Test Plan

Ivanti’s Website

13-01-2022

Version 2

Contents

[Introduction 2](#_Toc92981473)

[Test methodology 2](#_Toc92981474)

[Used approaches 2](#_Toc92981475)

[Unit tests 2](#_Toc92981476)

[Other tests 2](#_Toc92981477)

[Integration tests 3](#_Toc92981478)

[Acceptance tests 3](#_Toc92981479)

[SonarQube 5](#_Toc92981480)

[Conclusion 5](#_Toc92981481)

# Introduction

The Test Plan is designed to test the main functionality of Ivanti’s website project. The tests are broad and not specific as we are still at the start of the project. But they are going to give the main idea of whether the units and API work fine.

# 

# Test methodology

## Used approaches

### Unit tests

We have made unit tests so we can make sure whenever something in a certain unit (class) is changed it won’t break the whole application. Also, to do that we first run the tests and make sure they pass and afterwards we run the application.

* Repositories tests

Example:

A screenshot of a computer

Description automatically generated with medium confidence

* Service tests

Example:

A screenshot of a computer screen

Description automatically generated with medium confidence

* Model tests

Example:

1. Arrange : a @BeforeEach method setting up an Entity with data
2. A screenshot of a computer

   Description automatically generated with medium confidenceAct and Assert

### Integration tests

We have made controller tests in order to verify that all components starting from the API endpoint down to the data layer are working altogether without issues and that the endpoint retrieves the correct response.

A screenshot of a computer

Description automatically generated with medium confidence Example:

### Other tests

To ensure that all the parts of the application are working correctly we have added Repository and Service tests. We have made the repository tests so that we know the communication with the database is successful and we don’t loose any data when sending or retrieving information. Service tests were included so we can make sure that the “middle man” is sending the right packets into the database and getting right information from it.

### Pipeline

As we recently studied about CI/CD we will try to actively include it in our project because this feature is pretty useful for big projects where we have many tests on pieces of code and it will be easier to have a person developing some new feature instead of testing something which can be tested by a computer. Also, unit tests can be run using CI/CD. It is easy to update and maintain the existing pipelines.

### Acceptance tests

User acceptance tests are one of the most expensive as we can say. Thus, it is because we have to gather people who have no idea how the application works and give them instructions on how to use it or just let them explore it on their own. So it is time-consuming but this is also the most efficient way to see whether their unusual/unexpected actions are going to break the app. There we have some steps and expected results.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Test description** | **Step #** | **Test Steps** | **Expected Result** |
| 1 | Verify visitor can create an account | 1 | User has opened the web application | After confirmation the system creates an account |
| 2 | User clicks Register and fills in all the fields |
| 2 | Verify visitors can browse packages | 1 | Visitor clicks/opens “Store” page of the website | System loads all the available packages |
| 3 | Verify visitor can open a page of specific package | 1 | Visitor is on “Store” page | System opens a new page with detailed information about the package |
| 2 | Clicks on a package |
| 4 | Verify logged user can change his account details | 1 | Customer is on profile page | The system is updating his profile details |
| 2 | Clicks change details |
| 3 | Fills all the new information and saves |
| 5 | Verify customer can download a package | 1 | Customer is on specific package page | The system is sending the package to the user’s environment |
| 2 | Customer clicks “Download” button |
| 6 | Verify logged user can see all his downloaded packages | 1 | User downloads at least one package | The system is showing all the packages which were added to user’s environment |
| 2 | Clicks on downloaded packages |
| 7 | Verify logged user can add a package to favourites | 1 | User has opened specific package | System is saving this package so the user can navigate easier to the favourite package |
| 2 | User clicks add to favourite button |
| 8 | Verify visitors can search by name | 1 | Visitor is on “Store” page | The system filters the package names by the typed filter |
| 2 | Visitor types in the search field |
| 9 | Verify customer can activate content creator’s account | 1 | Customer is on profile page | The system is adding this role to the user’s account |
| 2 | Clicks “Become content creator” |
| 10 | Verify content creator can upload a package | 1 | Content creator is on upload a package page | System uploads the package in the database so it can be visible for all the visitors |
| 2 | Fills all the information of the package |
| 11 | Verify content creator can update a package | 1 | Content creator sees all his uploaded packages | The system adds the new version to the package so the user can download it |
| 2 | Clicks update package |
| 3 | Adds the new version information and saves |

### SonarQube

Using SonarQube we can see easily how maintainable and usable is our code. We can see security vulnerabilities, warnings of probable bugs and more.  
 --- picture after including it

# Conclusion

In this Test plan we have shown how the application was tested. This way we can make sure that everything is working according to the wanted behavior, without any bugs and problems.   
We will see whether we need Tests Results document, or we can use this one.