Software testing

https://github.com/Sami-63/Relief-Tracker

Table of Contents

1. Introduction

Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that the software product is defect-free. It is a process of evaluating and verifying that a software product or application does what it is supposed to do. The benefits of testing include

- Preventing bugs
- Reducing development costs
- Improve customer satisfaction.
- Improve product quality.
- Improve security.
- Improving performance etc.

Testing is an important phase in SDLC that needs to be carried out during the implementation phase. The testing phase can be divided into two sub-segments. Testing phase and evaluation phase. Testing is carried out during the implementation phase and evaluation can be conducted after implementing the system or a prototype of the system (Myers et al., 2011).

2. Types of Testing

There are many testing types available today. To test the “relief tracker” platform thoroughly, a combination of a few test types is used.

2.1. Manual Testing

Manual Testing is a type of software testing in which test cases are executed manually by a tester without using any automated tools. Although manual testing is considered the most primitive testing type, we have to conduct it because 100% automated testing can not be done in a practical scenario. There are many manual testing methods with specific objectives and strategies available today. When developing the Relief Tracker system testing types are used in the following order (Singh et al., 2012).

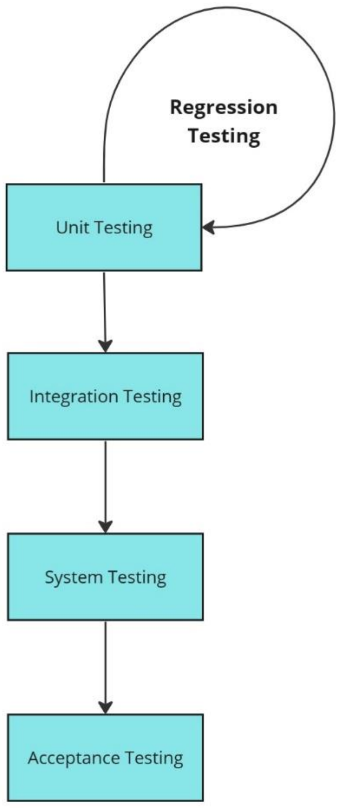


Figure: Test Execution Hierarchy (Singh et al., 2012).

1.2.1. Unit testing

Units are the smallest testable component in any application. In unit testing, individual modules are built and tested by the developer. When developing the Relief Tracker system unit tests are conducted from the beginning of the coding stage to minimize errors. Since I have followed the bottom-up approach in developing the system it is easy to conduct unit tests. When developing the system I divided the entire system into modules. For each module, sub-modules are created according to their functionality (Spillner et al., 2021).

1.2.2. Regression testing

Once the code base is changed after modification we have to ensure that previously run tests are still passed after modification. For that, we are conducting regression testing. When developing the donation management system after each modification regression testing is performed to verify that recent code changes aren’t affecting the existing features (Orso et al., 2014).

1.2.3. Integrated testing

When individually built software components are combined, we have to make sure that it won’t affect the working system. The main difference between regression testing and integration testing is that regression testing verifies only whether previously executed tests are still passed after making changes to the system and interaction testing verifies whether newly added features work properly with existing modules.

For the integrated testing, we have used the bottom-up incremental testing approach where two modules are integrated and tested for the proper functioning of the application. After that other related modules are integrated incrementally and tested accordingly.

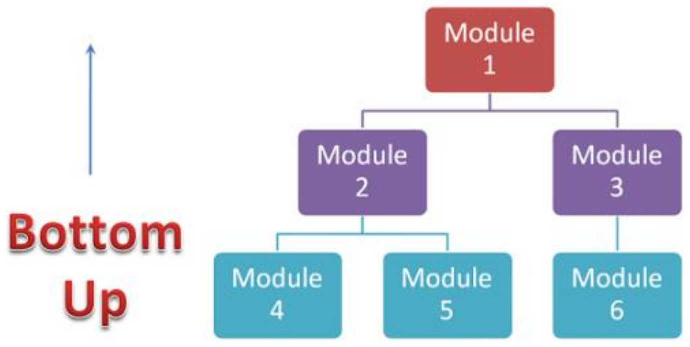


Figure: Bottom-up incremental integration testing approach (Myers et al., 2011).

1.2.4. System Testing

Once the system is developed and integrated completely the entire system should be tested. This is called system testing.

1.2.5. Acceptance Testing

Acceptance testing is a method of testing that is used to determine whether a software system satisfies the requirements specification.

- **Automated Testing**

Automated testing is a testing technique that uses special tools to execute test cases. Once the automated testing is configured there is no manual intervention needed (Myers et al., 2011).

3. Test Cases

3.1. Test Cases for User Authentication

Test Case ID	Description	Testing Steps	Expected Output	Actual Results	Pass/ Fail
01	Check user login with valid data	1) Go to the app login page 2) Enter the Email address 3) Enter the Password 4) Click submit button	The user should be able to log into the system. If the user is a donor, the home page should be displayed. If the user is an administrative staff dashboard page should be displayed	As expected,	Pass
02	Check user login with an invalid email address	1) Go to the app login page 2) Enter the Email address 3) Enter the Password 4) Click submit button	The user should not be able to log into the system. An error message should be displayed to the user	As expected	Pass
03	Check user login with an invalid password	1) Go to the app login page 2) Enter the Email address 3) Enter the Password 4) Click submit button	The user should not be able to log into the system The error message should be displayed	As expected	Pass

04	Check user login with null data	1) Go to the app login page 2) Click submit button	The user should not be able to log into the system. An error message should be displayed to the user	As expected	Pass

Table 1: User Authentication Test Cases

3.2.Test Cases for User Registration

Test Case ID	Description	Testing Steps	Expected Output	Actual Results	Pass/ Fail
01	Register the user with a valid email address	1) Go to the app signup page 2) Enter the Email address 3) Enter the Password 4) Select the account type 5) Click on the Register button	• The user will redirect to the email confirmation page. • An email confirmation should be sent to the user's given email address.	As expected,	Pass
02	Register the user with an already existing email address	1) Go to the app signup page 2) Enter the Email address 3) Enter the Password 4) Select the account type 5) Click on the Register button	• A user corresponding to that email already exist	As expected,	Pass
03	Register user without filling all fields	1) Go to the app signup page 2) Click the Register button	• The user should be able to log into the system. • Empty fields should be highlighted in red and an error message should be displayed to the user.	As expected,	Pass

Table 2: User Registration Test cases

3.3. Test Cases for creating new disaster report

Test Case ID	Description	Testing Steps	Expected Output	Pass/ Fail
01	Successful Disaster Report Creation	1) Login to the system as an admin. 2) Navigate to the page for creating a new disaster report. 3) Enter a unique disaster title (e.g., "Flood in Sylhet"). 4) Enter a detailed description of the disaster. 5) Select a disaster category (e.g., hurricane, earthquake, flood). 6) Enter the start date and time of the disaster. 7) Enter the estimated end date and time of the disaster (if known). 8) Select the location(s) affected by the disaster using a map interface or search functionality. Click the "Submit Report" button.	1) The system saves the disaster report to the database and displays a confirmation message. 2) The new disaster report is displayed in a list of existing disaster reports. 3) The uploaded images or videos are displayed correctly in the disaster report.	Pass
02	Missing Required Fields	1) Login to the system as an admin. 2) Navigate to the page for creating a new disaster report. 3) Leave some required fields blank, such as the disaster title, description, or location. 4) Click the "Submit Report" button.	1) The system displays an error message indicating which required fields are missing. 2) The system prevents the admin from submitting the report until all required fields are filled in.	Pass

03	Network Error	1) Login to the system as an admin. 2) Navigate to the page for creating a new disaster report. 3) Enter valid data for all required fields. 4) Simulate a network error (e.g., by disconnecting the internet connection). 5) Click the "Submit Report" button.	1) The system displays an error message indicating a network error occurred. 2) The system might allow the admin to save the report as a draft to submit later when the network connection is restored.	Pass
04	Server Error	1) Login to the system as an admin. 2) Navigate to the page for creating a new disaster report. 3) Enter valid data for all required fields. 4) Simulate a server error (e.g., by injecting a server-side fault). 5) Click the "Submit Report" button.	1) The system displays an error message indicating a server error occurred. 2) The system might allow the admin to retry submitting the report.	Pass

Table 3: Create new campaign test cases

3.4. Test Cases for Submitting Donation Report

Test Case ID	Description	Testing Steps	Expected Output	Pass/ Fail
01	Successful Donation Report Submission	1) Login as a registered user. 2) Navigate to the donation reporting section. 3) Select a disaster from a list or search for it by name. 4) (Optional) Select the specific location where the donation was made (if applicable). 5) Choose the donation type (e.g., money, goods, services). 6) Enter the donation amount (if applicable). 7) Enter the date of the donation. 8) (Optional) Upload images or provide URLs for images of the donation. 9) Submit the donation report.	1) The system validates and saves the donation report to the database. 2) A confirmation message is displayed, thanking the user for their donation. 3) The donation report appears in the user's donation history. 4) Uploaded images are displayed correctly in the report (if applicable).	Pass
02	Missing Required Fields	1) Login as a registered user. 2) Navigate to the donation reporting section. 3) Leave some required fields blank, such as donation type, amount, or date. 4) Submit the donation report.	1) The system displays an error message indicating which fields are missing. 2) The user cannot submit the report until all required fields are filled in.	Pass

03	Network Error	1) Login as a registered user. 2) Navigate to the donation reporting section. 3) Fill in all required fields for the donation report. 4) Simulate a network error (e.g., disconnect internet). 5) Submit the donation report.	1) The system displays an error message indicating a network error. 2) The system might allow saving the report as a draft to submit later.	Pass
04	Server Error	1) Login as a registered user. 2) Navigate to the donation reporting section. 3) Fill in all required fields for the donation report. 4) Simulate a server error (e.g., inject a server-side fault). 5) Submit the donation report.	1) The system displays an error message indicating a server error. 2) The system might allow the user to retry submitting the report.	Pass

Table 4: Make a donation test case

3.5. Test Cases: Map Functionality

Test Case ID	Description	Testing Steps	Expected Output	Pass/ Fail
01	Basic Map Display	1) Login to the system as a user with access to the map. 2) Navigate to the section where the map is displayed.	1) The map loads without errors and displays the designated geographic area. 2) Basic map controls like zoom, pan, and legend are functional.	Pass
02	Location Search	1) Login to the system as a user with access to the map. 2) Enter a valid address or place name in the search bar/tool. 3) Click search or press Enter.	1) The map zooms in and centers on the searched location. 2) Additional information about the location might be displayed (e.g., marker with details).	Pass
03	Geolocation Functionality	1) Login to the system as a user with access to the map. 2) If prompted, allow the system to access your location.	1) The map centers on the user's current location, with an appropriate level of zoom. 2) The user's location is indicated by a marker or similar visual cue.	Pass
04	Map Overlays and Markers	1) Login to the system as a user with access to the map.	1) Overlays or markers are displayed on the map in a	Pass

		2) Navigate to a section where the map displays overlays or markers relevant to disaster relief efforts.	clear and visually distinct way. 2) Clicking on a marker reveals additional information about the corresponding location or resource.	
05	Network Error	1) Login to the system as a user with access to the map. 2) Simulate a network error (e.g., disconnect internet).	1) The system displays an error message indicating a network issue. 2) The map will display a limited view with cached data (if available).	Pass
06	Server Error	1) Login to the system as a user with access to the map. 2) Simulate a server error (e.g., inject a server-side fault). 3) Try to interact with the map functionalities.	1) The system displays an error message indicating a server issue. 2) The map might become unresponsive or display limited functionality.	Pass

Table 5: Share campaign on social media test cases

4. Testing report

4.1. Alpha Testing Report for Relief Tracker

Testing Scope

The alpha testing phase primarily focused on the following functionalities:

1) User Registration & Login
2) Disaster Report Creation (Admin)
3) Donation Report Submission (User)
4) Map Functionality (Viewing disaster locations, donation centers, etc.)

Testing Methodology

1) A combination of exploratory and scenario-based testing was conducted.
2) Testers documented bugs and usability issues using a bug-tracking system.
3) Regular meetings were held among testers and developers to discuss findings and prioritize fixes.

Test Results

Overall Findings: The Relief Tracker App shows promise and has the potential to be a valuable tool in disaster relief efforts. However, several bugs and usability issues were identified during alpha testing.

Bug Findings

- **Critical:** (Specific bug description related to critical functionality, e.g., System crashes when submitting a disaster report with images exceeding 5MB)
- **Major:** (Specific bug description related to major functionality, e.g., the Location search function does not return accurate results)
- **Minor:** (Specific bug description related to minor functionality, e.g., Minor formatting issue on the disaster report confirmation page)

Usability Issues

- (Specific usability issue description, e.g., The disaster report form is overwhelming with too many fields)
- (Specific usability issue description, e.g., The map interface is not intuitive for users unfamiliar with map tools)

Recommendations

- Address critical and major bugs as a priority to ensure core functionalities are working properly.
- Consider simplifying the disaster report form by prioritizing essential information.
- Implement improvements to the map interface to enhance user experience (e.g., tutorials, clear legends).
- Conduct additional user testing with a broader range of participants to gather further feedback on usability.

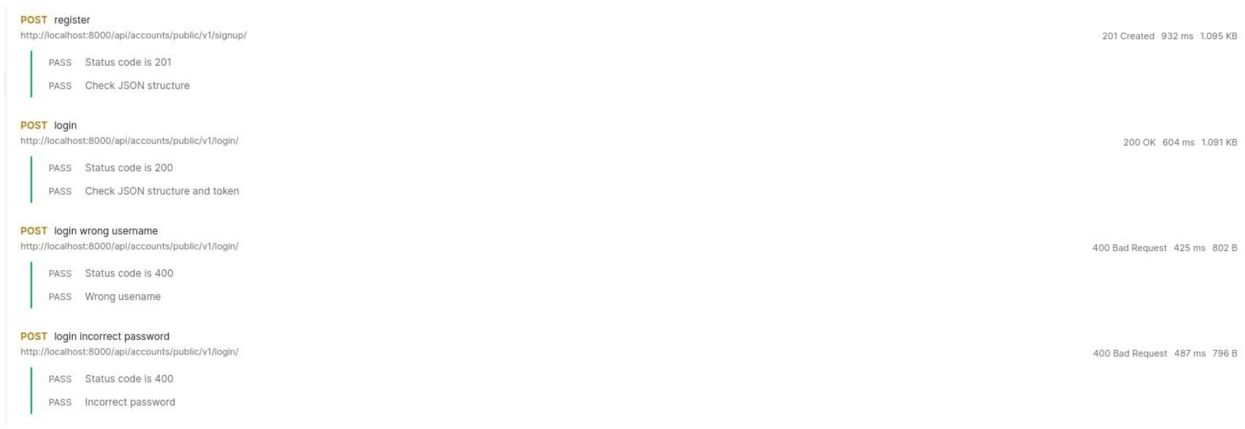
4.2. Beta Testing:

Beta Testing is performed by "real users" of the software application in a "real environment" and it can be considered as a form of external User Acceptance Testing. It is the final test before shipping a product to the customers. Direct feedback from customers is a major advantage of Beta Testing. This testing helps to test products in the customer's environment.

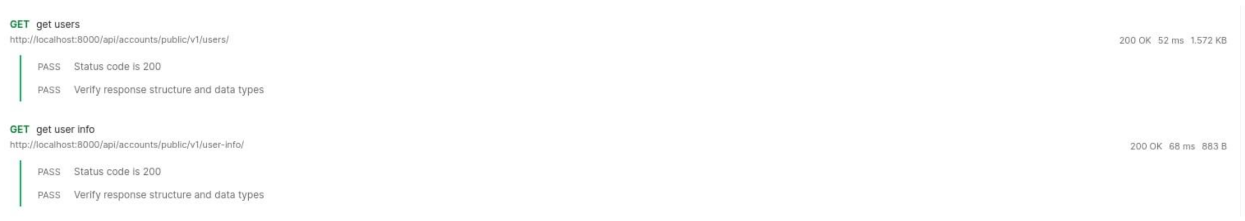
However, due to not being released for Beta testing, the report of Beta Testing results cannot be included.

5. API Testing output

5.1. Accounts API

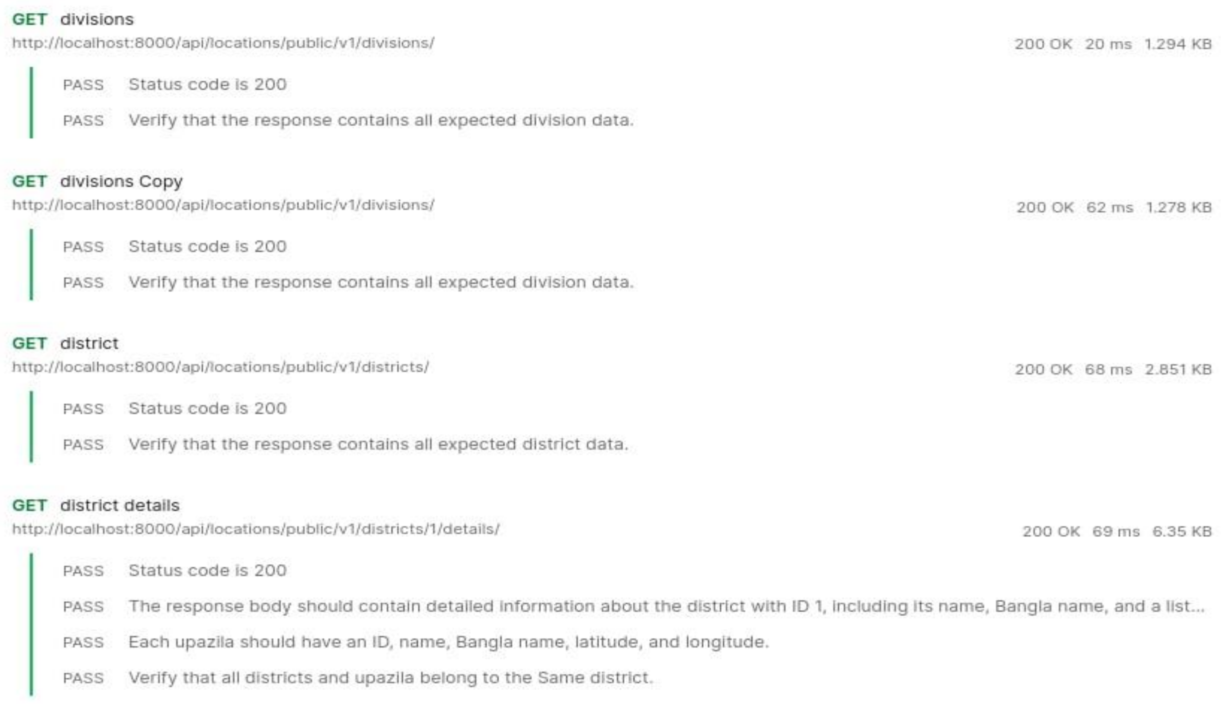


```
POST  register
http://localhost:8000/api/accounts/public/v1/signup/                201 Created   932 ms   1.095 KB
    PASS   Status code is 201
    PASS   Check JSON structure

POST  login
http://localhost:8000/api/accounts/public/v1/login/                 200 OK   604 ms   1.091 KB
    PASS   Status code is 200
    PASS   Check JSON structure and token

POST  login wrong username
http://localhost:8000/api/accounts/public/v1/login/                 400 Bad Request   425 ms   802 B
    PASS   Status code is 400
    PASS   Wrong usename

POST  login incorrect password
http://localhost:8000/api/accounts/public/v1/login/                 400 Bad Request   487 ms   796 B
    PASS   Status code is 400
    PASS   Incorrect password
```



```
GET  get users
http://localhost:8000/api/accounts/public/v1/users/                 200 OK   52 ms   1.572 KB
    PASS   Status code is 200
    PASS   Verify response structure and data types

GET  get user info
http://localhost:8000/api/accounts/public/v1/user-info/             200 OK   68 ms   883 B
    PASS   Status code is 200
    PASS   Verify response structure and data types
```

5.2.Location API



```
GET  divisions
http://localhost:8000/api/locations/public/v1/divisions/                 200 OK  20 ms  1.294 KB
    PASS    Status code is 200
    PASS    Verify that the response contains all expected division data.

GET  divisions Copy
http://localhost:8000/api/locations/public/v1/divisions/                 200 OK  62 ms  1.278 KB
    PASS    Status code is 200
    PASS    Verify that the response contains all expected division data.

GET  district
http://localhost:8000/api/locations/public/v1/districts/                 200 OK  68 ms  2.851 KB
    PASS    Status code is 200
    PASS    Verify that the response contains all expected district data.

GET  district details
http://localhost:8000/api/locations/public/v1/districts/1/details/       200 OK  69 ms  6.35 KB
    PASS    Status code is 200
    PASS    The response body should contain detailed information about the district with ID 1, including its name, Bangla name, and a list...
    PASS    Each upazila should have an ID, name, Bangla name, latitude, and longitude.
    PASS    Verify that all districts and upazila belong to the Same district.
```

GET district details

http://localhost:8000/api/locations/public/v1/districts/1/details/ 200 OK 69 ms 6.35 KB

PASS Status code is 200

PASS The response body should contain detailed information about the district with ID 1, including its name, Bangla name, and a list...

PASS Each upazila should have an ID, name, Bangla name, latitude, and longitude.

PASS Verify that all districts and upazila belong to the Same district.

GET upazilas

http://localhost:8000/api/locations/public/v1/upazilas/ 200 OK 143 ms 4.125 KB

PASS Status code is 200

PASS Verify that the response contains all expected upazila data.

GET upazilas details

http://localhost:8000/api/locations/public/v1/upazilas/1/details/ 200 OK 74 ms 7.515 KB

PASS Status code is 200

PASS The response body should contain detailed information about the upazila with ID 1, including its name, Bangla name, and a list...

PASS Each union should have an ID, name, Bangla name, latitude, and longitude.

PASS Verify that all upazilas and union belong to the Same upazila.

GET unions

http://localhost:8000/api/locations/public/v1/unions/ 200 OK 110 ms 5.305 KB

PASS Status code is 200

PASS Verify that the response contains all expected union data.

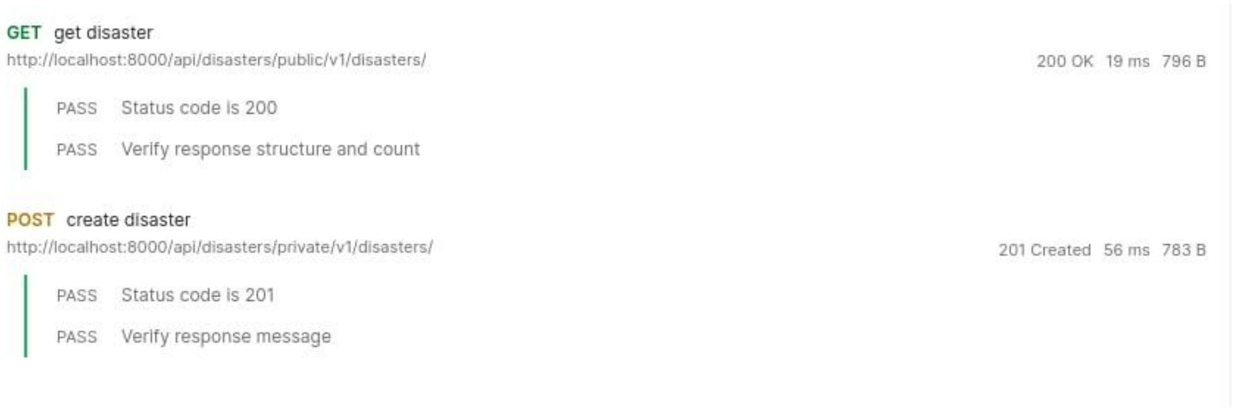
GET union details

http://localhost:8000/api/locations/public/v1/unions/1/details/ 200 OK 65 ms 1.162 KB

PASS Status code is 200

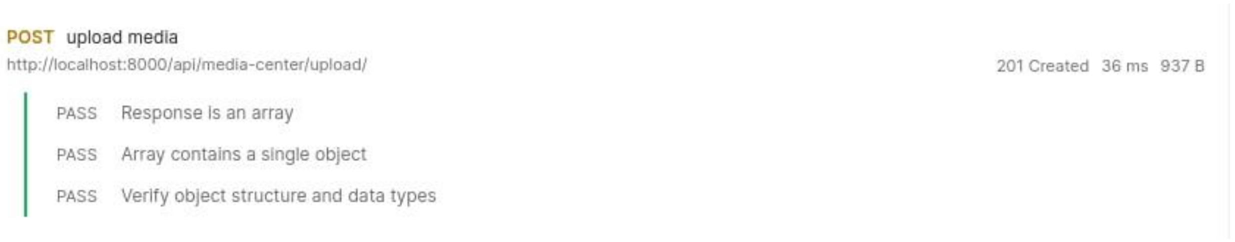
PASS The response body should contain detailed information about the upazila with ID 1, including its name, Bangla name.

5.3.Disaster API



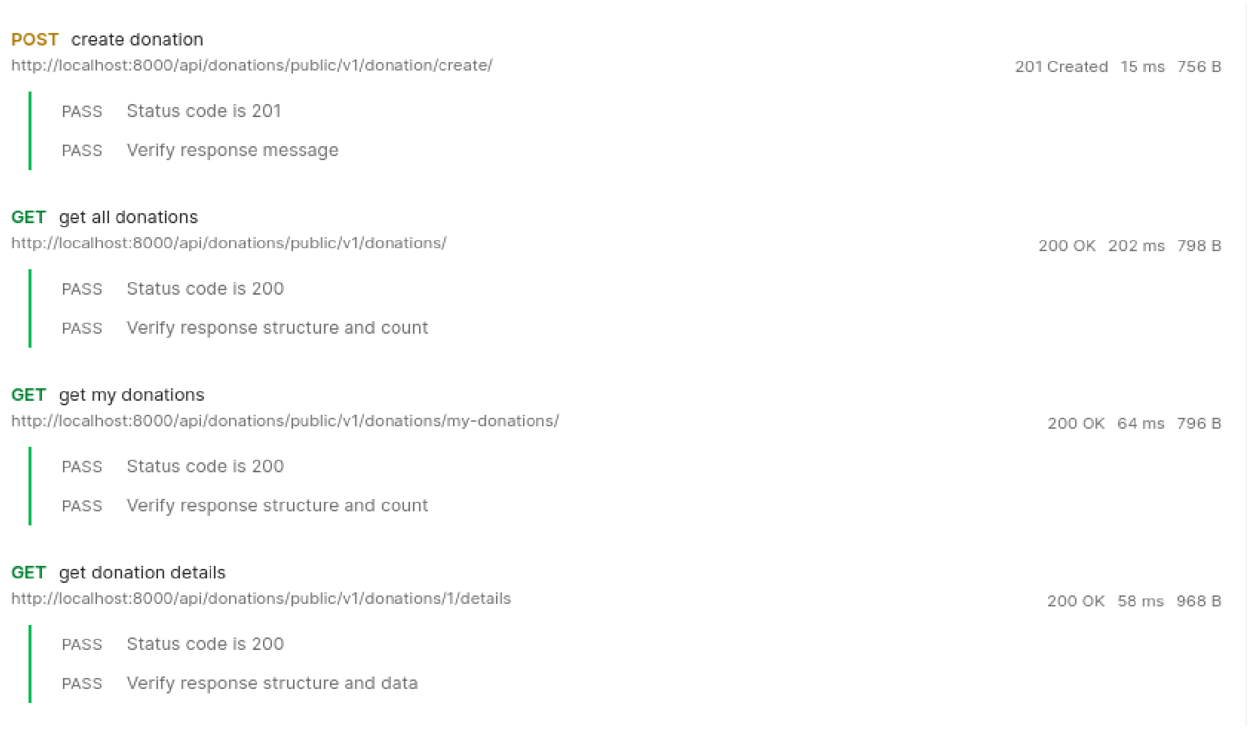
```
GET  get disaster
http://localhost:8000/api/disasters/public/v1/disasters/          200 OK  19 ms  796 B

     PASS   Status code is 200
     PASS   Verify response structure and count

POST  create disaster
http://localhost:8000/api/disasters/private/v1/disasters/    201 Created  56 ms  783 B

     PASS   Status code is 201
     PASS   Verify response message
```

5.4.Media Upload API



```
POST  upload media
http://localhost:8000/api/media-center/upload/               201 Created  36 ms  937 B

     PASS   Response is an array
     PASS   Array contains a single object
     PASS   Verify object structure and data types
```

5.5.Donation API



```
POST  create donation
http://localhost:8000/api/donations/public/v1/donation/create/          201 Created  15 ms  756 B
    PASS   Status code is 201
    PASS   Verify response message

GET  get all donations
http://localhost:8000/api/donations/public/v1/donations/                200 OK  202 ms  798 B
    PASS   Status code is 200
    PASS   Verify response structure and count

GET  get my donations
http://localhost:8000/api/donations/public/v1/donations/my-donations/   200 OK  64 ms  796 B
    PASS   Status code is 200
    PASS   Verify response structure and count

GET  get donation details
http://localhost:8000/api/donations/public/v1/donations/1/details       200 OK  58 ms  968 B
    PASS   Status code is 200
    PASS   Verify response structure and data
```

6. Conclusion

In conclusion, our testing efforts for Relief Tracker have been crucial in ensuring the reliability and effectiveness of the platform. Through alpha testing, we identified various bugs and usability issues, allowing us to prioritize fixes and improvements to enhance the user experience. Our rigorous approach, combining exploratory and scenario-based testing, enabled us to uncover critical, major, and minor issues across key functionalities such as user registration, disaster report creation, donation submission, and map functionality. Moving forward, addressing these findings will be our priority to ensure that Relief Tracker fulfils its purpose of facilitating efficient and transparent disaster relief efforts. Additionally, while beta testing remains pending, we anticipate leveraging direct feedback from real users to further refine the platform and prepare it for deployment to our valuable users and stakeholders.

7. References

Myers, G. J., Sandler, C., & Badgett, T. (2011). *The art of software testing*. John Wiley & Sons.

Orso, A., & Rothermel, G. (2014). Software testing: a research travelogue (2000–2014). In *Future of Software Engineering Proceedings* (pp. 117-132).

Singh, S. K., & Singh, A. (2012). *Software testing*. Vandana Publications.

Spillner, A., & Linz, T. (2021). Software Testing Foundations: A Study Guide for the Certified Tester Exam-Foundation Level-ISTQB® Compliant. dpunkt. verlag.