Test plan

MeldAPp

Contents

Version Control

N°	Date	Distribution	Changes
0.01	2020-05-08	(first draft) Risk analysis + test strategy + stakeholders	
0.02	2020-05-15	Project description + introduction (draft) + adjustments feedback	
0.03	2020-05-15	Adjustments feedback and clean-up document	
1.00	2020-05-22	Team MeldAPp	Last small adjustments + lay-out

Terms and abbreviations

Term	Description
AP	Artesis Plantijn University College
ELL	Campus Ellermanstraat
NOO	Campus Noorderplaats
OS	Operating System
PO	Product Owner

Introduction

This document is meant to describe the testing scope of our application and the strategies that will be used to make sure the application is developed according to its requirements.

This test plan contains information which is interesting for our development team and the PO, such as workhours, software, risk, contingencies, etc.

This way we want to win the trust of the PO and make sure our team of developers won’t put more effort than needed into testing different functionalities, because they will have a clear guideline of the testing scope and process. Furthermore, less unforeseen issues will occur and answers to possible issues have already been predefined, which will boost the team’s efficiency even more.

Our application is divided into three layers:

- The front-end

- The middleware
- The database

A lot of things can go wrong, not only in these layers, but also between these layers.
To make sure our application is bug free and secure, we must:

- Test the connection between the database, middleware and front-end
- Test if the business logic is correctly implemented in the units
- Test the security

We also need to test the connection between the application and Azure.

Once the application itself and its connection with other services has been tested, we will test how well the AP employees can work with the application by performing user acceptance tests.

In this test plan a lot of software tests are described. For the tests that will be done, there is an explanation on how the test will be performed, how you can see if the test was successful or unsuccessful and what the tests covers.

Project description

Even though the campuses ELL and NOO are relatively new, the AP University College students and staffs encounter various defects on a daily basis. The management of defects is unstructured and happens via different channels (verbally, via e-mail, via calls, post-its etc.).

Because there is no structure and it’s not clear if a certain defect has already been reported or not, some defects are reported multiple times or won’t be reported at all. Some of the defects that are reported, are at times forgotten or lost. Same story for tasks (e.g. requested catering for a meeting).

To solve this problem, the students of the second year of Applied Informatics were asked to develop an application in small groups.

The goal of the project is centralizing the management of the defects (and tasks) for the two campuses, with the option of expanding the app for other campuses of the AP University College in mind. The application will reduce the duplicate notifications about the same defect. It also reduces the chance of human errors. In the application, all AP employees can easily report defects. Next to defects there is also the possibility to submit tasks by the head of education, facility/logistic administrators and super administrator. Once solved (or cancelled), the defects and tasks will be archived to make later revision possible. The whole application together will be a clear system that lightens the workload of the facility/logistic service and the head of service.

List of functionalities:

- Login
- Overview
- Create defect/task
- Manage defect/task
- Assign defect/task
- Cancel defect/task
- Archive defect/task
- Ask questions to the notifier

- Agenda to plan repair date
- Export to pdf
- Emergency number
- Assign roles

Stakeholders

Name	Contribution/Role
The stakeholders of this project	

Name	Contribution/Role

Risk Analysis

Description	Risk	Impact	Prior.
The system crashes and no backups have been made; the project cannot be fully recovered. We have lost a lot of progress which is a major setback. Motivation is gone and so is all the time we’ve already spent.	5	5	5

Action	Action type
We use Gitlab’s version control to be on the safe side of all things. When something bad happens, we can always go back to the latest save, which means we will never lose ‘everything’. Even better, we can also see a summary of actions and changes and who performed them.	Avoiding

Use Gitlab as version control system.

Description	Risk	Impact	Prior.
Code works locally but doesn’t work when published, even though all unit-tests were performed, and they all succeeded.	3	5	3

Action	Action type
We use Gitlab to push our code. Gitlab’s push function acts like a smoke test to be sure if the deployed build works fine. We will also test the published project manually. If any issues occur, we will replace the build with the previous one and try solving the error.	Limiting

Always test published code to avoid surprises during Sprint Reviews and after the Go-Live.

Description	Risk	Impact	Prior.
A single piece of code which is a precondition for multiple other tasks does not work properly. This results in multiple functionalities covered in errors.	3	5	3

Action	Action type
Predefine the most important pieces of code, which is probably the most used code as well. Write integration tests, manually test and triple check if it works, then refactor it into a component to make it reusable and then test it again. This is the reason why we use React (component-based language) as front-end language in the first place.	Limiting

Refactor pieces of code into components so they can be reused, and errors will be centralised. Then test the components.

Description	Risk	Impact	Prior.
Students can login (with their AP-email or their personal mail) and do what every employee can do. Students can abuse the application by e.g. entering fake data or spamming inappropriate comments.	3	5	3

Action	Action type
Check at the login if the user is a student or employee by checking if there is a p or s in front of the personal user id of their personal email. Based on that first character we can allow or deny access to users. This will also be manually tested.	Avoiding

Be sure to check whether only the privileged users can use the app or not.

Description	Risk	Impact	Prior.
Software used in the project (e.g. included CSS / JavaScript links) becomes temporarily unavailable due to unknown circumstances or due to user interactions. Some parts of the application are now broken an cannot be used anymore.	3	3	2

Action	Action type
Use open source programs as much as possible. Inspect statistics, documentation and community for each software before using it. There will be a message when users visit the website that tells them which software is needed for the system to run. It’s the user’s responsibility to have the basic software installed and running (e.g. javascript).	Limiting

Research multiple software before using them. Inform the user about needed software.

Description	Risk	Impact	Prior.
An employee can access pages and functions which should only be available for the administrator. This indicates that users aren’t correctly assigned to predefined roles.	3	4	2

Action	Action type
Work with role authorization and test it for both front-end and middleware. In front end this will be tested manually, and we will use unit tests in middleware.	Avoiding

Test all roles to be sure everyone has the right privileges.

Description	Risk	Impact	Prior.
Written code does not work on all OS/ browsers. Some functions are broken. Therefore, our system is not optimized and is not 100% applicable to the customer’s wishes.	2	5	2

Action	Action type
Manually test every function BEFORE changing the status to “Done”, using different browsers / platforms. Before putting the code into production, run the written tests multiple times switching browsers / platforms. Also, cross-browser and cross-system tests will be foreseen to be sure everything works as expected.	Avoiding

Test every tiny piece of code to be sure everything works. Use different browsers and platforms to be sure the application is fully operational and available for everyone.

Description	Risk	Impact	Prior.
The integration with the client’s already existing hardware fails at the kick-off. Some functionalities need to be revised. (e.g. the integration with the client’s existing file server fails so the images cannot be stored)	2	5	2

Action	Action type
Test the application when it’s running on the client’s hardware before the kick-off. Be sure to save some space for error-fixing.	Limiting

The application needs to be tested while running on the client’s hardware to make sure everything works as expected.

Description	Risk	Impact	Prior.
The system crashes when too many users use it at the same time. Data has been lost, time is wasted, and the client is not happy.	2	5	2

Action	Action type
Performance tests will be provided to check how the program deals with usage-spikes. Load balancers will be attached; they will split the pressure to multiple servers.	Avoiding

Add load balancers and performance tests to ensure the working of the application, even when the pressure builds up high.

Description	Risk	Impact	Prior.
When posting a new defect, the user tries to upload an image that is too big (over 10mb). An error occurs, system crashes, both data and valuable time are lost.	3	3	2

Action	Action type
The user will see a clear error message that explains that the image is too big. This does not break the application; the user can choose another image.	Limiting

Clear error communication so the user knows what happened.

Description	Risk	Impact	Prior.
When posting a new defect, the user tries to upload a file which is not an image type.	3	3	2

Action	Action type
The user will see a clear error message that explains that the file type is not supported, and which file types are. The user has the option to choose another file.	Limiting

Clear error communication so the user knows what happened.

Test strategy

Test type	Planned?	Coverage and criteria
Unit tests	Yes	These are the first tests to be performed because they test the basics of the code. How: Writing automated tests for every piece of code. Successful: If the code is correct, the test passes and succeeds. Unsuccessful: If the code does not meet the system requirements, the test fails. Coverage: Unit tests cover the business logic (in our code they cover the services in the middleware). We will cover the main units.
Integration tests	Yes	Performed after the unit tests because they test the units working together. How: Writing automated tests for every component. Successful: If the units work together correctly, the test passes and succeeds. Unsuccessful: If the units cannot work together, the test fails. Coverage: Different components will be tested. These components exist out of multiple units. We will test every component that has a database table, a piece of middleware and a piece of front-end.
System tests	Yes	System Testing means testing the whole system. All the modules and components are integrated in order to verify if the system works as expected or not. [3] We don’t have a lot of systems talking to each other. The only extern system we have is Azure. This means that most of the system tests are covered by integration tests.

Test type	Planned?	Coverage and criteria
Smoke tests	Yes	Smoke testing will determine whether the deployed build is stable or not. [4] How: In our case, the application will be smoke tested by deploying our software to the virtual private server, using Gitlab. Successful: If the deployment succeeds, the test passes. Unsuccessful: If the deployment fails, the test fails. Coverage: The stability of the deployed build will be tested.
Sanity tests	No	Testing if bug fixes really did fix the bugs and if it didn’t create any new ones. This is covered by unit and integration tests.
Interface tests	No	Interface Testing is defined as a software testing type which verifies whether the communication between two different software systems is done correctly. [6] We don’t need this test because it is covered by the integration tests. Every component has different software systems. So, by testing the different components the different software systems are tested.
Regression tests	Yes	Regression testing is defined as a type of software testing which confirms if a recent program or code change has affected existing features or not. [7] How: Regression tests will be performed by running the unit- and integration tests again. Successful: If all the written tests are successful, the test passes. Unsuccessful: When one of the written tests fails. Coverage: The unit and integration tests of the part of software that has been changed.
User Acceptance tests	Yes	User Acceptance Testing is a type of testing performed by the end user or the client to verify/accept the software system before moving the software application to the production environment. [8] How: We write test criteria where the users will test on. The users are a few AP staffs. Successful: When the end-users are enthusiast and really want to use the application, the test succeeds. Unsuccessful: When the end-users do not like the structure, the looks, the way the application works or something else, the test fails. Coverage: The front-end part of the application.

Test type	Planned?	Coverage and criteria
Performance tests	No	The purpose of Performance Testing is to eliminate performance bottlenecks in the software or device. [9] You can divide performance tests into 6 different tests: load, stress, endurance, spike, volume, scalability. We can't perform the stress and spike tests because it needs to be done on the AP-servers and this is not allowed. The endurance, volume and scalability tests are not necessary for our system because they test the servers more than the system itself. The load tests we will perform and because of this is further explained below.
Load tests	Yes	The load tests check the application's ability to perform under anticipated user loads. The objective is to identify performance bottlenecks before the software application goes live. [9] How: We will use Google Lighthouse to perform these tests. Successful: Google lighthouse has a rating system on which we will validate if the test was successful. Unsuccessful: If the application’s ability to perform is not good enough by the Google lighthouse rating system. Coverage: The whole application if it is accessible with multiple users at the same time and the time that’s needed for loading the application.
Security tests	Yes	How: Online tools for security tests. Successful: If the tools can’t get through the security of the system, the test succeeds. Unsuccessful: If the tools can get through the security and the delicate information is visible, the test fails. Coverage: The security of the system.
Cross-browser and cross-system tests	Yes	To test if the front-end behaves consistent on multiple browsers/operating systems (computer/tablet/mobile, apple/windows/Linux, ...). How: Manual test every time before the end of a sprint (on different browsers). Successful: When the site behaves consistent across browsers/operating systems, the test succeeds. Unsuccessful: When the site behaves different and is no longer ideal, the test fails. Coverage: The front-end part of the application. Every main page of the application.

Test type	Planned?	Coverage and criteria
Usability tests	Yes	Usability tests look at how the end-users use the system and checks if the system is easy to use and if it is user-friendly. How: The participant gets a task and needs to perform the task while the facilitator observes. Afterwards, the participant gives feedback. Successful: When the participants find everything they need under 20 seconds and if they can perform all the objectives they get, the test succeeds. Unsuccessful: When the participant can’t perform all the objectives or if they search longer than 20 seconds to find something, the test fails. Coverage: The front-end part of the application.

Tests which will or won’t be performed.

Tools

Name	Version	Supplier	Description	Ref
Chrome	Versie 81.0.4044.138+ (Official build) (64-bits)	Google	Browser: Google Chrome	[13]
Edge	Microsoft Edge 44.18362.449.0+	Microsoft	Browser: Microsoft Edge	[14]
Firefox	Version 75.0+ (64-bit)	Mozilla	Browser: Mozilla Firefox	[15]
Safari	Versie 13.1 (15609.1.20.111.8)	Apple	Browser: Safari	[16]
Windows	10	Microsoft	OS: Windows 10	[17]
MacOS	MacOS Catalina	Apple	OS: MacOS	[18]
Selenium	Webdriver	Selenium	This is a tool for browser testing	[19]
Google Lighthouse	3.0	Google	This is a tool for performance testing	[20]
Visual Studio Code	1.45.1	Microsoft	IDE	[21]

Name	Version	Supplier	Description	Ref
IntelliJ IDEA	2019.3.2	Jet Brains	IDE	[22]
ZAP	v2.9.0	OWASP	This is a security testing tool	[23]
JUnit	JUnit 4 (v4.8.0+) JUnit 5 (v5.1.0+)	JUnit	This is a tool for unit testing	[24]

Software related to testing, used in this project.

Literature

[1] Software testing help (16-04-2020). Key To Successful Unit Testing – How Developers Test Their Own Code? Collected from https://www.softwaretestinghelp.com/unit-testing/.

[2] Guru99 (2020). Integration Testing: What is, Types, Top Down & Bottom Up Example. Collected from https://www.guru99.com/integration-testing.html.

[3] Software testing help (16-04-2020). What Is System Testing – A Ultimate Beginner’s Guide. Collected from https://www.softwaretestinghelp.com/system-testing/.

[4] Guru99 (2020). What is Smoke Testing? How to do with EXAMPLES. Collected from https://www.guru99.com/smoke-testing.html.

[5] GeeksforGeeks. Sanity Testing | Software Testing. Collected from https://www.geeksforgeeks.org/sanity-testing-software-testing/.

[6] Guru99 (2020). What is Interface Testing? Types & Example. Collected from https://www.guru99.com/interface-testing.html.

[7] Guru99 (2020). What is Regression Testing? Definition, Test Cases (Example). Collected from https://www.guru99.com/regression-testing.html.

[8] Guru99 (2020). What is User Acceptance Testing (UAT)? with Examples. Collected from https://www.guru99.com/user-acceptance-testing.html.

[9] Guru99 (2020). Performance Testing Tutorial: What is, Types, Metrics & Example. Collected from https://www.guru99.com/performance-testing.html.

[10] Guru99 (2020). What is Security Testing? Types with Example. Collected from https://www.guru99.com/what-is-security-testing.html.

[11] software testing help (16-04-2020). What Is Cross Browser Testing And How To Perform It: A Complete Guide. Collected from https://www.softwaretestinghelp.com/how-is-cross-browser-testing-performed/.

[12] Guru99 (2020). What is Usability Testing? UX(User Experience) Testing Example. Collected from https://www.guru99.com/usability-testing-tutorial.html.

[13] Google Chrome (2020). Doe meer met het nieuwe Chrome. Collected from https://www.google.com/intl/nl/chrome/.

[14] Microsoft Edge (2020). Dit is de nieuwe Microsoft Edge. Collected from https://www.microsoft.com/nl-nl/edge/.

[15] Mozilla FireFox (2020). Download de nieuwste Firefox-browser. Collected from https://www.mozilla.org/nl/firefox/new/.

[16] Safari (2020) Safari. The best way to see the sites. Collected from https://www.apple.com/safari/.

[17] Microsoft Windows 10 (2020). Collected from https://www.microsoft.com/nl-be/windows/get-windows-10/.

[18] MacOS (2020). MacOS Catalina. Collected from https://www.apple.com/benl/macos/catalina/.

[19] Selenium (2020). Selenium automates browsers. That's it!. Collected from https://www.selenium.dev/

[20] Google Developers (10-05-2020). Lighthouse | Tools for Web Developers | Google. Collected from https://developers.google.com/web/tools/lighthouse/.

[21] Visual Studio Code (2020). Code editing. Redefined. Collected from https://code.visualstudio.com/.

[22] IntelliJ IDEA (2020). Capable and Ergonomic IDE for JVM. Collected from https://www.jetbrains.com/idea/.

[23] OWASP (2020). OWASP Zed Attack Proxy (ZAP). Collected from https://www.zaproxy.org/.

[24] JUnit 5 (2020). The new major version of the programmer-friendly testing framework for Java. Collected from https://junit.org/junit5/.