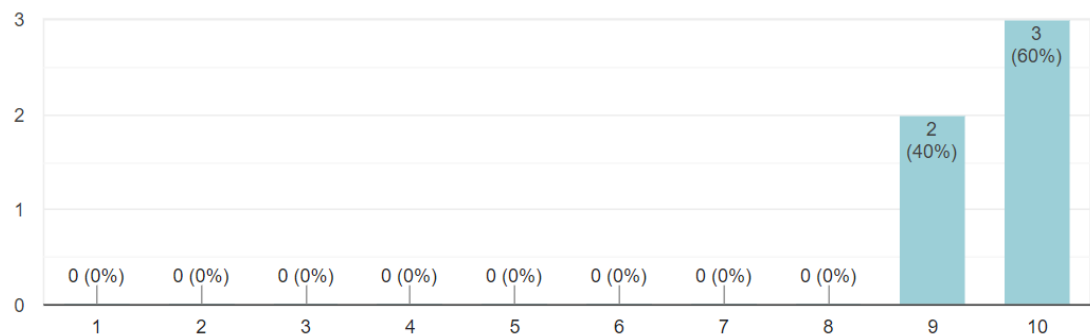


Here we can see that the majority of respondents found our application easy to use and navigate.

Next, we will show and discuss from our second form. This form was filled in by members of our households. Overall, we had five members of our households participate.

How would you rate the overall experience of the website?

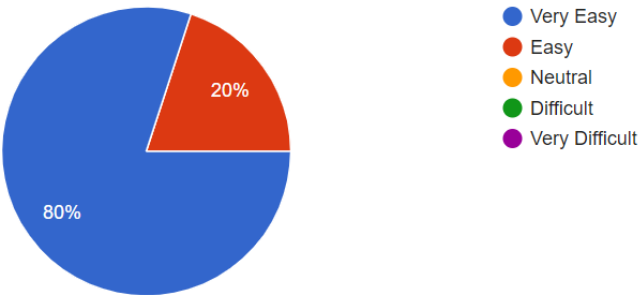
5 responses



As you can see, respondents reported a very positive experience of our application with all testers scoring 9 and above.

How easy was the text to read?

5 responses



How easy was the website to navigate?

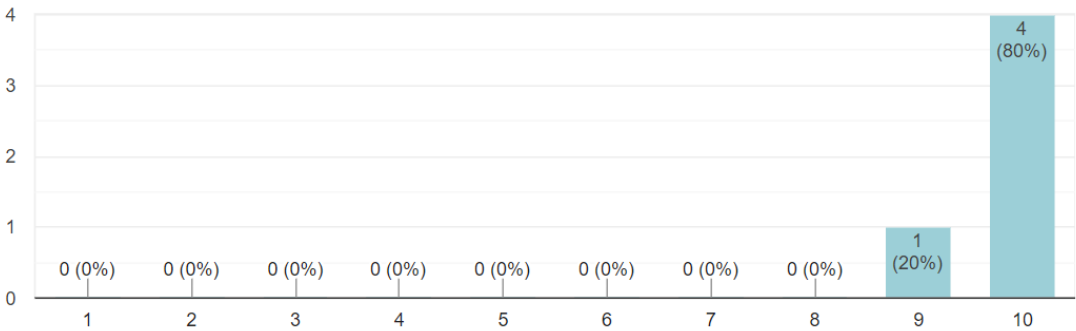
5 responses



Respondents found our application very easy to navigate throughout the whole testing.

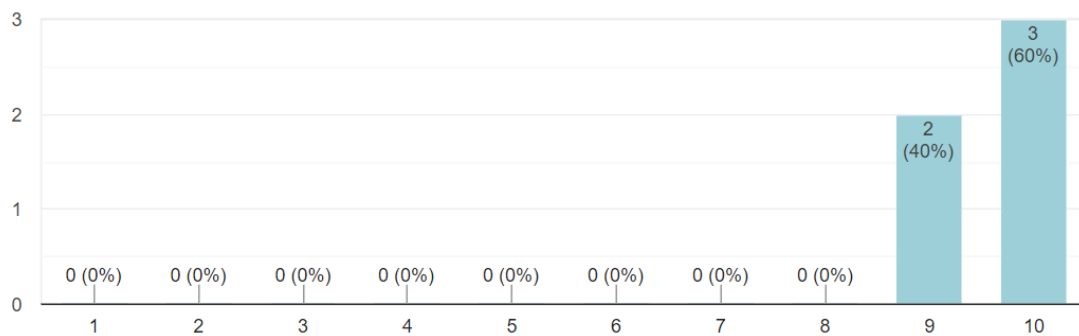
How did you find the process of joining a module?

5 responses



How did you find the process of watching a lecture video?

5 responses



Respondents found features such joining a module and watching a lecture video very easy on a scale of 1 to 10.

Overall, our feedback from both forms was very positive and we were very satisfied with the responses received.

## 1.4 Acceptance Testing

Using an acceptance testing approach, we ensured that as many areas and functionality as possible were covered for all users of the application. At the beginning of our project, we set out to achieve the outlined goals, the majority of which we were able to implement. In the future, we plan on implementing the goals not yet achieved.

No.	Functionality	Accept	Reject	Comment
1	DCU only access	✓		
2	User email verification	✓		
3	Login/Sign Up error handling	✓		
4	The lecturer can create a module	✓		
5	The lecturer can upload a video lecture	✓		
6	Students can join a module by password	✓		
7	Students can watch all uploaded videos in enrolled modules	✓		

8	The student's webcam takes continuous screenshots	✓		
9	Student can opt-out of the webcam		×	At the moment, there is no visual option for students to not partake in the facial analysis. Instead, they can just remove their webcam
10	The lecturer can view data reports for all videos	✓		
11	The lecturer can delete a module	✓		
12	The lecturer can delete a specific video	✓		
13	Student can unjoin a module		×	At the moment, students do not have the option to unjoin a module once joined. This is something we hope to implement in the future.

## 1.5 Accessibility

For testing accessibility, we ran multiple Chrome development tools to ensure our website was as accessible as possible for every student and lecturer. One of the main tools we used was the chrome extension "axe DevTools".



axe DevTools - Web Accessibility Testing

Offered by: [www.deque.com](http://www.deque.com)

This provided all the necessary tools to examine our website such as making sure all images have alternative text, aria-labels are present and even if the colour contrast throughout our

website is at an acceptable level.

The screenshot shows the Intendi website interface on the left and the DevTools axe-core extension on the right. The website has a header with navigation links (Home, About, Join Module, My Modules, Sign Out) and a main content area with a greeting 'Hello Karl!' and a section 'Here are your Modules' containing two cards for 'CA600' and 'CA420'. The DevTools extension on the right displays the 'TOTAL ISSUES' as 7, with a breakdown: Automatic Issues (7), Review Issues (3), and Guided Issues (0). A 'severity breakdown' link is also present. Below this, a list of issues is shown, with the first issue highlighted: 'IDs of active elements must be unique' (1 issue). The issue details panel on the right provides more information about this specific violation, including its impact, location, and a fix suggestion.

This screenshot shows the DevTools axe-core extension interface. At the top, it displays the URL 'http://localhost:3000/StudentHome'. Below this, a summary of issues is shown: 'TOTAL ISSUES' (7), 'Automatic Issues' (7), 'Review Issues' (3), and 'Guided Issues' (0). A 'severity breakdown' link is also present. Below this, a list of issues is shown, with the first issue highlighted: 'IDs of active elements must be unique' (1 issue). The issue details panel on the right provides more information about this specific violation, including its impact, location, and a fix suggestion.

Here we can see an example of the Lecturer Home page being tested. We soon realised some errors were not applicable to React JS as parts of the tests were more focused on the basic HTML aspects of it.

## 1.6 Heuristics

### Shneiderman's 8 Golden Rules

#### **Strive for consistency**

Throughout the development of the application, we ensured to stay consistent with both our designs, particularly as we technically have two types of websites (student and lecturer). The colour scheme and main theme of the website were decided on at an early stage and we ensured we kept a saved palette of colour in which we felt aligned with our brand image. We also tried our best to stay consistent with sizing with different aspects of the website. We also ensured the full coverage of the website was responsive.

#### **Enable frequent users to use shortcuts**

We wanted to make sure that the website was as easy as possible for users to use. This, therefore, led to us creating a navigation bar at the top of the page, which allowed users to jump from page to page on the website, whether it was going to the create a module screen then jumping straight into a specific module using the My Modules dropdown. The navigation bar ensures that all possible shortcuts are available to the user at any given time.

#### **Offer informative feedback**

We made sure the website felt like it was as interactive as possible. Wherever there was any form of any sort, we made sure to include well-designed alerts with vital information to make sure the user is well informed of the process of any given task they are doing. This included multiple form input checks prior to submitting a form and alerting the user to any possible errors, right up to letting the lecturer know that the video had successfully uploaded. Another informative feedback we implemented was once a lecturer uploaded a video to a module, all enrolled students received an automatic email to inform them of the new video available to watch.

#### **Offer simple error handling**

Error handling was implemented throughout the project in all the forms to ensure all the inputted information was correct and to make sure that the API request responses returned with a successful message.

#### **Permit easy reversal of actions**

We created multiple easy reversals of actions throughout, most notably the ability for the lecturer to both delete a video or even a whole module. We also allow users to reset their password, among other things.

#### **Design dialogues to yield closure**

Responses from common functions within the application resulted in informative feedback on whether or not the user's action was performed correctly and successfully.

**Support internal locus of control**

The user is not obliged to partake in any process, and all operations are of a full clear explanation of what they are.

**Reduce short-term memory load**

We made sure to keep all pages as clean and straightforward as possible. This meant bare minimum options, little to no window motion and easy to understand page location.