Testing Report Shotmaniacs Team 1

In this report there will be descriptions and explanations about the methods, results, and implications obtained after doing the software and user story testing in our test-driven developmentA picture containing sketch, drawing, origami, design

Description automatically generated

# Table of Contents

[Table of Contents 2](#_heading=h.30j0zll)

[Postman – Application Testing 3](#_heading=h.3znysh7)

[JUnit Testing – Unit Testing 3](#_heading=h.2et92p0)

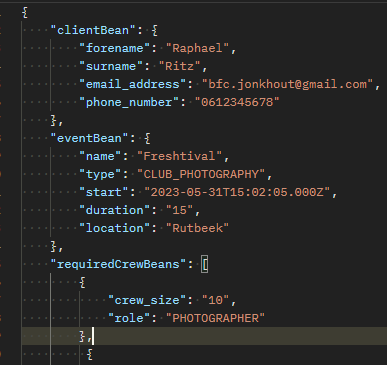
[Manual Testing 3](#_heading=h.tyjcwt)

[SonarQube – Static Application Security Testing 4](#_heading=h.3dy6vkm)

[Conclusion 4](#_heading=h.1t3h5sf)

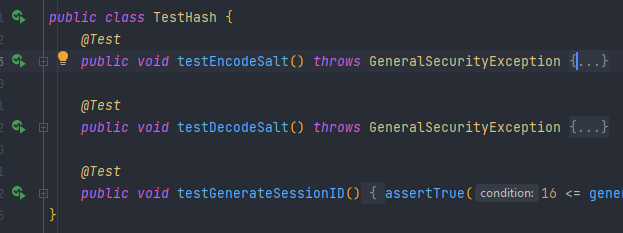
# 

# Postman – Application Testing

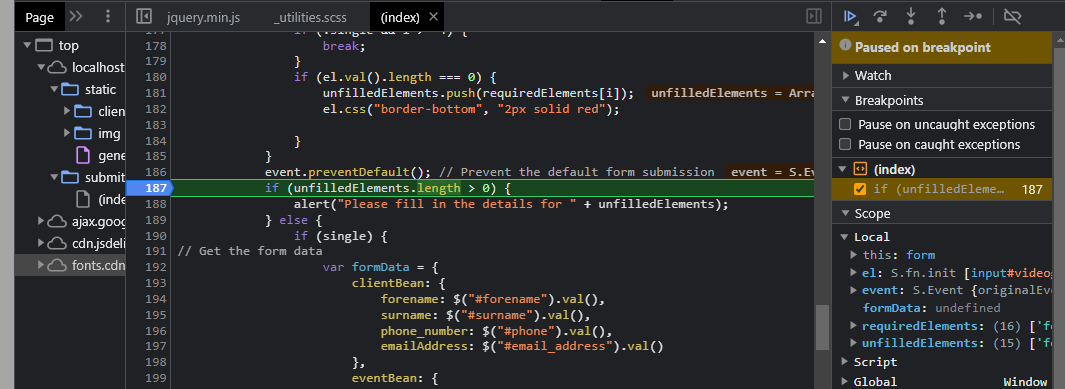
The first tool that was used in this project. This tool is probably the tool that was used the most within the project. For each request, a postman request was created. It was generally easy to create a collection of requests, one for logging in, one for logging out. This way, it could easily check for correct cookies, but it also was possible to see the error on the server. When starting up the tomcat server, it was easy to see what the error message was and how to fix it. 

Postman also worked as a software contract for us. Together, you could specify the form of the JSON body and the one person would start on the front-end and the other on the back end. This tool allowed for good team synergy.

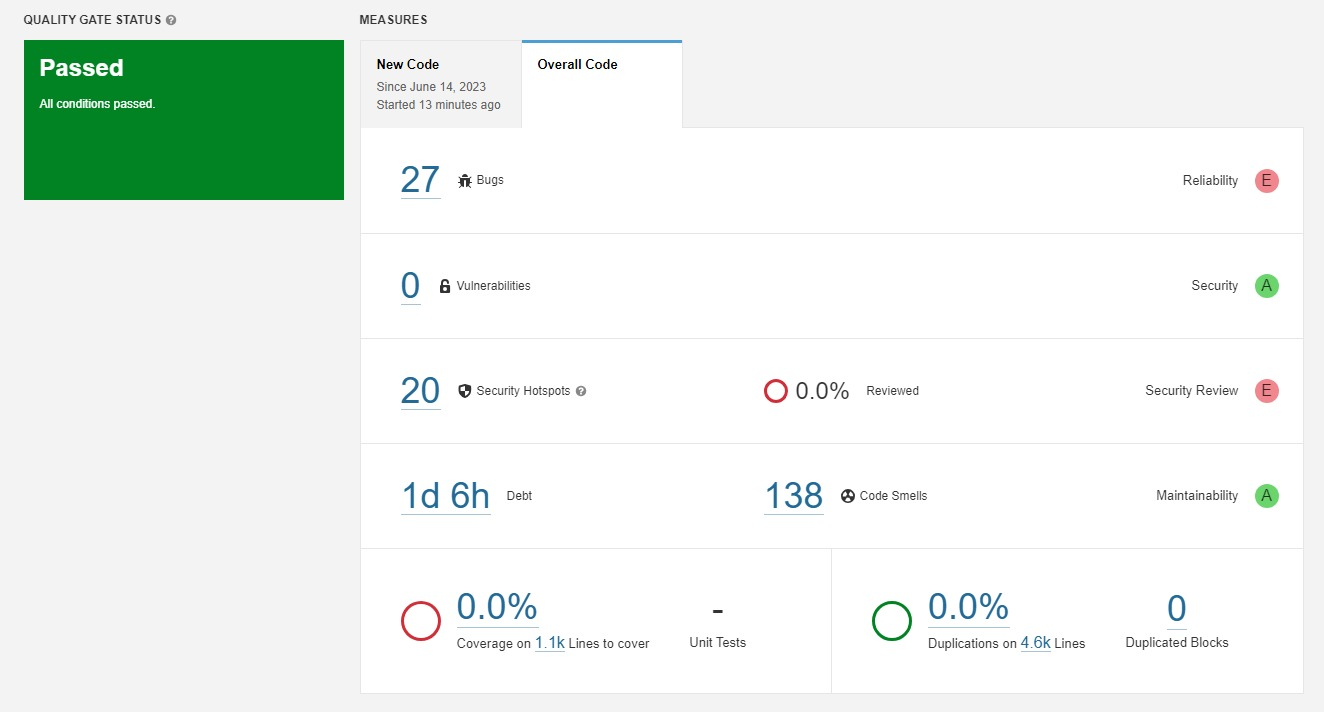
# JUnit Testing – Unit Testing

The second tool that was used were unit tests. During the development, it felt natural to write a main method to compare the expected result with the actual result. However, it was quickly noticed that this was not a correct way to do so. Because this is precisely what unit tests are supposed to do. As a result, it was decided to write unit tests whenever we needed to test something, we could rely on the unit tests. This way, it was easy to know what the expected value was and how it works. Using this in line with the debugging tool of IntellIJ, it provided a lot of room for correct development. 

# Manual Testing

The third tool that was used was manual testing. This included setting up the server and just pressing buttons whether this works. This type of testing is essential, and it allows us to also work with front-end development testing. To provide more information during the manual testing, we used the debugging tool and breakpoints in the browser as well. 

# SonarQube – Static Application Security Testing

For those who do not know it, the fourth we have used is SonarQube. SonarQube is a code quality tester, which scans for possible security flaws. This testing tool falls in the category of static application (security) testing. It can also be used in combination with CI/CD on GitLab. But for this project, it was believed that this was outside of the scope.   
The result of this tool is very user friendly. Since the time we have installed was a little late, it showed that we needed to work on our consistency. A lot of the errors and coding issues it noticed were based on incorrect variable naming and other relatively small threats. However, it also showed us a security issue. It showed that we used an insecure random number generator. This is taken care of within the security branch. 

# Conclusion

The testing of the back end is mostly done by Unit tests. The connectivity between back-end and front-end is mostly done by Postman. The front-end testing is mostly done by the debugging tool and manual testing. And generally, the full 6k+ lines of code are scanned by SonarQube.