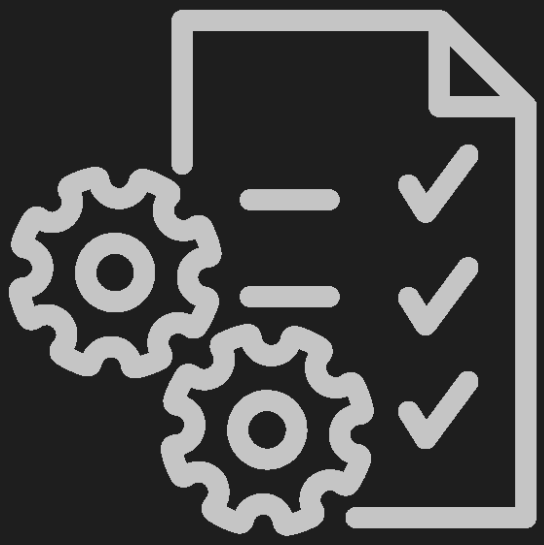
INFOSAFE

Testing Policy



InfoSafe

Seed Analytics

Team Name: FrAgile

Team Members: Christof Steyn
Chris Mittendorf
Karel Smit
Yané van der Westhuizen
Alistair Ross

Team Contact: fragile.cos301@gmail.com

Table of Contents

Introduction

Nowadays software applications and systems have become integral to almost every aspect of our lives. Software testing is the process of evaluating software to ensure that it meets the intended requirements, functions correctly, and operates without errors. It also allows us to identify defects, errors, or inconsistencies and to ensure that the code functions according to specified requirements.

Software testing ensures the quality and reliability of software products. It helps identify and eliminate defects, ensuring that the software performs as expected. It also helps to mitigate risk and thus protects an organisation's reputation. As stated in many of our supporting documents, security is crucial to the operations of our system. Software testing ensures that software complies with industry standards and regulatory requirements. It also helps identify security vulnerabilities and weaknesses, reducing the risk of data breaches.

Procedure for Testing

In order to meet industry standards we have tried to maximise our testing across the system. This has been in order to identify any defects or mishaps in integration. We have tests for all pages in the frontend test the loading, functions and actions using the React and Jest frameworks. A linter was also set up on the frontend pages to make sure that all aspects of the frontend are uniform across the system to improve readability.

There are also tests set up for the backend using JUnit which is a java testing framework. Tests have been set up to test functions and operations in order to ensure they are operating correctly and return the correct type of data.

For our integration, we have our end-to-end (e2e) testing which tests the flow of data through the system from the frontend, through the backend and into the database. The Cypress test framework was used for our e2e testing.

Lastly, we have used a few testing frameworks to test some of our non-functional requirements like availability, security, performance etc. This helps us improve the overall usability of the system as well as increase the security. These tests take the form of some online test frameworks to measure the overall uptime of the website and its performance, checking our security against industry standards and user testing to gain feedback and recommendations from the users perspective.

Testing Tool Choices

Frontend - React and Jest

Due to the frontend of our system being of javascript React, it was easy to make the choice of using React and Jest testing frameworks. React testing was used to make sure pages generate as intended and that components are correctly generated. Jest was then used to test these components with mock data to make sure they are operating correctly and are returning the correct data in the correct format.

Backend - JUnit

Once again it was very easy to choose and justify the testing tool for the backend. Our system is a Java Spring Boot application and so the JUnit test framework, which is a Java Testing framework, was easily integrated into our system. JUnit tests were set up to test the functionality of our backend components and operations to ensure that the correct routing and data is returned from the database. Mock data was also used to test the functionality of all operations

Integration and end-to-end (e2e) - Cypress

Cypress is a javascript testing framework which is why it was the perfect choice for our system that uses a javascript frontend and a java backend. As Cypress was designed

primarily for e2e testing we thought it to be the best framework choice for this aspect of our system testing. Cypress also tests in an actual browser, unlike many other testing frameworks, this simulates a real user experience and what can be expected when using the system. Cypress allows us to test navigation, button clicks, entering data and expecting returns through the system. Cypress was set up to test our flow through the system to ensure all aspects work and that the database and backend work correctly to return the correct data and functionality to the frontend.

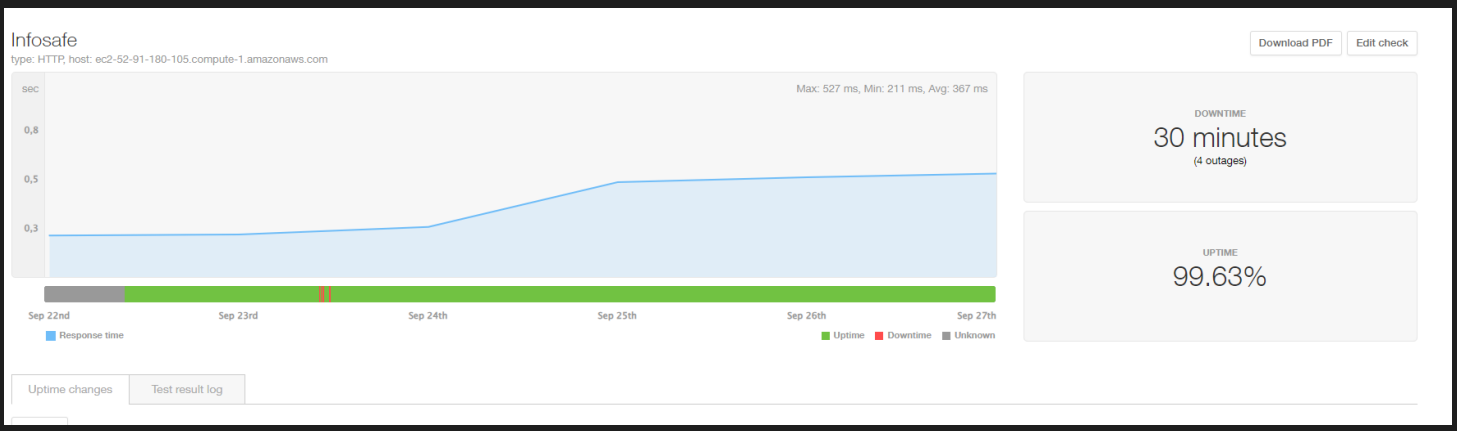
Non-Functional Testing

Availability Testing - Pingdom

Availability testing is a technique that measures the time a system is available to be used. For this test we have opted to use Pingdom which is an availability and testing tool that is available online here. Pingdom makes a whole host of testing options available but in this case we are using availability. This test checks the total uptime of the system and checks for any downtimes as well in order to give you an availability percentage. It continuously monitors the site to calculate the availability percentage using the following formula:

$$availability = \frac{uptime}{uptime + downtime}$$

The availability report is shown here from Pingdom:



Security Testing - OWASP Principles

Security is a very important aspect to our project to ensure privacy of user data and the secureness of documents and information stored in the system. OWASP (The Open Web Application Security Project) is an online foundation that provides guidance on how to develop and maintain trustworthy and secure systems. It outlines 10 of the most common application risks and gives recommendations on how to mitigate these risks. We have outlined the OWASP top 10 below and how we have tried to mitigate these risks:

1. Broken Access Control

   Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorised information disclosure, modification, or destruction of all data or performing a business function outside the user's limits. To reduce this we have implemented various security systems and policies. Only the System Administrator may create new users and then are only granted access. Two factor authentication is required to change a user's password and security tokens are used throughout the system that have expiration times when a user becomes dormant.

2. Cryptographic Failures

   Also known as Sensitive Data Exposure, which is more of a broad symptom rather than a root cause, is the focus on failures related to cryptography (or lack thereof), which often lead to exposure of sensitive data. We have used GitHub secrets to maintain sensitive configuration data in terms of workflows and our CI/CD pipeline so no unauthorised development or changes can take place. We are also using the AWS secrets manager to keep all login credentials secure for our security tokens and access to the database.

3. Injection

   Injection: Injection attacks occur when untrusted data is sent to a code interpreter, such as a database or SQL query. This can allow attackers to execute arbitrary code or access sensitive data. To mitigate this we have frontend and backend tests that check for specific data to be entered and will return errors otherwise.

4. Insecure Design

   Insecure design flaws can occur at any stage of the software development lifecycle, from requirements gathering to implementation. Common examples include insecure session management, broken access control, and insecure direct object references. As stated before, throughout the system users will need a session security token to allow access in the system.

5. Security Misconfiguration

Security misconfigurations can occur in any part of the system, including the web server, application server, database, and operating system. Common examples include insecure default configurations, weak passwords, and outdated software. We have implemented JWT security throughout our system which is a community sourced security injection. It helps manage session tokens and system authorisation.

6. Vulnerable and Outdated Components

   Third-party components, such as libraries and frameworks, can introduce security vulnerabilities into web applications. It is important to keep components up to date and to patch or remove vulnerable components as soon as possible. All our dependencies and libraries are kept up to date as our project uses Maven dependencies and we are notified as soon as any dependency becomes outdated. We are then easily able to update these libraries to ensure the most up to date versions.

7. Identification and Authentication Failures

   Identification and authentication failures can occur when users are not properly authenticated or when users are able to access resources that they are not authorised to access. Common examples include weak password requirements, lack of multi-factor authentication, and session fixation attacks. As stated before authorisation is secured using specific policies and security tokens. Passwords also have to meet specific criteria to be accepted.

8. Software and Data Integrity Failures

   Software and data integrity failures can occur when software or data is modified without authorization. This can allow attackers to tamper with data or execute malicious code. We have a private repository where our system is deployed on and so only authorised users may make changes to online code. GitHub secrets and an AWS Secrets Manager also restrict the access unless the correct credentials are provided.

9. Security Logging and Monitoring Failures

   Security logging and monitoring failures can make it difficult to detect and respond to security incidents. It is important to log all security-relevant events

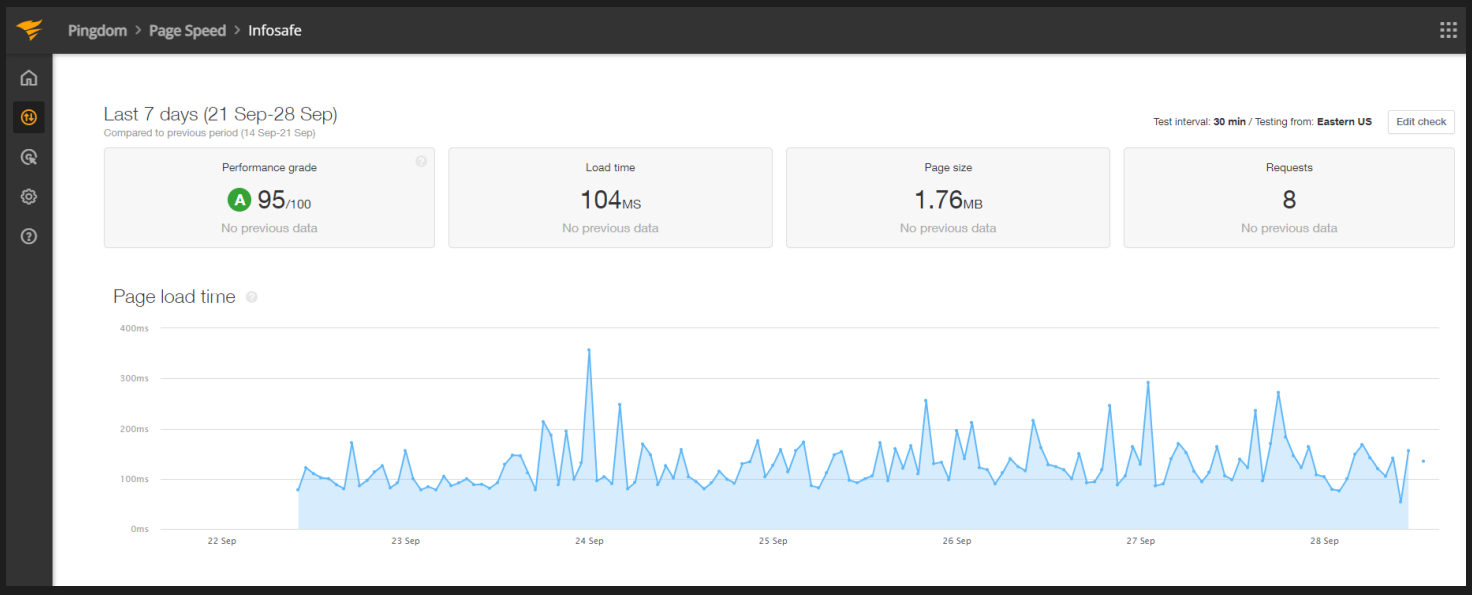
and to monitor logs for suspicious activity. Our system does log failures and failed login attempts to an error log that can be viewed to understand the errors.

10. Server-Side Request Forgery (SSRF)

SSRF attacks occur when an attacker is able to force a server to make unauthorised requests to other servers. This can allow attackers to exfiltrate sensitive data or execute malicious code on remote servers. We decided to use AWS as our main hosting server which is regarded to be very secure and well maintained. This should help mitigate the risks of an SSRF attack.

Performance Testing - Pingdom

Performance is another aspect of the system we wanted to focus on. We wanted the user to have a wait-free experience that will help increase productivity and efficiency in the workplace when using our system. For our performance testing we also used Pingdom, which also can test a systems performance. Using this tool we were able to identify the page loading times as well as page sizes to see if larger pages hinder the systems performance. Pingdom will also be able to give us insights on our users as well as eventually determine the maximum number of users that can use the system concurrently. Below is a report on the website's performance from Pingdom that assessed the performance of our system using the aid of the YSlow tool. As you can see the system performance has an A grade of 95/100 for performance.



Usability Testing - User Testing and Feedback

We were able to complete our usability testing by allowing individuals to use our system by giving them access credentials. This also helped generate better reports on Pingdom as it was able to measure speeds and uptimes when users were on the system. Users were asked to give us feedback on their experience and recommendations for changes and improvements they would like to see in the system. Appendix A has the feedback and recommendations for you to see.

Automated Tests

Our automation takes place on specific triggers stated in our GitHub workflows. We have four workflows for deployment, linting, unit tests and cypress tests. The deployment workflow simply builds the system and deploys it to our hosting server and won't be discussed further here.

The Linter workflow executes the linter to test for uniformity across the system to improve readability and generates a report of inconsistencies if the linter fails.

The Unit-Tests workflow runs automated tests across the frontend with mock data to make sure all pages generate correctly and that functions operate as expected. This workflow will be automatically triggered on a push or pull-request to any branch in the system.

The Cypress-Test workflow runs automated integration and e2e tests across the system. The workflow waits 45 seconds for the system to boot up before beginning the tests in the system. This workflow will also be automatically triggered on a push or pull-request to any branch in the system.

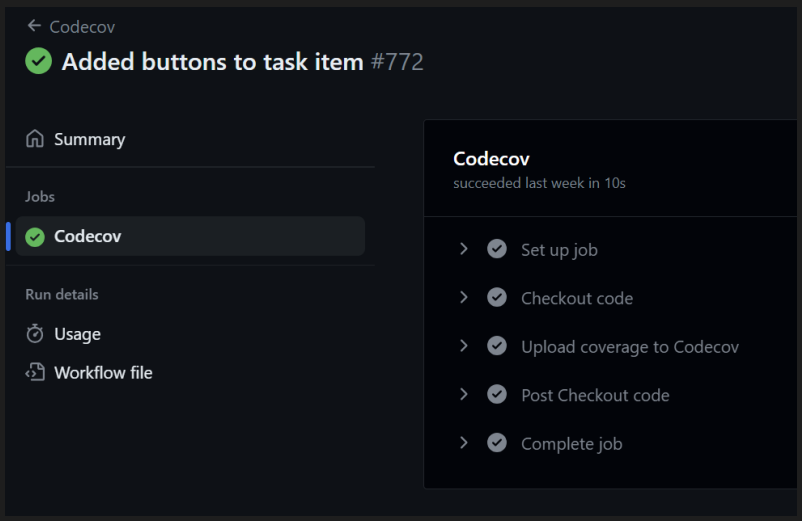
Reports

Below you can find reports for our various tests. Links will be provided to our GitHub actions to view test history as well as images for the latest report to view the execution of the workflows.

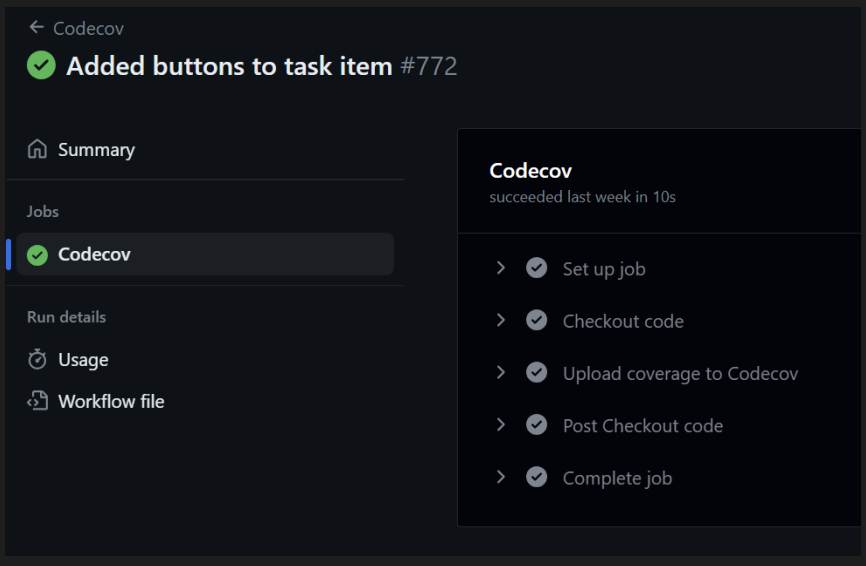
Codecov

Codecov was to check that we were covering mosting of the system and that tests were created for all aspects of our project. Our Codecov GitHub workflow can be found here and you can view our Codecov repo here. Below are screenshots of the latest statistics.

GitHub Repo:

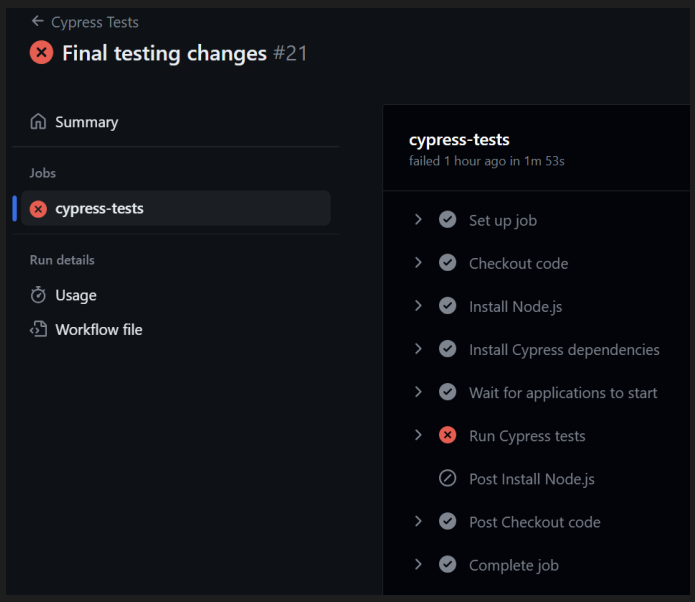


Codecov Repo:



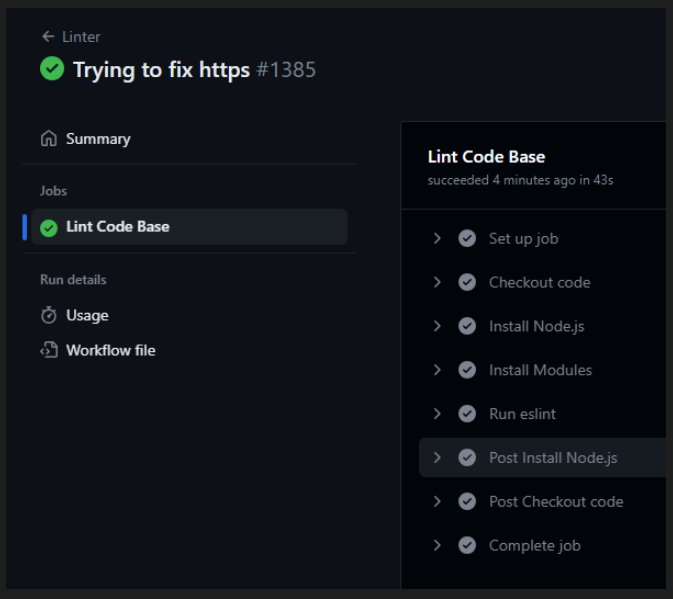
Cypress Tests

Our Cypress test history can be found on our GitHub repository here. Below is a report of the latest workflow:



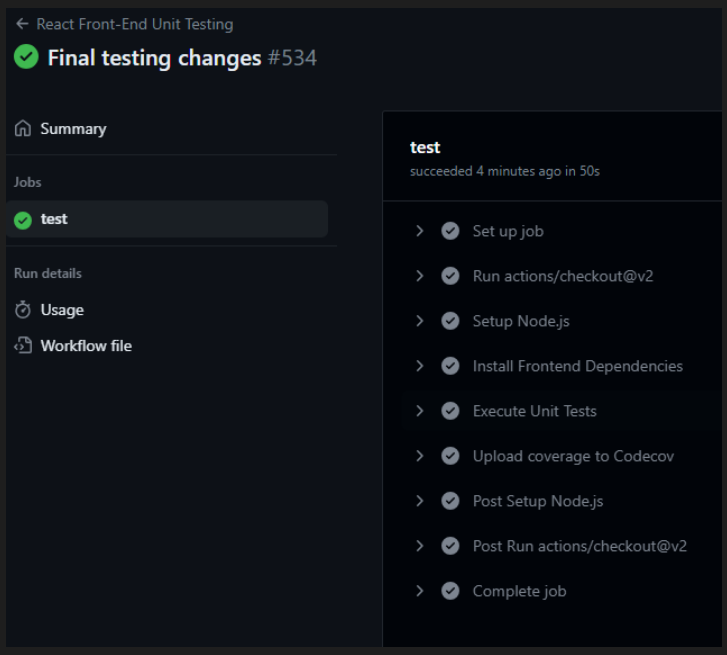
Linter

A report for our linter can be found here on our GitHub repository workflows. The latest report can be viewed below:



React and Jest Frontend Testing

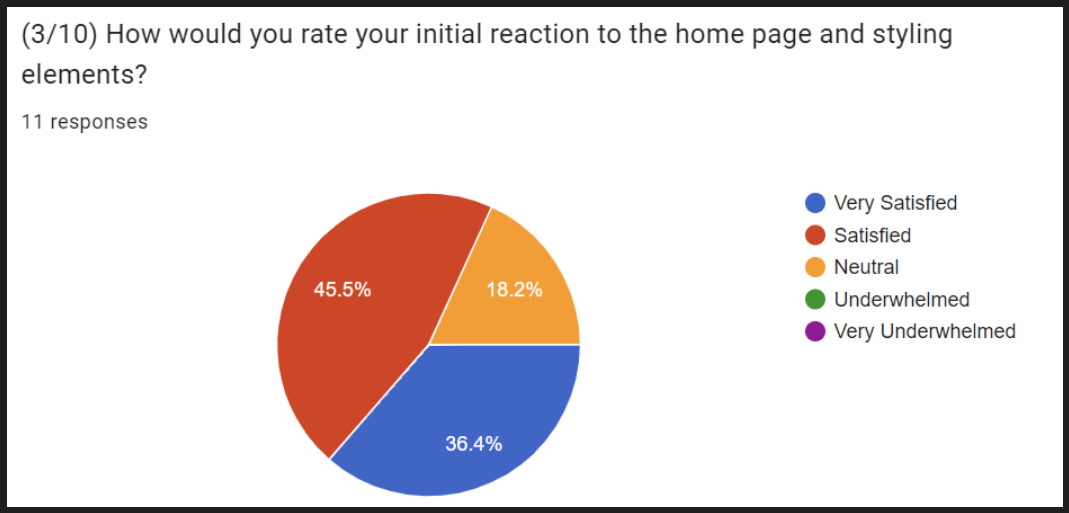
Our frontend tests were mostly created using React and Jest and the history can be viewed here on our GitHub repository workflows and a report of the latest test run can be viewed below:




```
281  Test Suites: 27 passed, 27 total
282  Tests:       50 passed, 50 total
283  Snapshots:   0 total
284  Time:        17.793 s
285  Ran all test suites.
```

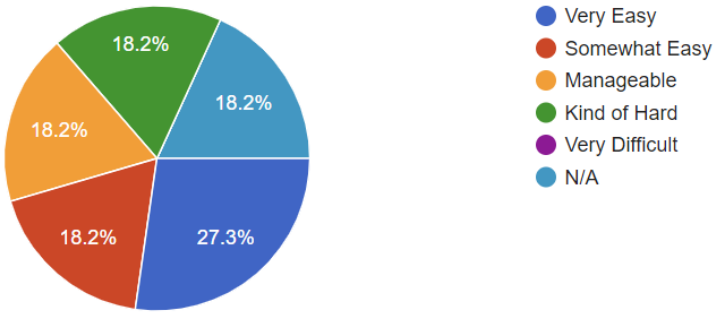
Appendix A

Below are charts and graphics with responses from feedback. We received some mixed reviews about the styling and colour palette. Lots of people enjoyed the home page and the breakdown of the statistics. There were a few comments not understanding what the system was and a few recommendations to improve the user experience with some personalisations and error/confirmation logs. The rest of the data can be view below, please not the first question was just asking for permission to use their responses in the report and the ninth question was written recommendations:



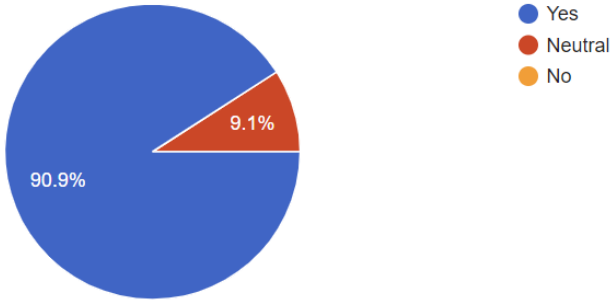
(4/10) How difficult did you find it easy to create entities (Users, Devices, Datascopes, Tasks etc.) (If your user role did not had access for creation, please select N/A)?

11 responses



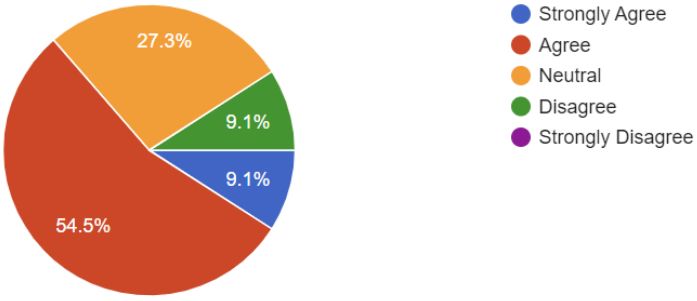
(5/10) Did you find it easy to use the navigation bar and move through the system?

11 responses



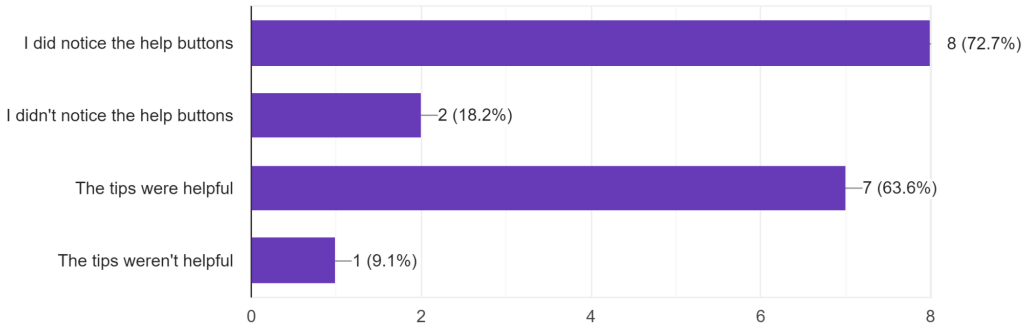
(6/10) Was the layout and elements in the system aesthetically pleasing and easy to use?

11 responses



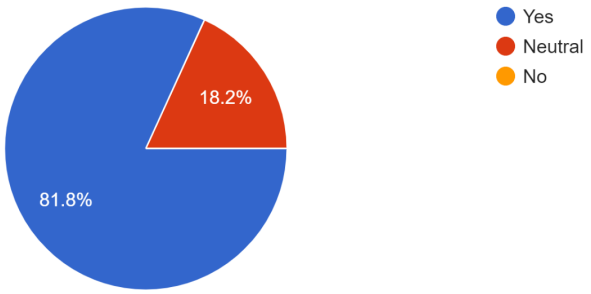
(7/10) If you had any problems did you notice the help buttons on the page?

11 responses



(8/10) Was the system performance (speed and loading) satisfactory? (You didn't have to wait too long)

11 responses



(10/10) How would you rate you overall experience from 1 to 10 ?

11 responses