Software Engineering
Test Plan Documentation

Impedia: An Appeal Management System

Team Members:

IIT2019102: Shreyas Gupta

IIT2019105: Harshdeep Singh

IIT2019117: Sarthak Maheshwari

IIT2019142: Garvit Chittora

Table of Contents:

SECTION 1: INTRODUCTION

This is the testing plan for the Web App Impedia, which is an appeal management system developed for educational institutions to effectively handle and streamline communication between Authorities and students. This document provides a report of the results of the tests performed on the application.

The results of the test execution will be presented in subsection as follows:

- Name and the file of the executed test.
- Description of the test, or collected results.
- Errors or Bugs found(if any), and maybe a reference to the bug and fixes report.
- If an error was found in a certain part of a test case, the part is mentioned in the Bug report.
- All test cases were first executed and the found bugs were fixed. Then all test cases were executed again and it was ensured that no bugs were found.

Test were run on following environment:

- RAM: 8GB
- Processor: Intel i5 9th Gen
- OS: Windows 10 Home
- Tools: VS Code

SECTION 2: UNIT TESTING

Since our web app is split into two major components, the Frontend and the Backend, the Unit tests are also split into Frontend tests and Backend tests.

In the Backend, each controller function is checked, whether or not it is working as expected in the ideal case scenario, and it is also checked that it is handling exception cases and errors as intended.

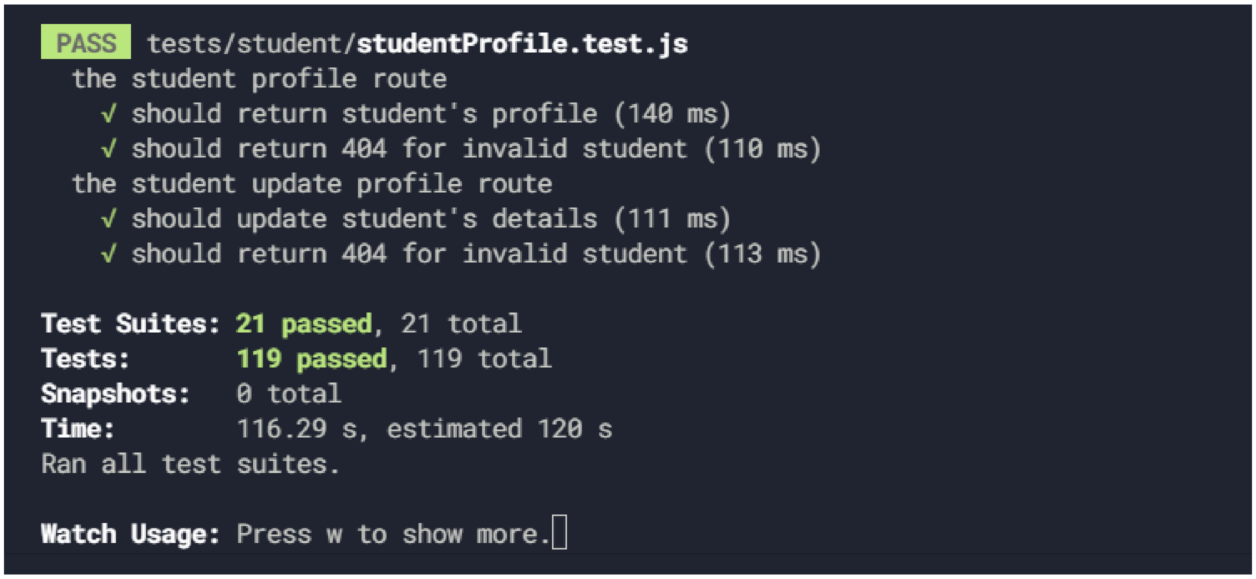
In the Frontend, the same is done for the components, which are checked if they are functioning normally in the ideal scenario, and also in the exceptional scenario, where some invalid data or action is performed by the user and it is tested to ensure that the app responds as intended.

All the tests are fully automated and are written using Jest and Supertest libraries. We have multiple test suites testing each and every controller on the backend and snapshot tests and login/logout and component tests on the frontend. The tests can be run by typing

npm run test

in the terminal. This will run the tests for backend/frontend depending upon which folder you are in currently.

You should see a final output as below in the image. The image below shows that out of a total 119 test cases, all 119 pass.


```
 PASS  tests/student/studentProfile.test.js
  the student profile route
    √ should return student's profile (140 ms)
    √ should return 404 for invalid student (110 ms)
  the student update profile route
    √ should update student's details (111 ms)
    √ should return 404 for invalid student (113 ms)

Test Suites: 21 passed, 21 total
Tests:       119 passed, 119 total
Snapshots:   0 total
Time:        116.29 s, estimated 120 s
Ran all test suites.

Watch Usage: Press w to show more.
```

Backend

Test suite no.	**Suite Name**	**Test Files**	**Test Case**
1.	admin	1a) auth	Admin authentication to ensure that the admin can login to the app.
		1b) setDomainAddAuth	Ensure that the admin can add authorities and set the organisation email domain.
		1c) authorityGroup	Checking to ensure that the admin can create authority groups.

		1d) getAppealsAndPetitions	Checking to see that the admin can access all the appeals and petitions that are made on the app.
2.	authority	2a) auth	Authority authentication to check whether the authority can login to the app.
		2b) authorityProfile	Checking whether the authorities can access and modify their profile.
		2c) getAuthAppealsAndPetitions	Checking whether the authority can access the appeals and petitions made to them.
		2d) getAuthority	Checking whether the authority data is returned.
3.	student	3a) registerStudent	Checking whether the student can register to the app.
		3b) loginStudent	Checks whether the registered student can login to the app.
		3c) studentProfile	Checking whether the students can access and modify their profile.
		3d) createAppeals	Checking whether the student can create new appeals
		3e) createPetitions	Checks whether the

			student can create a new petition
		3f) getAppealsAndPetitions	Checks whether the appeals and petitions made by the student can be accessed by them.
4.	appeal	4a) getAppealById	Checks whether an appeal's details can be accessed depending on the user type.
		4b) getAppealReplies	Checks whether the replies can be accessed by the users for a particular appeal
5.	group	getGroups	Checks whether the authority group data can be accessed.
6.	petition	6a) getPetitionById	Checks whether a petition's details can be accessed depending on the user type.
		6b) signPetitions	Checking whether the petition can be signed by the students.
		6c) getPetitionReplies	Checks whether the replies can be accessed by the users for a particular petition
		6d) decision	Checks whether the authorities can take a

			decision on the petition. Also ensures that only intended authorities can make a decision.
7.	reply	7a) addReply	Checks whether the users can reply to the appeals and petitions
8.	resetPassword	8a) resetPassword	Checks that the students and authorities are able to reset their password in case they forget it.

Test Suite 1: admin

1. **auth**
   This test is responsible for checking the process of logging in the Admin to the app. The credentials of the admin are sent from the frontend, then they are verified with the contents in the database, and if they are correct, a token is sent back to the frontend, via which it is checked that the admin is logged in.

2. **setDomainAddAuth**
   This test is responsible for two tasks, the setDomain functionality and the add authority functionality For setting domain, the admin is required to enter the email domain for the organisation and for adding authorities, the admin enters their emails, and a random password is generated and the authorities are notified for the same.

3. **authorityGroup**

The admin is responsible for making and modifying the authority groups for the application.
This test is responsible for ensuring that the admin can create, edit and delete authority groups.
For creating a group, the admin is required to enter the group name and the members' emails via the frontend.
For editing a group, the admin enters the required changes, like change in the name or adding or removing some members and submits the data and the group is found and modified.
For deleting a group, admin simply selects the group to be deleted and gives the confirmation and that data is deleted.

4. **getAppealsAndPetitions**
   The admin can access all the appeals and petitions being made on the app. This test is responsible to test this functionality and ensure that all the appeals and petitions data is sent to the admin as it was intended to be sent. The admin visits the view appeals and petitions route and the data is displayed to them.

Test Suite 2: authority

1. **auth**
   This test is responsible for checking the process of logging in the authority to the app. The credentials of the authority are sent from the frontend, then they are verified with the contents in the database, and if they are correct, a token is sent back to the frontend, via which it is checked that the admin is logged in.

2. **authorityProfile**

The authorities can view and edit some aspects of their profile on the application. This test is responsible for ensuring that the authorities can view their profiles as well as edit their profiles.

3. **getAuthAppealsAndPetitions**
   The authorities can view all appeals and petitions addressed to them at a single location. This test's task is to check whether this functionality is working as intended. It checks if the correct data is sent to the authority when they ask for it.

4. **getAuthority**
   All the users on the app can search for the authorities that are registered on the app. This test ensures that all the users can request for the authority data and whether the correct data is sent to them or not.

Test Suite 3: student

1. **registerStudent**
   A student of the organisation can register to the app with their organisation email whose domain is already specified by the admin. This test ensures that a student with valid data can register to the app whereas invalid users(those outside the organisation, or those who did not enter all the data) cannot register.

2. **loginStudent**
   The authorities can view and edit some aspects of their profile on the application. This test is responsible for ensuring that the authorities can view their profiles as well as edit their profiles.

3. **studentProfile**

The students can view and edit some aspects of their profile on the application. This test is responsible for ensuring that they can view their profiles as well as edit their profiles.

4. **createAppeals**
   All the students can create appeals to certain authorities or groups of authorities which can be viewed by the student and the authority/group of authorities. This test ensures that the students can create appeals with valid data and checks that the errors are sent in case of invalid user or incorrect data.

5. **createPetitions**
   All the students can create petitions to certain authorities or groups of authorities which can be viewed by all the users of the app. This test ensures that the students can create petitions with valid data and checks that the errors are sent in case of invalid user or incorrect data.

6. **getAppealsAndPetitions**
   All the students can view the appeals created by them and the petitions which are public. This test ensures that the students receive correct appeals and petitions data when they are logged in and they ask for the data.

Test Suite 4: appeal

1. **getAppealById**
   A user can access an appeal by its ID if the appeal is made by that user, or the appeal is made to the user or if the user is the admin. This test checks that the appeals can be accessed by the valid users and also checks that users cannot access appeals that they are not authorised to.

2. **getAppealReplies**

A user can reply to an appeal if the appeal was made by them or if it was made to them. This test ensures that the replies for an appeal can be fetched through a valid request.

Test Suite 5: group

1. **getGroups**
   A user can request for the authority groups data in order to select a group to make an appeal or a petition to. This test ensures that valid users can access authority groups.

Test Suite 6: petition

1. **getPetitionById**
   Petitions are public and any logged in user can access them. This test ensures that logged in users can ask for the petition data and also checks whether or not the correct data is being sent.

2. **signPetitions**
   A logged in user can sign a petition to show their support towards it. This test ensures that a petition can be signed by the users and that their data is stored as a signee for that petition when they sign.

3. **getPetitionReplies**
   A user can reply to any petition since petitions are public and can be viewed and signed by anybody. This test ensures that the replies for a petition can be fetched through a valid request.

4. **decision**

The authority or the group of authorities to whom the petition was made can make a decision on the said petition. This test ensures that the decision can only be made by the authorities to whom the appeal was made. It also checks that when the decision is made on the frontend, the data is updated on the backend and the decision is stored.

Test Suite 7: reply

1. **addReply**
   A user can add a reply to a petition or to an appeal if the appeal was made by the user or to the user. This test ensures that the user is able add replies to the valid appeals and petitions.

Test Suite 8: resetPassword

1. **resetPassword**
   The users can reset their passwords in case they forget it. This test ensures that the users are sent an email containing the information to reset their password. It also ensures that after the user changes their password, it is updated in the database and the user can log into the site.

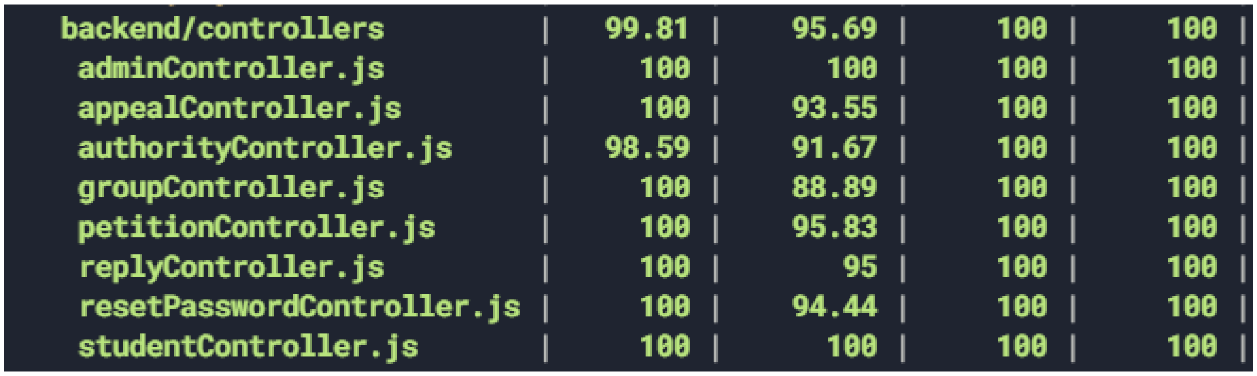
Coverage

Code coverage for the tests shows the percentage of code the tests are covering. The coverage for our tests can be seen by running

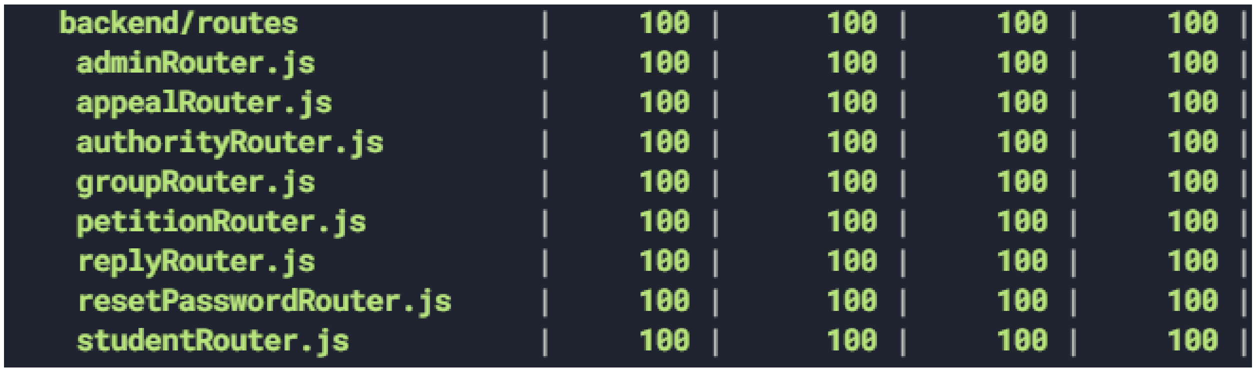
npm run coverage

in the terminal. The final coverage report for the controllers and the routes is added below.

Controllers:


```
  backend/controllers           |   99.81 |    95.69 |      100 |      100 |
   adminController.js           |     100 |      100 |      100 |      100 |
   appealController.js          |     100 |    93.55 |      100 |      100 |
   authorityController.js       |   98.59 |    91.67 |      100 |      100 |
   groupController.js           |     100 |    88.89 |      100 |      100 |
   petitionController.js        |     100 |    95.83 |      100 |      100 |
   replyController.js           |     100 |       95 |      100 |      100 |
   resetPasswordController.js   |     100 |    94.44 |      100 |      100 |
   studentController.js         |     100 |      100 |      100 |      100 |
```

Routes:

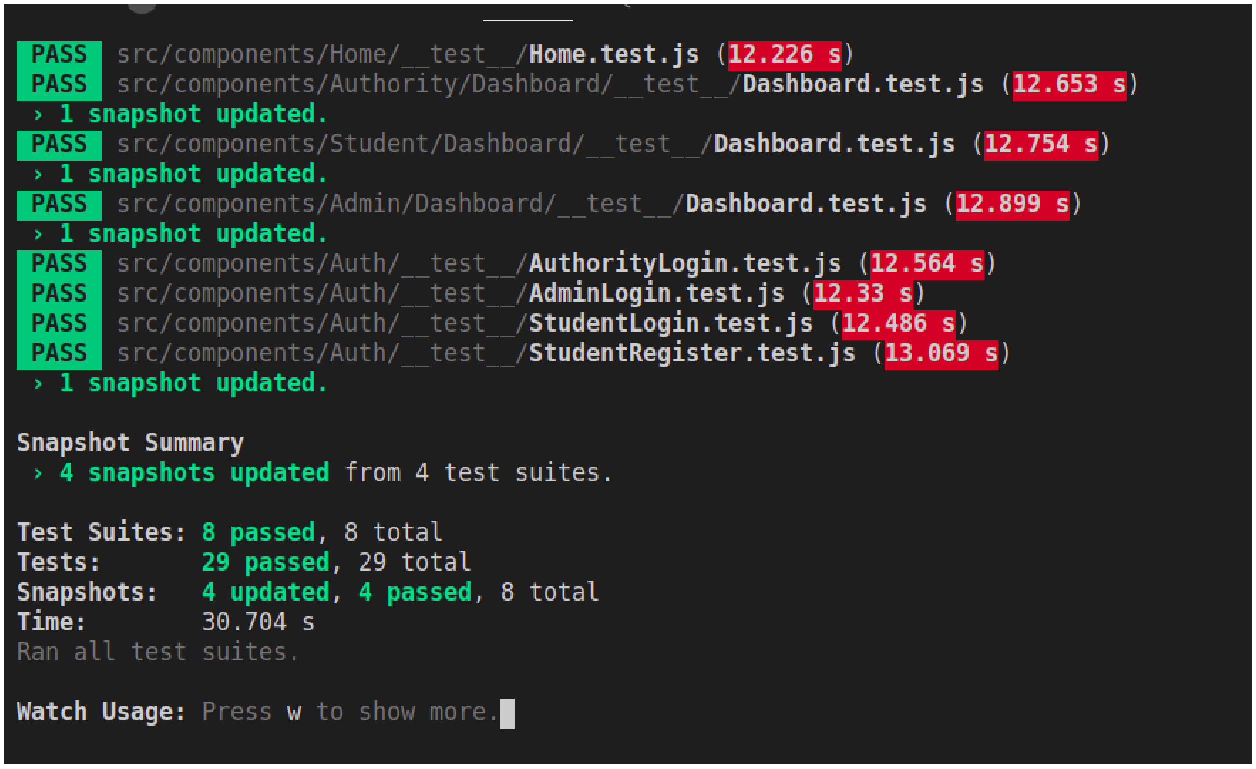

```
  backend/routes                |     100 |      100 |      100 |      100 |
   adminRouter.js               |     100 |      100 |      100 |      100 |
   appealRouter.js              |     100 |      100 |      100 |      100 |
   authorityRouter.js           |     100 |      100 |      100 |      100 |
   groupRouter.js               |     100 |      100 |      100 |      100 |
   petitionRouter.js            |     100 |      100 |      100 |      100 |
   replyRouter.js               |     100 |      100 |      100 |      100 |
   resetPasswordRouter.js       |     100 |      100 |      100 |      100 |
   studentRouter.js             |     100 |      100 |      100 |      100 |
```

Frontend

Test suite no.	**Suite Name**	**Test Files**	**Test Case**
1.	Admin	1a) AddAuthority	Ensure that the admin can add authorities.
		1b) AddGroup	Checking to ensure that the admin can create authority groups.
		1c) AllAppeals	Checking to see that the admin can access all the appeals that are made on

			the app.
		1d) AllPetitions	Checking to see that the admin can access all the petitions that are made on the app.
		1e) ChangeDomain	Ensure that the admin can set the organisation email domain.
		1f) EditGroup	Ensure that the admin can edit authorities groups.
		1g) Dashboard	Check to see that all the buttons available on the Admin Dashboard are redirecting to the correct page.
2.	Auth	2a) AdminLogin	Admin authentication to ensure that the admin can login to the app and only admin has access to admin use cases.
		2b) AuthorityLogin	Authority authentication to ensure that the authority can login to the app and only authority has access to authority use cases.
		2c) StudentLogin	Student authentication to ensure that the student can login to the app and only student has access to student use cases.
		2d) StudentRegister	Student registrations to ensure that the Student can register to the app.
3.	Authorit y	3a) AllAppeals	Checking to see that the authority can access all the appeals that are created for him.
		3b) AllPetitions	Checking to see that the authority can access all the petitions that are made

			on the app.
		3c) UpdateProfile	Checking to see that the authority can update his/her info.
4.	Home	4a) Home	Check to see that all the buttons available on the homepage are redirecting to the correct page.
5.	Student	5a) CreateAppeal	Checking to see that the student can create appeals for any single authority or for any authority group.
		5b) CreatePetition	Checking to see that the student can create a petition for any single authority or for any authority group.
		5c) AllAppeals	Checking to see that the student can access all the appeals that are created by him.
		5d) AllPetitions	Checking to see that the student can access all the petitions that are made on the app.
		5e) Dashboard	Check to see that all the buttons available on the Student Dashboard are redirecting to the correct page.
		5f) UpdateProfile	Checking to see that the Student can update his/her info.



```
 PASS  src/components/Home/__test__/Home.test.js (12.226 s)
 PASS  src/components/Authority/Dashboard/__test__/Dashboard.test.js (12.653 s)
  › 1 snapshot updated.
 PASS  src/components/Student/Dashboard/__test__/Dashboard.test.js (12.754 s)
  › 1 snapshot updated.
 PASS  src/components/Admin/Dashboard/__test__/Dashboard.test.js (12.899 s)
  › 1 snapshot updated.
 PASS  src/components/Auth/__test__/AuthorityLogin.test.js (12.564 s)
 PASS  src/components/Auth/__test__/AdminLogin.test.js (12.33 s)
 PASS  src/components/Auth/__test__/StudentLogin.test.js (12.486 s)
 PASS  src/components/Auth/__test__/StudentRegister.test.js (13.069 s)
  › 1 snapshot updated.

Snapshot Summary
 › 4 snapshots updated from 4 test suites.

Test Suites: 8 passed, 8 total
Tests:       29 passed, 29 total
Snapshots:   4 updated, 4 passed, 8 total
Time:        30.704 s
Ran all test suites.

Watch Usage: Press w to show more.
```

Test Suite 1: Admin

1. AddAuthority

This test is responsible for checking if the email Ids taken as input from the Admin are stored properly in an array; checking that the request to add authorities is being sent at the correct path to the backend with all required fields and either confirming that the status received is 201 or showing appropriate error to the user.

2. AddGroup

This test is responsible for checking if the email Ids and the group name taken as input from the Admin are stored properly and the request to create a new group is sent to the correct path at the backend; and either confirming that the status received is 200 or 204 or showing appropriate error to the user.

3. **AllAppeals**
   This test is responsible for checking if the “get” request to fetch all the appeals is sent to the correct path at the backend, is received and the state containing it is updated and then is finally rendered at the frontend.

4. **AllPetitions**
   This test is responsible for checking if the “get” request to fetch all the petitions is sent to the correct path at the backend, is received and the state containing it is updated and then is finally rendered at the frontend.

5. **ChangeDomain**
   This test is responsible for checking if the state storing the value for new domain is updated accordingly, and also that the request to change domain is sent to the correct path at the backend which either receives a status of 200 / 204 or showing the particular error that occured .

6. **EditGroup**
   This test is responsible for checking if the states containing email Ids and the group name are updated properly and the request to edit the group is sent to the correct path at the backend; and either confirming that the status received is 200 or 204 or showing appropriate error to the user.

7. **Dashboard**
   This test is responsible for checking if the buttons on the dashboard for various use cases are working and redirect the admin to the specific route. Also, it must ensure that if the user is not an admin, the user gets redirected to the login page first.

Test Suite 2: Auth

1. **AdminLogin**
   This test is responsible for checking if the states storing values of email and password are updated on change, the request with all the credentials is sent to the right path at the backend and ensuring either the status is 200 or showing an appropriate message to the user about the error . Also, it must check that upon successful login the admin gets redirected to admin’s dashboard immediately.

2. **AuthorityLogin**
   This test is responsible for checking if the states storing values of email and password are updated on change, the request with all the credentials is sent to the right path at the backend and ensuring either the status is 200 or showing an appropriate message to the user about the error . Also, it must check that upon successful login the authority gets redirected to the authority's dashboard immediately.

3. **StudentLogin**
   This test is responsible for checking if the states storing values of email and password are updated on change, the request with all the credentials is sent to the right path at the backend and ensuring either the status is 200 or showing an appropriate message to the user about the error . Also, it must check that upon successful login the student gets redirected to the student's dashboard immediately.

4. **StudentRegister**
   This test is responsible for checking if the states storing values of all the required student fields are updated on change, the request with all the credentials is sent to the right path at the backend and ensuring either the status is 201/200 or showing an appropriate message to the user about the error . Also, it must check that upon successful registration the page is redirected to the student's dashboard.

Test Suite 3: Authority

1. **AllAppeals**
   This test is responsible for checking if the “get” request to fetch all the appeals meant for this authority is sent to the correct path at the backend, is received and the state containing it is updated and then is finally rendered at the frontend.

2. **AllPetitions**
   This test is responsible for checking if the “get” request to fetch all the petitions meant for this authority is sent to the correct path at the backend, is received and the state containing it is updated and then is finally rendered at the frontend.

3. **UpdateProfile**
   This test is responsible for checking if the states storing values for Name, Receive Notifications and New Password updates on change and the particular request is sent to the correct path at the backend and ensures that Old Password is required when the request is to set a new password. Also, it ensures either the status received from the backend is 204 or it shows appropriate errors to the user.

Test Suite 4: Home

1. **Home**
   This test is responsible for checking if the buttons on the Home Page f are working and are redirecting the user to the specific route.

Test Suite 5: Student

1. **CreateAppeal**
   This test is responsible for ensuring that the “get” request for fetching details of authorities and groups is sent to the backend at the correct route and states storing it at frontend are updated on successful fetch of the details. It also checks if the states storing the values for the authority/ group to send the appeal to, the Appeal Title and Content are updated. It also has to ensure that the formatted text corresponding to the HTML or Markdown Input is visible for preview without errors. Also, it has to check that the request for New Appeal is sent to the correct path at the backend and either the status received is either 201 or display the specific error.

2. **CreatePetition**
   This test is responsible for ensuring that the “get” request for fetching details of authorities and groups is sent to the backend at the correct route and states storing it at frontend are updated on successful fetch of the details. It also checks if the states storing the values for the authority/ group to send the petition to, the Petition Title and Content are updated. It also has to ensure that the formatted text corresponding to the HTML or Markdown Input is visible for preview without errors. Also, it has to check that the request for New Petition is sent to the correct path at the backend and either the status received is either 201 or display the specific error.

4. **AllAppeals**
   This test is responsible for checking if the “get” request to fetch all the appeals meant for this student is sent to the correct path at the backend, is received and the state containing it is updated and then is finally rendered at the frontend.

5. **AllPetitions**
   This test is responsible for checking if the “get” request to fetch all the petitions meant for this student is sent to the correct path at the backend, is

received and the state containing it is updated and then is finally rendered at the frontend.

6. **UpdateProfile**

   This test is responsible for checking if the states storing values for Name, Receive Notifications and New Password updates on change and the particular request is sent to the correct path at the backend and ensures that Old Password is required when the request is to set a new password. Also, it ensures either the status received from the backend is 204 or it shows

   appropriate errors to the user.

SECTION 3: BUGS FOUND AND FIXED

Bug 1:

Description: At the frontend, where suggestions/ options feature was provided, eg: Selecting Authority Names from the list while Adding Authorities - the whole page would go blank and would eventually collapse.

Reason : We found out that the reason for this was that the options container required an array to display the options and since we are getting these details from the backend - which has to be an asynchronous request, it is not available immediately and for that time the options input remains undefined which causes the page to collapse.

Fix : While we do not receive a response from the asynchronous request we set the options initially to be an empty array and only when the frontend receives that data from the backend we set the options accordingly.

Bug 2:

Description: Although the admin should be able to see all the appeals, none of the appeals were being fetched from the backend and an error message saying Unauthorised was received.

Reason : Since an appeal only has to be visible to the student who created it or the authority it was made to, there was a condition at the backend specifically for this. However, we missed to check if it was the Admin who made this request. The appeal should then also be sent .

Fix : We checked this condition first if the user was admin, and if it was the case the backend authorises the admin directly and sends the appeal.

Bug 3:

Description: The option to sign a petition is also visible to Admin and Authorities.

Reason : We did not check if the user is not a Student and also since the page to view a petition is common for all, it was not possible to send this through the backend.

Fix : We used cookies instead of local storage to store more information like the type of user at the time of login itself to overcome this.

Bug 4:

Description: The comment section becomes very long when there is a large number of replies.

Reason : The height of the Replies box was set to adjust with the replies added which caused it to take a lot of space in case of a large number of replies.

Fix : We set a maximum height for the Replies box and in case the content inside it overflows, we set scroll property on it so that now its length doesn't go beyond a certain limit and instead it has a scrolling option.

Bug 5:

Description: The appeals and petitions being sent to the frontend were not populated properly, that is, only the ids of the authority/group to whom the appeal/petition was directed was sent, instead of the data of the authority/group.

Reason : The query to retrieve and populate the data of the appeals/petitions was in place but it was not being executed correctly and hence the data was being sent without being populated completely.

Fix : An .execPopulate() command was chained to the query in order to execute the requested data changes which led to the data being populated and sent as required.

Bug 6:

Description: Users with malformed login tokens could access all the features of the app which were not meant for them without any verification.

Reason : We were checking the token id to be either admin's, or authority's or student's and then allowing access to the parts of the website meant for them. This could also allow people who malformed the tokens to fake as an admin or authority.

Fix : Instead of just checking with the IDs, the database was also queried to ensure that the token's user exists and has the same role as the ID is pointing towards.