

ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ  
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ  
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ



**ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ ΒΑΣΙΣΜΕΝΗ ΣΤΟΝ ΕΛΕΓΧΟ ΚΑΙ ΤΗΝ  
ΣΥΜΠΕΡΙΦΟΡΑ ΓΙΑ ΤΗ ΔΙΑΣΦΑΛΙΣΗ ΠΟΙΟΤΗΤΑΣ ΣΥΣΤΗΜΑΤΩΝ ΛΟΓΙΣΜΙΚΟΥ  
(BEHAVIOR DRIVEN AND TEST DRIVEN DEVELOPMENT FOR QUALITY  
ASSURANCE OF SOFTWARE SYSTEMS)**

Διπλωματική Εργασία  
του  
**Παπαδόπουλου Ελευθέριου**

Θεσσαλονίκη, Φεβρουάριος 2023

**ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ ΒΑΣΙΣΜΕΝΗ ΣΤΟΝ ΕΛΕΓΧΟ ΚΑΙ ΤΗΝ  
ΣΥΜΠΕΡΙΦΟΡΑ ΓΙΑ ΤΗ ΔΙΑΣΦΑΛΙΣΗ ΠΟΙΟΤΗΤΑΣ ΣΥΣΤΗΜΑΤΩΝ ΛΟΓΙΣΜΙΚΟΥ**

**(BEHAVIOR DRIVEN AND TEST DRIVEN DEVELOPMENT FOR QUALITY ASSURANCE OF  
SOFTWARE SYSTEMS)**

**Παπαδόπουλος Ελευθέριος**

Πτυχίο Εφαρμοσμένης Πληροφορικής, Πανεπιστήμιο Μακεδονίας (ΠΑΜΑΚ),  
2021

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΙΚΗ

Επιβλέπων Καθηγητής

**Χατζηγεωργίου Αλέξανδρος**

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 01/03/2023

1° Ονοματεπώνυμο

Χατζηγεωργίου Αλέξανδρος

2° Ονοματεπώνυμο

Σακελλαρίου Ηλίας

3° Ονοματεπώνυμο

Αμπατζόγλου Απόστολος

## Περίληψη

Κατά την διάρκεια διεκπεραίωσης της εργασίας αυτής, επιτεύχθηκε αναλυτική έρευνα στο κομμάτι της πληροφορικής που σχετίζεται με τη διασφάλιση της ποιότητας συστημάτων λογισμικού, ενώ ταυτόχρονα δημιουργήθηκε ένα ανάλογο έργο λογισμικού σε συνεργασία με την εταιρεία “Doctoranytime” με στόχο την δημιουργία ενός εργαλείου για την διασφάλιση της ποιότητας του λογισμικού της. Σκοπός και στόχος της εργασίας ήταν να αναλυθούν τα είδη και οι πρακτικές του “Ελέγχου λογισμικού”. Επιπρόσθετα, η εργασία αυτή αφοσιώθηκε εκτενέστερα στην ανάλυση των μεθοδολογιών του “Ελέγχου λογισμικού”, και πιο συγκεκριμένα αυτών ονόματι Test Driven Development (TDD), και Behavior Driven Development (BDD). Προσπαθώντας να επιτευχθεί η αξιοποίηση των προτερημάτων και των δύο μεθοδολογιών, αναλύθηκε γραπτώς και δημιουργήθηκε ένα υβριδικό έργο λογισμικού με την χρήση και των δύο μεθοδολογιών με στόχο την καλύτερη διασφάλιση της ποιότητας του λογισμικού της εταιρείας. Μερικές από τις τεχνολογίες που χρησιμοποιήθηκαν για την διεκπεραίωση του έργου αυτού είναι οι “Python”, “Playwright”, “Cucumber”, “JavaScript/TypeScript”, “Allure”. Η χρήση της καθεμίας μεθοδολογίας ξεχωριστά και μεμονωμένα, πάντα θα αφήνει κάποιο κενό στο έργο μιας εταιρείας καθώς η καθεμία μεθοδολογία εξασφαλίζει και καλύπτει διαφορετικές ανάγκες ενός έργου. Για τον λόγο αυτό η υβριδική χρήση και των δύο πρέπει να προτιμάται προκειμένου να καλύπτονται όλες οι ανάγκες ενός έργου λογισμικού που απώτερο σκοπό έχει την διασφάλιση της ποιότητας συστημάτων λογισμικού.

**Λέξεις Κλειδιά:** Έργο λογισμικού, είδη, πρακτικές, μεθοδολογίες (του “Ελέγχου λογισμικού”), ποιότητα λογισμικού, υβρική χρήση, υβριδικό project, TDD, BDD, Python, Playwright, Cucumber, JavaScript, TypeScript, Allure.

## Abstract

During the implementation of this work, analytical research was achieved which was related with the part of IT that is referred with the expression “Quality Assurance” or more descriptively, Quality Assurance of Software Systems. At the same time, a real case project was created in collaboration with the company "Doctoranytime" with the aim of creating a tool for quality assurance of its software. The scope and aim of the work was to analyze the types and practices of Testing. Additional work was dedicated to testing methodologies analysis, and specifically to Test Driven Development (TDD) and Behavior Driven Development (BDD). Trying to utilize the advantages of both methodologies, analytical research and a report were created and, more than this, a hybrid project was created using both methodologies with the aim of improving the quality of the company's software. Some of the technologies that have been used for the project are “Python”, “Playwright”, “Cucumber”, “JavaScript/TypeScript” and “Allure”. The use of each methodology shall always leave a gap in a company's project as each methodology ensures and covers different needs of the project. For this reason, the hybrid use of both methodologies should be preferred in order to cover all the needs of a project whose ultimate goal is to ensure the quality of software systems.

**Keywords:** Testing, types, practices, methodologies (of Testing), Quality Assurance, hybrid use, hybrid project, TDD, BDD, Python, Playwright, Cucumber, JavaScript, TypeScript, Allure.

## **Special Thanks**

At this point I would like to thank my professor and supervisor Mr. Alexandros Chatzigeorgiou for his guidance and willingness to amplify me as a student. Additionally, I feel thankful and thrilled because Mr. Vangelis Christou and Vasileios Fokas, the CTO and the Technical Manager of Doctoranytime, gave me an amazing opportunity which enabled me to practice my skills, both technical and soft, in order to achieve realistic and demanding tasks of a business project. Through their support I was able to understand basic concepts on how to spot the needs of a project, organize and start it, how to track any existing errors and finally how to analyze and solve them.

# Table of Contents

List of images .....	6
Symbolisms .....	8
1. Introduction .....	9
1.1 Thesis' significance .....	9
1.2 Purpose – Aims .....	10
1.3 Contribution to the QA field of informatics .....	11
1.4 Basic terminology .....	12
1.5 Structure of the research .....	16
1.6 Research methodology .....	16
2. Bibliographic Review – Theoretical Background .....	17
2.1 Software Testing .....	17
2.1.1 Testing basics .....	17
2.1.2 Ways of Testing .....	20
2.1.3 Cross-related Testing .....	21
2.1.3.1 Cross-browser Testing .....	21
2.1.3.2 Cross-platform Testing .....	21
2.2 Automated Software Testing .....	22
2.2.1 Test Automation .....	23
2.2.2 Types of Automated Testing .....	24
2.2.2.1 Functional tests .....	24
2.2.2.2 Non-Functional tests .....	24
2.2.3 Types of Automated Tests .....	25
2.2.4 TDD Practice .....	27
2.2.5 BDD Practice .....	29
3. Real life project .....	32

3.1.1	Cross device Testing .....	33
3.2	QA Project .....	34
3.2.1	Cross browser Testing .....	34
3.2.2	Types of Automated Tests.....	35
3.2.3	Initial version of Testing Project.....	36
3.1.4.1	Technology Stack .....	36
3.1.4.2	Structure.....	38
3.1.4.3	Usage .....	41
3.2.4	Updated version of Testing Project .....	42
3.1.5.1	Technology Stack .....	42
3.1.5.2	Structure.....	44
3.1.5.3	Usage .....	49
3.3	QA Tool .....	52
3.3.1	Description.....	52
3.3.2	Backend – APIs.....	52
3.3.3	Frontend – UI.....	53
3.3.4	Functionality.....	54
3.3.5	Tests execution example .....	68
3.3.6	Feature creation example .....	73
3.3.7	Conclusion .....	75
3.4	DevOps in QA .....	76
3.4.1	Pipelines .....	76
3.4.2	Results – Notifications.....	77
4.	Conclusions.....	78
4.1	Summary and conclusions .....	78
4.2	Future work .....	79
	Bibliography .....	80

## List of images

<b>Figure 2.1 TDD Cycle</b> .....	27
<b>Figure 2.2 BDD Cycle</b> .....	31
<b>Figure 3.1 Doctoranytime logo</b> .....	32
<b>Figure 3.2 Example of "POM" design pattern</b> .....	39
<b>Figure 3.3 Initial version structure</b> .....	40
<b>Figure 3.4 Updated version structure</b> .....	45
<b>Figure 3.5 Playwright configurations typescript file</b> .....	46
<b>Figure 3.6 TDD structure</b> .....	47
<b>Figure 3.7 BDD structure</b> .....	49
<b>Figure 3.8 "Instructions" page</b> .....	55
<b>Figure 3.9 Testing methodology dropdown button</b> .....	55
<b>Figure 3.10 Results information button</b> .....	56
<b>Figure 3.11 Terminate program button</b> .....	56
<b>Figure 3.12 BDD "Tests" page</b> .....	56
<b>Figure 3.13 TDD "Tests" page</b> .....	57
<b>Figure 3.14 Testing levels dropdown</b> .....	57
<b>Figure 3.15 Device dropdown</b> .....	58
<b>Figure 3.16 Browser-Emulators dropdown</b> .....	58
<b>Figure 3.17 Headless mode button</b> .....	59
<b>Figure 3.18 Disable/Enable tests button</b> .....	59
<b>Figure 3.19 "Configurations" page</b> .....	60
<b>Figure 3.20 Configurations dropdown</b> .....	60
<b>Figure 3.21 Add configuration file modal</b> .....	61
<b>Figure 3.22 Edit/Delete configuration file</b> .....	61
<b>Figure 3.23 "Results" page</b> .....	62
<b>Figure 3.24 Delete results modal</b> .....	62
<b>Figure 3.25 Tests' execution results page</b> .....	63
<b>Figure 3.26 "Allure" report</b> .....	64
<b>Figure 3.27 "Playwright" report (a)</b> .....	64
<b>Figure 3.28 "Playwright" report (b)</b> .....	65



<b>Figure 3.29 “Cucumber” report .....</b>	<b>65</b>
<b>Figure 3.30 “Feature” page.....</b>	<b>66</b>
<b>Figure 3.31 Feature’s confirmation modal.....</b>	<b>67</b>
<b>Figure 3.32 QA Tool icon.....</b>	<b>68</b>
<b>Figure 3.33 Auto-updates of QA Tool .....</b>	<b>68</b>
<b>Figure 3.34 Selecting configurations (a).....</b>	<b>69</b>
<b>Figure 3.35 Selecting configurations (b).....</b>	<b>70</b>
<b>Figure 3.36 Selecting tests .....</b>	<b>70</b>
<b>Figure 3.37 Tests execution results .....</b>	<b>71</b>
<b>Figure 3.38 Results in “Allure” report .....</b>	<b>72</b>
<b>Figure 3.39 Results in “HTML” report .....</b>	<b>72</b>
<b>Figure 3.40 Pre-submitted “Feature” page.....</b>	<b>74</b>
<b>Figure 3.41 “Feature” page final step .....</b>	<b>74</b>
<b>Figure 3.42 DevOps .....</b>	<b>76</b>
<b>Figure 3.43 Slack results notification .....</b>	<b>77</b>

## Symbolisms

- BDD (Behavior Driven Development)
- TDD (Test Driven Development)
- AAT (Automated Acceptance Tests)
- AATDD (Automated Acceptance Test Driven Development)
- ISTQB (International Software Testing Qualifications)
- QA (Quality Assurance)
- UI (User Interface)
- FE (Front End)
- BE (Back End)
- FS (Full Stack)
- E2E (End-to-end)
- API (Application Programming Interface)

# 1. Introduction

## 1.1 Thesis' significance

It is very crucial for all companies and for all projects to be usable and to have no conflicts or errors. Companies devote substantial effort to considerable lengths to ensure that their projects are error-free. To achieve this, most companies have already started investing in their quality assurance (QA) or testing departments to test their products before they reach the end users. QA departments consist of manual and/or automated testers. Manual testers are responsible for testing the company's product by manually performing all possible user actions. Automated testers, on the other hand, create scripts to automate the possible user actions. Over the years, a lot of research has been done in the field of testing and many types and methodologies have been developed. There are many different types of testing and several methodologies to accomplish all these testing types.

All the above points will be analyzed in this master thesis, with a special focus on the methodologies, namely the so-called Test Driven Development (TDD) and Behavior Driven Development (BDD). The latter is a new methodology capable of closing the gap between developers and product contributors, a problem that most companies are trying to solve.

## **1.2 Purpose – Aims**

The main purpose of this master thesis is to analyze the notion and the necessity of Testing in a life cycle of software implementation in order to ensure the Quality Assurance of this software. Afterwards, a wide analysis related to the methodologies of Automated Software Testing, and more specifically the ones named Test Driven Development (TDD) and Behavior Driven Development (BDD), shall take place. Moreover, this master thesis explains the need of practicing both testing methodologies within a project, in order to increase the Quality Assurance of the product. Finally, this master thesis explains, how developers and non-developers are able to collaborate by using the BDD methodology. Except the bibliography part of this master thesis, a relative project, in collaboration with the company named “Doctoranytime”, has been generated. In this project, both of the above testing methodologies are used, which helped the company to handle easily its testing project, and increase the Quality Assurance of its product.

## **1.3 Contribution to the QA field of informatics**

1. A review of the relevant literature has been made, considering both official guides (e.g. ISTQB information) and scientific literature (i.e. articles).
2. Information related to Testing in general and Testing methodologies was gathered from the relevant literature.
3. This information was filtered and combined in order to create the bibliography part of the present master thesis.
4. A real-life project was created, by using new technologies (Playwright, Cucumber).
5. Two testing methodologies (TDD and BDD) were used and combined in order for a real-life project to be generated.
6. In order to ease the handling of the Test Project, a platform (QA Tool), with BE and FE parts was created. There testers (manual and automated) are able to set the desired configurations, select and filter the available tests that will be executed and check afterwards, the respective results.

## 1.4 Basic terminology

### ➤ **Software**

*“Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.” [1] [2]*

### ➤ **Functionality**

*“The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.” [1] [2]*

### ➤ **Software Testing**

*“Software testing is a way to assess the quality of the software and to reduce the risk of software failure in operation.” [3]*

### ➤ **Automated Software Testing**

*“Automated testing is the application of software tools to automate a human-driven manual process of reviewing and validating a software product.” [4]*

### ➤ **Test Automation**

*“Test automation is the practice of running tests automatically, managing test data, and utilizing results to improve software quality. Test automation is primarily a quality assurance measure, but its activities involve the commitment of the entire software production team.” [5]*

### ➤ **TDD Practice**

*“Test Driven Development is a Testing Practice. Its main idea is to perform initial unit tests for the code that must be implemented; that is, first codify the test and, subsequently, develop the business logic. In TDD, tests are written by the developers based on the requirements specified by the clients.” [2]*

➤ **ATDD Practice**

*“Acceptance Test-Driven Development (ATDD), also known as Story Test-Driven Development (STDD), is a technique similar to TDD, but at a different level. This technique involves team members with different perspectives (clients, development, tests) who collaborate to write acceptance tests before the implementation of a functionality.” [2]*

➤ **BDD Practice**

*“Behavior-Driven Development (BDD), is a synthesis and refinement of software engineering practices that helps teams generate and deliver higher quality software quickly. Its base represents some agile practices and techniques, including, in particular: Test-Driven Development (TDD) and Acceptance Test-Driven Development (ATDD).” [2]*

➤ **POM**

*“Page Object Model (POM) is a design pattern, popularly used in test automation that creates Object Repository for web UI elements. The advantage of the model is that it reduces code duplication and improves test maintenance.” [6]*

➤ **APIs**

*“APIs are mechanisms that enable two software components to communicate with each other using a set of definitions and protocols. For example, the weather bureau’s software system contains daily weather data. The weather app on your phone “talks” to this system via APIs and shows you daily weather updates on your phone.” [7]*

➤ **Modal**

*“A modal (also called a modal window or light box) is a web page element that displays in front of and deactivates all other page content. To return to the main content, the user must engage with the modal by completing an action or by closing it. Modals are often used to direct users’ attention to an important action or piece of information on a website or application.” [8]*

➤ **Software feature (Feature)**

*In software, the term feature has several definitions, which are often distinct from the more general definitions of the term. The Institute of Electrical and Electronics Engineers (IEEE) defines the term in IEEE 829 (a now-defunct standard for software test documentation) as a "distinguishing characteristic of a software item (e.g., performance, portability, or functionality)." [9]*

➤ **Production environment**

*"The last environment in software development, this is where new builds/updates are moved into production for end users." [10]*

➤ **Test environment**

*"As the name implies, this is where application testing is conducted to find and fix errors." [10]*

➤ **Staging environment**

*"Here, all the work done in the development environment is merged into the built system (often used to automate the process of software compilation) before it is moved into the production environment." [10]*

➤ **Software release (Release)**

*"A release is the distribution of the final version or the newest version of a software application. A software release may be public or private and generally signifies the unveiling of a new or upgraded version of the application." [11]*

➤ **DevOps**

*"DevOps is a combination of software development (dev) and operations (ops). It is defined as a software engineering methodology which aims to integrate the work of development teams and operations teams by facilitating a culture of collaboration and shared responsibility." [12]*

➤ **DevOps pipeline (Pipelines)**

*"A DevOps pipeline is a set of automated processes and tools that allows developers and operations professionals to collaborate on building and deploying code to a production environment." [13]*



➤ **Continuous Integration**

*“Continuous integration is a software development process where developers integrate the new code they've written more frequently throughout the development cycle, adding it to the code base at least once a day. Automated testing is done against each iteration of the build to identify integration issues earlier, when they are easier to fix, which also helps avoid problems at the final merge for the release. Overall, continuous integration helps streamline the build process, resulting in higher-quality software and more predictable delivery schedules.” [14]*

➤ **Continuous Delivery**

*“Continuous delivery lets development teams automate the process that moves software through the software development lifecycle, and it can provide many benefits when provisioning an integrated toolbox.” [15]*

## **1.5 Structure of the research**

In chapter two (2) the research methodology will be analyzed. In the next chapter (3) the theoretical background of the master thesis will be explained. More specifically, many details about Software Testing will be given, such as the Testing basics, the Ways of Testing and plenty other information. More than this, details and information about Automated Software Testing shall be provided too (Test Automation, Types of Automated Testing/Tests, Testing Practices). In chapter four (4) the practical part of master thesis will be explained. In particular, firstly, the initial version of the project shall be explained and, then, the new version of the project will be presented. Finally, in the last chapter (5), a summary and several conclusions exist. More than this, some ideas and possible functionalities are provided in order to expand this project in the future.

## **1.6 Research methodology**

This master thesis investigates and analyzes the meanings and practices of Software Testing. It is separated into two parts. In the first part, the more theoretical one, data and information from websites, articles and several other master theses were utilized. The data collected from several sources, were combined to create a completed and structured bibliography part related to Software Testing. A reader of this master thesis is able to fully understand the meanings and the importance of Software Testing. The data that have been used, have already been collected or created by other researchers (secondary research). The second part of the present thesis, the more practical one, includes data and details about the development of a product, with several features, procedures and techniques (applied research). For the creation of this product, several technologies, programming languages and frameworks were used. More than this, several testing methodologies and techniques were utilized. The practical part of this master thesis is based on the theoretical one. A reader of this master thesis and a user of the product is able to fully understand how the theoretical part can be applied to a product of a real-life project.

## **2. Bibliographic Review – Theoretical Background**

### **2.1 Software Testing**

#### **2.1.1 Testing basics**

Nowadays more and more software systems are created. Many daily human activities are related to software systems. From a company's point of view, it is very important for its applications to have as few failures, bugs, and errors as possible. This is very crucial because if the company's products, which are used by many people, do not work properly, this can lead to many negative consequences, including loss of money, time or business reputation.

For this reason, many companies have begun to invest in software testing in order for their products to work as expected and, more than this, to be efficient and productive. It's not only that companies have begun to create Quality Assurance (QA) departments in order to test their products, but, also, have begun to train their developers to test their own source code either manually or with automated way.

Software testing, as an official definition from ISTQB industry, is a way to assess the quality of the software and reduce the risk of software failure in operation.

Software testing is an investigation conducted to provide stakeholders with information about the quality of the software product or service under test. More simply, software testing is a method to check whether the actual software product matches the expected requirements and ensure that software product is defect free. It can also provide an objective and independent aspect regarding the software to allow the business appreciate and understand the risks of software implementation. [\[3\]](#) [\[16\]](#) [\[17\]](#)

Testing methods involve the execution of software/system components using manual or automated tools to evaluate one or more functionalities. The main purpose of testing is to identify errors, gaps or missing requirements in contrast to actual requirements. In particular, test techniques include the process of executing a program or application with the intent of finding software bugs (errors or other defects), and verifying that the software product is ready to be used.

Software testing is very important because if any bug or errors exist in the software, these can be identified early enough and can be solved before the delivery of the software product to the end-user. Properly tested software product ensures reliability, security and high performance which further results in time saving, cost effectiveness and customer satisfaction.

Software testing can be conducted as soon as executable software, even if partially completed, exists. The overall approach to software development often determines when and how testing is conducted. In a phased process, the biggest part of testing is taking place after system requirements have been defined and implemented in testable programs. On the contrary, under an agile approach, requirements, programming and testing are often executed simultaneously. [\[18\]](#) [\[19\]](#) [\[17\]](#)

Testing is not only the procedure of executing tests. Test planning, analyzing, designing and implementing are very crucial parts of testing. More than this, testing includes the operation of reporting progress, showing results and evaluating the quality of a test object. Moreover, testing includes reviewing “work products”. By “work products” we mean requirements, user stories and source code. Last but not least, testing includes validation. In this procedure, it is checked whether the system is able to cover the user’s needs in its operational environment(s). [\[3\]](#) [\[16\]](#) [\[17\]](#)

Furthermore, testing meets many benefits. Some of these benefits are mentioned below:

- ✓ Cost-Effectiveness
- ✓ Security
- ✓ Product Quality
- ✓ Customer Satisfaction

All of them will be analyzed further.

To begin with, the first and most important benefit of software testing is that it is cost-effective. In particular, testing every IT project on time helps companies to save money for the long term. In case that the bugs of a project are caught in the earlier stage of software testing, it will cost much less to the company to fix it. [\[20\]](#) [\[21\]](#) [\[22\]](#)

In addition, the most vulnerable and sensitive benefit of software testing is security. Clients are always looking for products that can trust. Software testing is very important as it is a practice that removes the majority of risks and problems at a very early stage. [\[20\]](#) [\[21\]](#) [\[22\]](#)

Moreover, software testing increases products' quality. This is an essential requirement of any software product. Testing ensures that a product of high quality shall be delivered to the clients. [\[20\]](#) [\[21\]](#) [\[22\]](#)

Finally, the main aim of any product that will be delivered to clients is to satisfy them. Software testing preserves the best user experience for the client from a business perspective. [\[20\]](#) [\[21\]](#) [\[22\]](#)

### 2.1.2 Ways of Testing

There are two ways of testing. The first one is the manual testing and the second one is the automated testing. In the past, manual testing was more familiar to the companies, as it was the one and only way of testing that companies were conducting. Nowadays the most usual type of testing is the automated one. [\[23\]](#) [\[17\]](#)

In recent years manual testing is not so often used as the automated testing, but this does not mean that it has been erased from the testing world. Sometimes, manual testing is the only way to test a feature of a software in order to be sure about its validation. [\[23\]](#) [\[17\]](#)

Many companies are trying to create automated test scenarios and test cases in order to test their products and make all testing processes automated. As it was mentioned before, usually manual testing is the one and only way to test a feature of a software. For this reason, the results of the “experiments” that have been made by the companies in the field of testing, have indicated that a hybrid model of testing (both manual and automated) is the best way to test software and increase the quality assurance of the product. [\[23\]](#)

## **2.1.3 Cross-related Testing**

### **2.1.3.1 Cross-browser Testing**

Cross-browser testing is the type of testing that verifies that the functionalities of an application and all of its features work correctly and as expected in different browsers. This type of testing represents a process of verifying an application's compatibility with different browsers. In particular, cross-browser testing provides a programmer with the possibility to check whether a website works properly when accessed through different browsers combinations (versions are included too), different devices, and last but not least, different assistive tools. [\[22\]](#) [\[24\]](#) [\[25\]](#)

To define and explain the above information more simply, cross-browser testing is a process rendering the possibility to test a website or an application in multiple browsers in order to make sure that it works as expected and steadily. [\[22\]](#) [\[25\]](#)

### **2.1.3.2 Cross-platform Testing**

Cross-platform testing is a testing process that is used in order to check the functionality and adjustability of an application across different platforms. These different platforms are operating systems and devices. Specifically, cross-platform testing provides a tester with the possibility to check whether a program works as expected for all operating systems (Windows, Linus, iOS or Android). [\[22\]](#) [\[26\]](#)

This type of testing (cross-platform testing) is indispensable in terms of quality assurance. Once a product starts to be developed for multiple platforms, then amount of platforms that can be chosen is endless. Cross-platform testing is needed to be performed in order to determine what behavior an application or website adopts in many different environments. By using this type of testing, it is feasible to identify any existing errors that may vary among platforms and configurations. These errors may be related to stability, user interface, usability, and performance. [\[22\]](#) [\[26\]](#)

## **2.2 Automated Software Testing**

Automated software testing cannot be missing from a company that aims to increase the quality assurance of its products. This type of testing does not include only writing and executing automated scripts. It includes several operations before and after the aforementioned actions. Before the implementation and the execution of the source code, there are some other steps such as test planning, analyzing and designing. After the implementation and execution of the test scenarios and test cases, the steps of reporting and evaluating the results follow. Finally, the last step is the one of reviewing the operations.

Focusing on the steps of the implementation of the test scenarios and test cases, there are mainly two ways to create automated tests and scripts. The first one, named Test Driven Development (TDD), is the most widely known and used. By this way of testing, only the developers participate into the source code implementation. The second one, named Behavior Driven Development (BDD) is, in a way, a new testing methodology, that is capable of creating very productive and safe communication between developers and upscale employees. Specifically, this technology is able to embed employees from the above grades of a company to the operation of creating the test scenarios and test cases.

More details about these two testing methodologies will be mentioned below.



### 2.2.1 Test Automation

Test automation is the practice of running tests automatically, managing test data, and utilizing results to improve software quality. This practice saves time, effort and money to the companies. In order a test to achieve these goals, it must comply with some criteria. The most crucial of them will be mentioned below. [\[3\]](#) [\[16\]](#) [\[17\]](#)

For starters, a test should be repeatable. Tests should be created in such a way that can be used multiple times.

In addition, a test should be determinant. This means that the test must return every time the same result without deviating from the expected goal. More simply, the execution, having always the same data as an input in order to execute the test, should always conclude in a similar, if not the same, outcome. The term “similar outcome” means that either the test will literally return every time the same result, or, the test will confirm that a process works properly. [\[3\]](#) [\[16\]](#)

Test automation is a practice that companies should include in their processes, in order to increase their productivity and succeed to survive in the market.

## **2.2.2 Types of Automated Testing**

There are two main types of automated testing, the functional and the non-functional one.

### **2.2.2.1 Functional tests**

Functional tests are capable of testing the business logic behind an application. This type of test automation includes scripts that are able to validate the business logic and the anticipated functionality of an application or a website. More specifically, functional testing includes tests that evaluate functions that the system should perform. [\[27\]](#) [\[28\]](#)

### **2.2.2.2 Non-Functional tests**

Non-functional tests are capable of testing the non-business specifications of an application. These specifications are the ones that are related to the usability, performance and security of an application or a website. These specifications can be stable or they can be scaled by the size of the software. [\[28\]](#)

### 2.2.3 Types of Automated Tests

There are several types of automated tests. These several types of automation tests define what kind of test suites can be automated. Some of them will be further analyzed. The types of automated tests that will be analyzed are the following:

- Unit tests
- Code analysis tests
- Smoke tests
- Integration tests
- End-to-end (E2E) tests
- Acceptance tests
- Performance tests

To begin with, **Unit tests** are small and very quick tests that are responsible to check the functionalities of the code. These tests are on a build server and they do not communicate with database, external APIs, or file storage. They are made to test only the code itself and not the external dependencies. As their name indicates, this kind of tests is responsible for validating the proper functionality of a simple unit. This type of testing is being implemented mostly from the developers of a team and not from the testers. Finally, this type of testing can be succeeded for both the front-end (FE) and the back-end (BE) part of a product. [\[29\]](#)  
[\[30\]](#)

**Code analysis tests** are running when a developer checks the code. Other than configuring rules and keeping the tools up to date, there is not much source code writing to do with these automated tests. There are several types of code analysis tools. Some of them spot security defects, and others test the code's style and form. [\[31\]](#) [\[22\]](#)

**Smoke tests** are very quick tests with small complexity in their functionalities. These tests are responsible to check crucial parts and features of a product. They are mainly needed to ensure that all services and dependencies are “up” and running. A smoke test is not

necessary to be an all-out functional test. It can also be run as part of an automated deployment or triggered through a manual step process. [\[3\]](#) [\[22\]](#)

**Integration tests** are responsible to check the more complex parts of a product. They are responsible to test the whole integration of a program or a system, even with external operations or other micro services. For this reason, an integration test is more complicated and difficult to set up. Integration is mostly ‘valuable’ for the business core of a company or organization. [\[29\]](#) [\[22\]](#) [\[30\]](#)

**End-to-end (E2E) tests** are responsible to check several combined functionalities of a product, in a queue from start to endpoint. These tests verify that all components of a program or a system are able to run and be combined as expected. [\[32\]](#) [\[33\]](#)

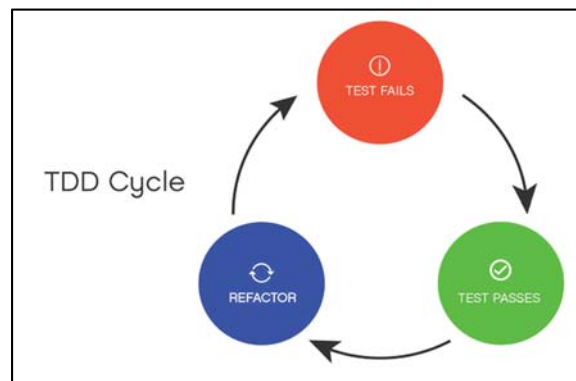
**Acceptance tests** are executed at the end of the testing process. They are responsible to determine whether a program or a system has the expected functionality and more importantly, the functionality that has been agreed upon to have. Therefore, it is crucial for developers, businesses and testers to expand their cooperation in order to create the proper acceptance test cases under the client’s requirements. [\[30\]](#)

**Performance tests** are very important for a product. These tests are responsible to check the performance of an application in several circumstances. There are many kinds of performance tests. Each one is responsible to check a different operational part of a program or a system. For instance, according to the specialists on the performance field, each web platform should be visible in less than six seconds. Having a platform opening in more than six seconds, consists a red limit that should be “caught” by the QA sector as a performance issue. [\[29\]](#) [\[22\]](#) [\[3\]](#)

## 2.2.4 TDD Practice

Test Driven Development (TDD) is a software development process. This practice is a software development process that repeats short development cycles. The idea of this practice is that firstly developers should create the test cases and then write the respective source code. Then the code that is written for a functionality of the software, is repeatedly tested by these test cases, until it passes them. In the next paragraph a complete analysis about the steps of the TDD practice is presented. [34]

TDD software development process has mainly three steps.



**Figure 2.1 TDD Cycle**

[<https://blog.ippon.tech/a-roadmap-to-tdd-adoption>]

To begin with, the first step is to write down the aim of a new feature and create a test function for this feature. For example, if the purpose of a test is to check the content of the site's robots.txt file, then the respective test function can be named "check-robots-txt-file-content.spec.ts" (TypeScript syntax). This function will include all the necessary steps in order to get and check this file's content. A browser would need to be opened on the desired page. Then, the content of the page would be validated so it can meet the expected outcome.

Then, in the second step, a developer needs to run the test and check for errors and failures in the results. For example, all the above functions are going to be executed. Then the results of this process should be evaluated. Is the robots.txt file's content the expected

one? If yes, then the test is ready. Otherwise, there is a problem and an investigation should take place.

In the third part, the developer is fixing the code for the new feature and rerunning the test function. Then, the developer is checking again for errors and failures. This step should be repeated until no error and failure exist as a result from the test function.

By following this practice, developers are confident regarding their source code and sure that their code operates properly. More than this, by implementing this operation, the quality of the code is increased and the technical debt is being eliminated. Finally, this practice is capable of achieving high test coverage. The entire life-cycle of a TDD is often referred to as the “red-green-refractor” cycle.

Although TDD is a development practice, testers use this technique for automation testing development. Focusing on the usage of the TDD method in the QA field, initially, testers appoint to the goal of a new feature by their supervisor or manager. The new feature is, most times, already created by the developers. Then, the testers create a test scenario with several test cases. They run this test scenario with all its test cases and check the results for errors and failures. After this process, testers fix these errors and failures in their source code and they execute again the test scenarios and test cases. This process should be repeated until no error or failure exists as a result from the whole tests.

The most important characteristic of this practice, that will increase the effectiveness of the testing part and consequently the quality of the source code, is that the test scenario and test cases should be created before the creation of the new feature by the developers. In this way, the test scenarios and test cases will not be affected by the creation of the new feature, but they will depend on the idea of the new feature. [\[34\]](#) [\[2\]](#)

## 2.2.5 BDD Practice

Like Test Driven Development (TDD), Behavior Driven Development (BDD) is a software development process. This testing technology, as the name suggests, illustrates the methods of developing a feature based on its behavior. The behavior is described in a file in a very simple human language, easy to be read and understood by a person, developer and non-developer.

This practice helps teams to generate and deliver higher-quality software quickly. Behavior Driven Development or BDD is an extension of Test-Driven Development. In this practice, instead of writing test scenarios and test cases at the beginning, several behaviors are written. When these behaviors are written, then the required code for a program or a system to perform these behaviors, is going to be written. This testing technology is capable of making the collaboration of developers, testers, and business users feasible and easier. [\[2\]](#) [\[35\]](#)

This practice has more steps than TDD practice. More specifically, this practice consists of six steps. Some of them are similar to TDD. These steps will be mentioned below [\[35\]](#):

### ➤ Write the behavior of a program or system

*This behavior is going to be written in a file in simplified English language by the product owner or business analysts or QAs. Here is an example of such a file (feature file):*

```
Feature: Check homepage search button  
  
# Check homepage search button  
  
@Scenario  
Scenario: Check homepage search button  
Given I am to "/" page  
When I click the search button  
Then I should redirect to the main s page
```

➤ **Write the automated scripts**

*This text that was written in the English language is going to be converted into programming tests. These tests are going to be written in a programming language (Python, Java, JavaScript, etc.) and with the usage of a testing framework (Selenium, Playwright, Cypress, etc.). The structure of these automated scripts differs among programming languages and of course among different testing frameworks. Each step of the feature files is going to be converted into source code. In this way, the human language is going to be translated into language readable from computers. Here is an example of this conversion (written in programming language “TypeScript” and with the usage of testing framework named “Playwright”):*

```
// @When 'I click the search button'

When ('I click the search button'), async () => {

    const homepage = new Homepage();

    await homepage.clickSearchButton();

};
```

➤ **Execute the source code**

*The functional code underlying the behavior is then executed. For every step of a feature file, the respective implementation is going to be executed.*

➤ **Check if the results are passed**

*This source code that was written is going to be executed and checked for its results. If the results are the expected ones, then the next behavior is going to be prepared. Otherwise, this behavior should be fixed. These results can be imprinted and evaluated through a reporting tool.*

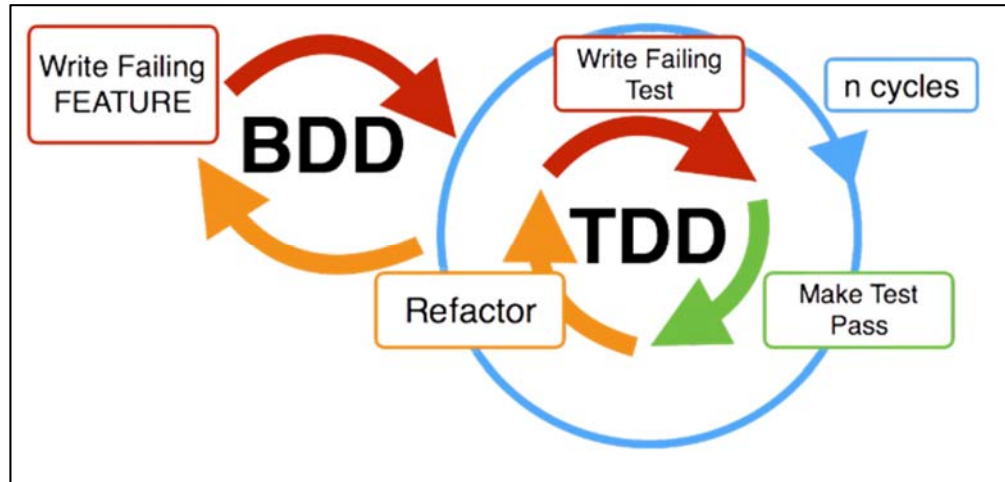
➤ **Refactor the source code**

*This source code that was written is going to be refactored and organized in order to be readable and maintainable. More important, it is going to be used from many other feature files that include some of these steps.*



➤ **Repeat steps 1-5 for new behavior**

*Finally, steps 1-5 are going to be repeated in the implementation of the next behaviors.*



**Figure 2.2 BDD Cycle**

[\[https://saucelabs.com/blog/a-two-minute-bdd-overview\]](https://saucelabs.com/blog/a-two-minute-bdd-overview)

### 3. Real life project

The company named “Doctoranytime” gave me the opportunity to structure my master thesis based on its product. More specifically, the present master thesis is related with the Testing Project of the company.



*Figure 3.1 Doctoranytime logo*

This testing project was used in the master thesis, in order for the project to be updated with new technologies and new methodologies. This project is separated in three parts:

- ✓ QA
- ✓ QA Tool
- ✓ DevOps

### 3.1.1 Cross device Testing

The initial QA project was also a cross-device testing project. The test scenarios and test cases were executing not only on multiple browsers, but also on multiple devices. This attribute ensures that a user can use Doctoranytime's product on multiple devices and more importantly on the device that this user desires the most.

As someone can easily understand, cross-device testing is as crucial as is the cross-browser testing for a product. It is also necessary to be created in order to increase the quality assurance of this product.

Without altering this characteristic, the updated version of the QA project is a cross-device testing project too. In fact, in the updated version of the QA project, test scenarios and test cases are able to be executed in more emulators (devices) than the ones in the initial version.

The **initial version** of the QA project was able to execute test scenarios and test cases in the following devices:

- Moto G4

More devices for cross-device testing can also be supported in this version.

The **updated version** of the QA project is able to run the test scenarios and the test cases in the following devices:

- Moto G4
- Galaxy S8

The updated version of the QA project is able to run test scenarios and test cases in more devices as it is created right now and shall be able to support even more devices.

## 3.2 QA Project

In this chapter, the QA part of the testing project shall be analyzed. The characteristics of the first part of the project, the QA part, shall be analyzed concerning the initial and the updated version of the project.

### 3.2.1 Cross browser Testing

The initial QA Project was a cross-browser testing project. In that case, the test scenarios and test cases were running in multiple browsers, in order to ensure that a user can use Doctoranytime's product on multiple browsers and more importantly on the browser that this user desires the most. As someone can easily understand, cross-browser testing is very crucial for a product in order to increase the quality assurance of it.

Maintaining the aforementioned attribute, the updated version of the QA project is a cross-browser testing project too.

The **initial version** of the QA project is able to run test scenarios and test cases in the following browsers:

- Chrome
- Mozilla Firefox
- Microsoft Edge
- Opera

The **updated version** of the QA project is able to run test scenarios and test cases in the following browsers:

- Chrome
- Mozilla Firefox
- Microsoft Edge

The updated version of the QA project is able to run test scenarios and the test cases in fewer browsers as it is formed right now. It is worth to be mentioned that in the future shall be able to support more browsers, even more than the initial QA project is able to support.

### 3.2.2 Types of Automated Tests

Both versions of Doctoranytime's QA project use several types of automated tests in order to increase the quality assurance of the company's product.

In particular, both versions of the QA project, include **smoke tests** (these are the most crucial and the fast tests) that produce quick results about the most crucial parts of Doctoranytime's product such as the "robots.txt" file content, the content of the sitemap, the SEO (Search Engine Optimization) characteristics of the site, etc. Moreover, within this type of automated tests of the QA project, there is a subcategory of automated tests, the API tests. A lot of tests have been created that check the functionality of the product's APIs.

More than this type of automated tests, both versions of Doctoranytime's QA project include **integration tests**. The duration of this kind of tests is longer than the one of the smoke tests. Also, integration tests check less crucial features of the company's product. More specifically, these tests are created in order to check the behavior of different units, modules and components of Doctoranytime's product, as a combined entity. Tests like the ones checking the login or the register functionality of the website, the search functionality of the website, etc., belong in this category.

Finally, both versions of Doctoranytime's QA project, include **end-to-end tests**. This type of tests lasts longer than integration tests and much longer than smoke tests. They are not as crucial as integration tests and of course, they are a lot less crucial than smoke tests.

This type of tests validates the entire product of the company from start to an end point, along with its integration with external interfaces. With this type of tests, the whole product

of Doctoranytime is tested, and more specifically, the dependencies, data integrity, and communication with other systems, interfaces and databases.

A test that checks a specific functionality of the product does not belong in this category. On the contrary a test that executes several functionalities and checks a main or possible behavior of a user of Doctoranytime's product, does belong to this category. An end-to-end test for Doctoranytime's product can be a test that executes a user's login, then the selection of a doctor for an appointment, then booking an appointment with this doctor, and lastly the user's logout. This is a test that represents the main behavior of a user of this product.

### **3.2.3 Initial version of Testing Project**

In this subchapter, the initial version of QA part of the project shall be analyzed.

#### **3.2.3.1 Technology Stack**

To begin with, the QA part of the project was created by the use of multiple technologies, frameworks and methodologies. To name but a few:

- Python
- Selenium
- Pytest
- JSON
- Allure

In more details, the programming language that was used to create the QA part of this project is the one named **“Python”**. This language is an Object Oriented programming language and it was chosen because of its numerous advantages and convenient usage.

In order to automate processes in the browser to succeed automation testing, the framework “**Selenium**” was used. This framework was preferred because of its numerous advantages, the big amount of information in several blogs and websites, and its very clarified documentation.

Afterwards, the framework “**Pytest**” was used as a testing management tool. Pytest is a very powerful framework with many advantages. Some of them shall be further mentioned. This framework provides testers with the possibility to manage tests’ executions, configure tests in general mode, and, in more individual mode too, by configuring specific tests. More than this, Pytest enables users to parameterize test scenarios and increase test cases.

To store several data, that were needed in the automated testing process, JSON files were used. In order to easily handle these files, JSON technology was used. In its libraries Python includes a JSON library that enables programmers to handle JSON files.

Finally, in order to store the results of the tests’ executions and make them readable, not only for testers but any employee of the company that was related to the testing project, the “**Allure**” technology was used. This framework has many possibilities and produces a detailed report. In an “Allure” report, anyone can find tests in an organized depiction. More specifically, tests are organized in suites. Every test scenario includes several test cases. Each test case has its own information (name, parameters, duration, retries, history) and its own results (status, messages, screenshots, videos). More than this, an “Allure” report includes general information, such as the total duration of the tests’ execution, several charts related to the duration and the success percentage of this tests’ execution, and even more.

### 3.2.3.2 Structure

Having fully analyzed the technology stack of the QA part of the testing project, further the structure of the QA part of the testing project shall be explained.

To begin with, the project was separated in several folders. The most important were the following:

- Selenium related
- Configurations
- Components
- Tests
- Results

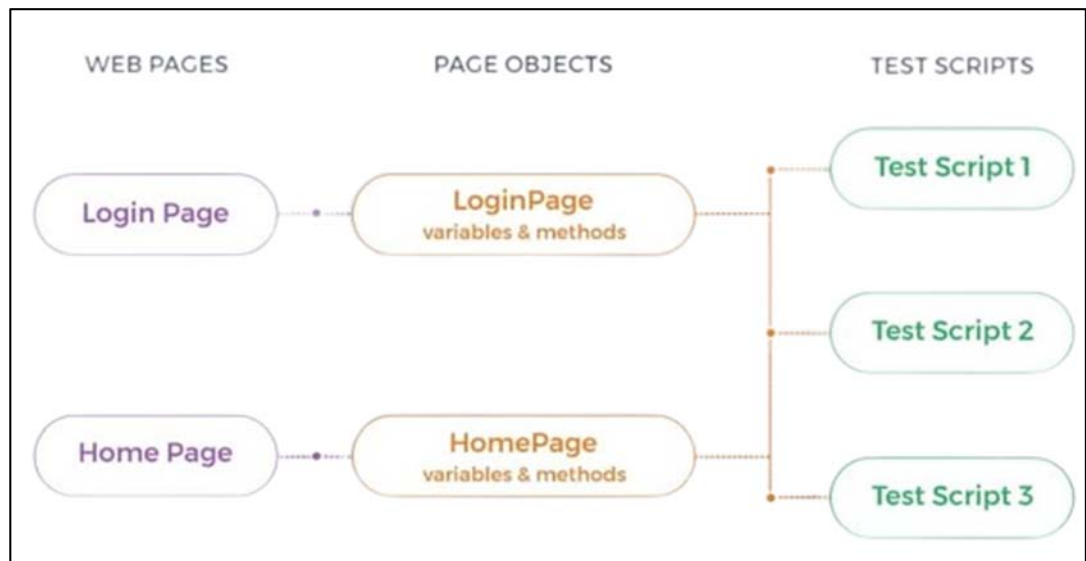
The first folder, named **"Selenium related"**, included files that were related to the framework "Selenium". In particular, in order for the testers to easily use the functionalities of the framework mentioned above, a small custom framework was created, which included methods with grouped Selenium functionalities. Such a method was the "click\_element" one. This method was responsible for finding the desired element, scrolling into the element's view, in order for the element to be visible, and, then, clicking it. Several other methods like the aforementioned were created, something that was very helpful for testers.

The folder named **"Configurations"** included several configurations that were necessary for the tests' execution. These configurations were stored in JSON files and contained information for every website/country where the company operates, every environment (staging, test and production) and several other information, such as the base URL, credentials that would be needed for the tests, etc.

For the QA project of the company, the POM concept was implemented. "POM", resulting from Page Object Model, is a concept in which every page of a website is represented with a class. In this class, the respective locators and methods are



contained. So, in the folder “**Components**”, a subfolder with pages-classes was included. These all pages-classes were included in a subfolder, named “Classes”.



**Figure 3.2 Example of "POM" design pattern**

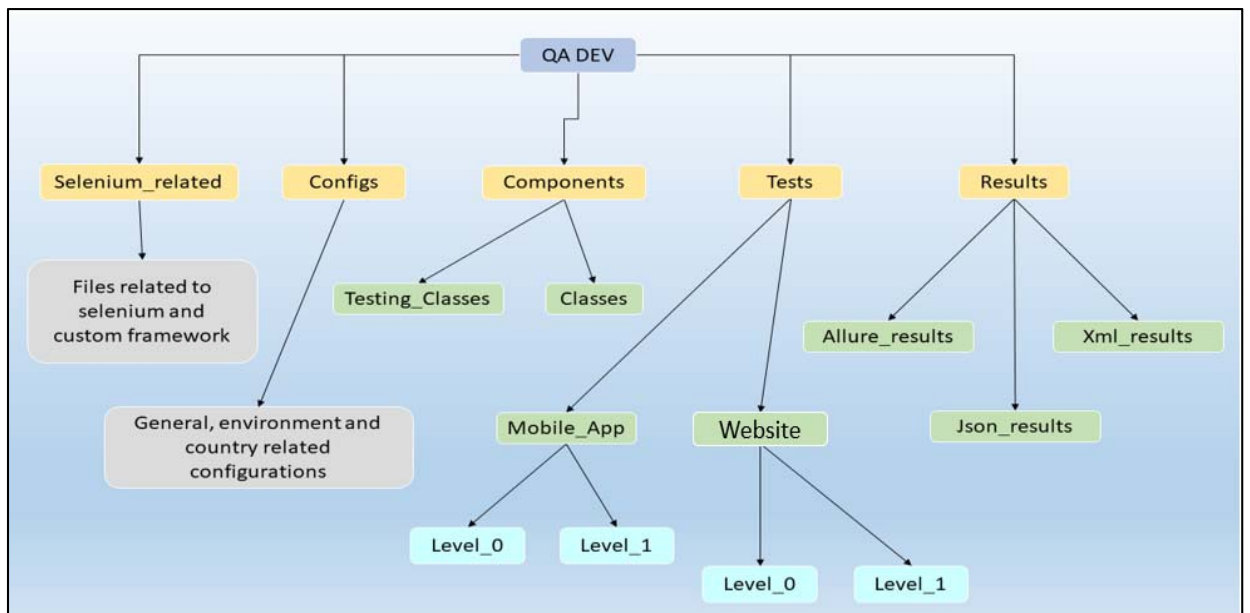
[\[https://blog.testproject.io/2021/01/06/test-automation-framework-benefits-pom-selenium-locators-opensdk-junit/\]](https://blog.testproject.io/2021/01/06/test-automation-framework-benefits-pom-selenium-locators-opensdk-junit/)

Besides this subfolder, an extra one was created, named “Testing Classes”. This folder contained classes which were implemented with the concept of POM, combining and making use of the previous mentioned classes and their methods, in order to create functions with grouped actions. These functions were called and executed in the test scenarios and test cases.

The folder named “**Tests**” included tests separated into categories. These categories were related to the kind of device that the code was testing, the criticality of the tests, and the type of these tests. For example, a test that was related to Search Ending Optimization (SEO) and was created to achieve Desktop testing was included in the first Level of the Desktop tests, in order to have higher priority and be executed sooner than the other tests. This test belonged in the smoke tests category. The tests of this folder were making use of the classes and methods that existed in the “Testing Classes” folder.

Finally, in the folder named **“Results”**, the results of the tests’ execution were stored. It is worth to be mentioned that this folder included results of three different formats. More specifically, this folder contained results with JSON and XML format and results with the format of the reporting framework “Allure”. The last mentioned technology is a framework produces several files (JSON, txt, images) for each tests’ execution, which are generated during the tests’ execution and are necessary in order for the “Allure” reports to be generated.

The structure that has been analyzed before is showed in the following image.



**Figure 3.3 Initial version structure**

As it was mentioned before, in order to execute the available tests, the technology “Pytest” was used. This technology is able to search for available tests on the path that the user has defined. Pytest provides the user with the possibility to execute the tests of his/her desire, with specific configurations, and detailed results. In order to easily handle this technology, a wrapper file was created (with the usage of the programming language “Python”), which included source code for the execution of the necessary processes before the tests’ execution. An indicative pytest command appears below:

```
pytest "./Tests/Website/" -k "check-robots-txt-content"
```

### **3.2.3.3 Usage**

The initial version of the QA part of the testing project was using only the testing methodology, Test Driven Development (TDD). Testers did not often communicate with the upper departments. They were responsible for the whole testing operation. In particular, they were responsible for preparing the test scenarios and tests cases and then creating them. The preparation of the test scenarios and test cases included the Test Design and the documentation of these tests. Moreover, testers were responsible for evaluating the functionality of these test scenarios and test cases after their creation. In the case that these tests were not ready or were not executed correctly, testers had to fix them. Finally, the testers of the company were responsible for evaluating the results of the tests' executions and informing the relevant employees for the existence of any bugs and errors of the company's product. This methodology and process were in progress for every test scenario, test case and every new feature that the website was about to include.

In this chapter, a brief analysis of the initial version of the project has been presented. From the information being provided above, someone can easily perceive that the responsibilities of testers have been plenty and the whole operation was disorganized.

An extensive analysis for the updated version of the testing project shall follow. The updated version contains a lot and different processes.

### 3.2.4 Updated version of Testing Project

In this subchapter, the updated version of the QA part of the project shall be analyzed.

#### 3.2.4.1 Technology Stack

To begin with, the updated version of the QA part of the project was created by the use of multiple technologies, frameworks, and methodologies, as the previous one. In this updated version, though, the technologies are not exactly the same with the ones that were mentioned in the previous chapter. Some extra technologies were added to increase the productivity of the testing project. In the list below, the technologies of the updated project appear:

- Python
- TypeScript
- JavaScript
- Playwright
- CucumberJS
- JSON

The programming language “**Python**” is used in order to create a wrapper file that shall control the whole testing operation. In particular, as in the initial version of the project, this wrapper file is responsible to set, orchestrate and execute the testing project.

The programming languages “**JavaScript**” and “**TypeScript**”, are Object Oriented programming languages and are used for the rest of the testing project. These programming languages were selected because of their numerous advantages and mostly for their compatibility with the testing framework that was selected for the update of this testing project (Playwright). TypeScript was preferred for the biggest part of the testing project, because of its code accuracy, which is achieved by its strict coding style.

In the updated project, the testing framework that was selected is the one named **“Playwright”**. Playwright is a new testing framework of Microsoft. It is compatible with many programming languages, but the default, programming language, and most preferred by its developers is the one named TypeScript. Playwright is a testing framework with plenty of advantages. In the case that testers use the programming language TypeScript, the testing project does not need any extras libraries. Playwright is capable of setting, organizing, creating, and executing tests. Playwright has its own settings and reports. By using Playwright, a tester is able to easily capture screenshots and videos and add them later in the Test Report. Finally, Playwright provides to a tester the possibility to easily write readable, organized, detailed, and very fast test scenarios and test cases.

An extra framework that was selected for the updated testing project is the one named **“Cucumber”** and more specifically the JavaScript compatibility of this framework, the **“CucumberJS”**. This framework is very powerful and useful for testing projects that include Behavior Driven Development (BDD) methodologies. This framework is a testing framework, but not like the previous one. In fact, Cucumber uses Playwright in order to achieve the execution of the tests. Cucumber’s role is to search inside the testing project and read several files that include test documentation. These files are called feature files. Feature files, as it was mentioned before, include lines written in a language identifiable to a human, using several keywords (Given, When, Then, And). This language is called **“Gherkin”** language. Each line that includes a keyword, is called a step. Cucumber is responsible for reading each step from every feature file in the project and trying to find the respective part of source code that matches each step. The content of a feature file appears below:

*Feature: Check homepage search button*

*# Check homepage search button*

*@Scenario*

*Scenario: Check homepage search button*

*Given I am to "/" page*

*When I click the search button*

*Then I should redirect to the main s page*

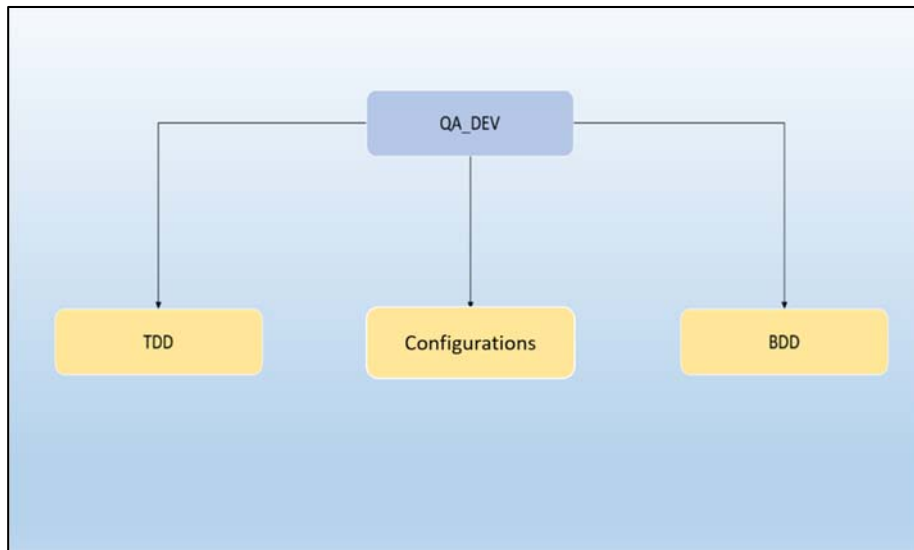
Finally, in order to store several data, that were needed in the automated testing process, JSON files were used. JavaScript and TypeScript include JSON libraries that provide the possibility to the programmers to handle JSON files.

### 3.2.4.2 Structure

Having fully analyzed the technology stack of the updated testing project, the structure of this testing project shall further be explained.

To begin with, the updated project slightly differs from the previous one. More specifically, the updated project is separated into two subprojects (in practice it is separated into two subfolders). The first subproject is related to the TDD methodology and the second one is related to the BDD methodology.

Each of these subprojects behaves separately, with its own structure, functionalities, results, and of course, testing methodology. Despite the fact that these subprojects act separately, they use the same configurations and information, which are stored in folders and files inside the main testing project. More than this, a main wrapper file (python file) exists, which is used in order to handle the possibilities of each testing framework in the aggregate, and is responsible to set, orchestrate and execute both of the testing subprojects.



**Figure 3.4 Updated version structure**

Firstly, the Test Driven Development (TDD) related folder is going to be analyzed.

In this subproject, in order for the setup of the Playwright framework to be achieved, a configuration file (typescript file) exists including the necessary information. Some of these configurations are:

- The directory in which the tests are stored
- The base URL of the website that shall be tested
- The type of the browser that shall be used
- The number of retries that shall occur in the case that a test fails
- Whether the execution shall be in a parallel mode
- The number of computer threads that shall be tethered from the system for a parallel execution

The format of this typescript file appears in the image below:

```
// playwright.config.ts
import { defineConfig } from '@playwright/test';

export default defineConfig({
  // Look for test files in the "tests" directory, relative to this configuration file
  testDir: 'tests',

  // Each test is given 30 seconds
  timeout: 30000,

  // Forbid test.only on CI
  forbidOnly: !!process.env.CI,

  // Two retries for each test
  retries: 2,

  // Limit the number of workers on CI, use default locally
  workers: process.env.CI ? 2 : undefined,

  use: {
    // Configure browser and context here
  },
});
```

**Figure 3.5 Playwright configurations typescript file**

Moreover, the TDD subproject includes three extra folders in order to organize the classes, tests and results of the project. This allocation makes the project easy to operate and maintain. More specifically these extra folders are the ones referred below:

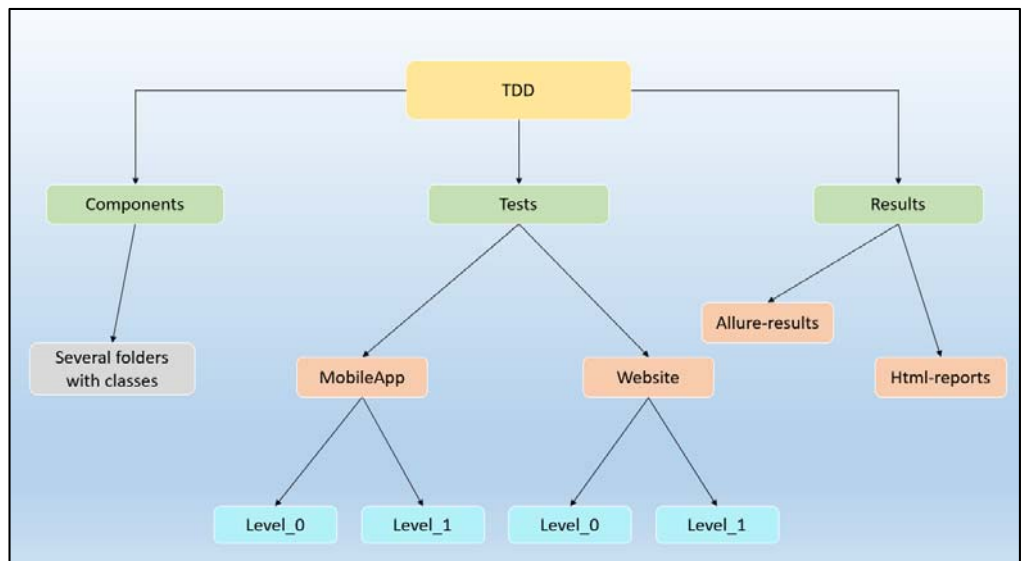
- Components
- Tests
- Results

In the first folder named “**Components**”, several classes exist. This subproject follows the “POM” (Page Object Model) concept. For this reason, each class represents a page and includes the respective locators and methods.

The folder named “**Tests**” includes tests separated into categories. More specifically, tests are separated into folders based on the device that they are going to evaluate, their priority, and type.



Finally, in the last folder, named “**Results**”, the results of the tests shall be stored. In this folder, two different kinds of results are contained. During a tests’ execution, several files are created, which are necessary in order for Allure and HTML (default report of the Playwright framework) reports to be generated.



*Figure 3.6 TDD structure*

Further down, the Behavior Driven Development (BDD) related subproject is going to be analyzed.

It is worth to be mentioned that Behavior Driven Development (BDD) folder, has a whole different structure from the one that the Test Driven Development (TDD) folder adopts.

To begin with, this folder includes two files that are very crucial for the execution of the technologies “Cucumber” and “Playwright”. More specifically, these two files are responsible for the initial steps of these technologies during a tests’ execution.

Additionally, the Behavior Driven Development (BDD) folder, as the TDD subproject, includes some extra folders, in order to organize the classes, features, step

definitions and results of the project. More specifically, these extra folders are the ones referred below:

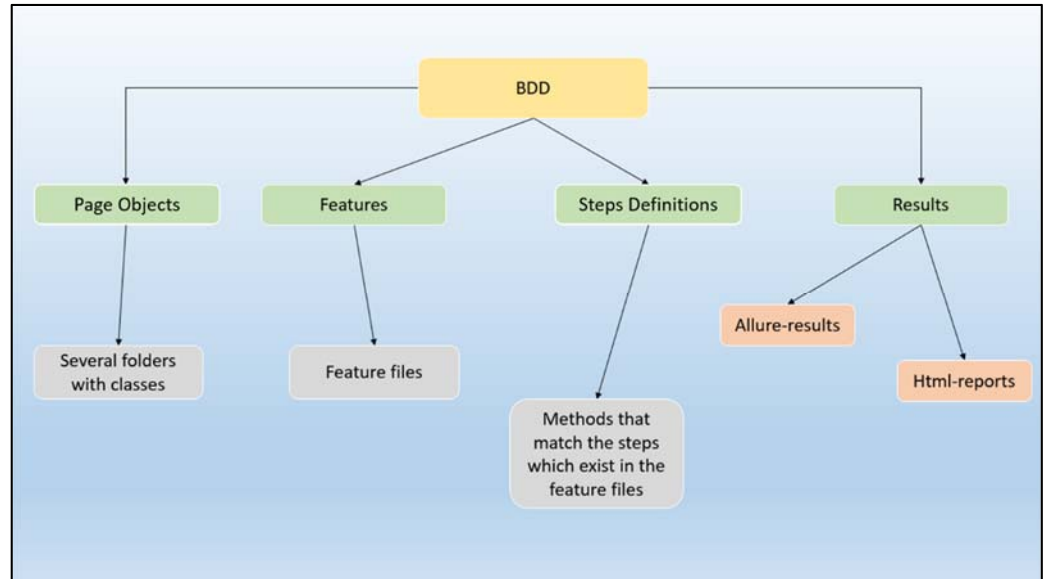
- Page Objects
- Features
- Step Definitions
- Results

The folder named “**Page Objects**” has the same role as the folder named “Components” for the TDD folder. As the “POM” is also followed in this project, the “Page Objects” folder includes several pages-classes.

The second folder, named “**Features**”, is the folder where all the feature files of the project are stored. All the files that represent documentation of the tests and include steps that are going to be executed, are stored in this folder.

The third folder, named “**Step Definitions**”, is the one that includes the source code that matches the steps of the aforementioned files. In particular, for every step of each feature file, a functionality exists, that is going to be executed when the necessary cucumber commands would be given.

Finally, in the folder named “**Results**”, the results of all the tests’ executions are stored. As in the TDD subproject, the results are generated in two different formats. A tester is able to evaluate the respective results via an Allure report or via an HTML report (default report of the Cucumber framework).



**Figure 3.7 BDD structure**

### 3.2.4.3 Usage

The use of this testing project and of course its subprojects (TDD, BDD) can be accomplished from the wrapper file. The execution of this file with the respective parameters enables the execution of the desired subproject.

To begin with, prior to the execution of the wrapper file, the user of this testing project should prepare the common configurations, the ones inside the folder “Configurations” as described in the chapter above. Afterwards, the user should select the desired tests for execution. These tests can be selected either isolated through a JSON file or collectively by selecting the desired tests’ category.

Then, in order to execute the TDD or the BDD project, and run the respective tests, some parameters in the execution command of the wrapper file, should be given. These parameters are:

- **results\_folder**: *The folder that the results of the tests are going to be stored*
- **main\_config\_id**: *The id of the configuration file that is going to be used*
- **application**: *The application/device that the tests are going to evaluate the product*
- **browsers**: *The browsers that the tests are going to evaluate the product*

- **tests\_level:** *The category of the tests that are going to be executed*
- **headless:** *The value of the headless mode that the user desires*
- **project\_type:** *The subproject that is going to be executed (TDD or BDD)*

```
pipenv run python runner.py --results_folder result_2023-02-09-00-29-23-02  
--application website --browsers 1 --headless true --project_type tdd
```

The above command will activate the TDD subproject and execute the following command:

```
tests_command =>  
  
=> "accept-cookies"  
  
npx playwright test --grep ${tests_command} --config=tdd/playwright.config.ts
```

In both subprojects, the role of the testing frameworks (Playwright, Cucumber) is to search for the desired tests, in the respective folder that the user declared, and execute them. Then, their role is to store the results of each execution in the declared position of the project. These results, afterwards, will be used in order to generate different types of reports.

It is worth to be mentioned, that both subprojects execute tests from a specific folder. The TDD subproject searches and executes tests that include source code. On the contrary, the BDD subproject searches and executes tests that include steps written in a human language. This gives the possibility to the employees of the company, which are not related to programming, to be responsible for some of the testing processes. More specifically, it enables them to create feature files that will be useful as documentation and generate test scenarios and test cases, according to their point of view. All they have to do is to add the created feature files in the BDD subproject and evaluate the results of their execution. A good strategy for the testers would be to “translate” the steps into source code beforehand. In this way, the

syntax is determined, and the evaluation is immediate. It is recommended for the upscale employees to be involved in the testing project only for the tests which are related to the more crucial and business-oriented features of the company's product. Their opinion on these tests would be valuable. The aforementioned operation shall increase the testing project's productivity, and consequently, the quality assurance of the company's product.

To sum up, the updated project is comprised of two subprojects. The first is related to Test Driven Development methodology and the second one to the Behavior Driven Development methodology. Both of them use common configurations stored in the main testing project. A user can handle them, from a wrapper file (python file) which is contained in the main testing project similarly to the configurations. After the tests' execution, several results are produced providing the user with the opportunity to generate several kinds of reports.

But how this operation and this collaboration can be succeeded?

In order for the employees of a company to easily collaborate and work with a testing project, like the one that was analyzed above, a QA Tool was made. This tool is able to automate and cover several processes of this collaboration. More details shall be found in the next chapter.

## 3.3 QA Tool

### 3.3.1 Description

QA tool already existed in the initial version of QA part of the testing project. Manual and automated testers are still using this tool in order to:

- add, edit, delete and set configurations,
- select, filter, and execute tests,
- view, check and evaluate the results.

The tool helps a lot the automated testers, which are able to handle easier and more productively the testing project. The QA tool is created as a full stack (FS) project, and includes a backend (BE) part with APIs and a frontend (FE) part with a friendly User Interface (UI). The updated attributes of the QA Tool shall be mentioned below.

### 3.3.2 Backend – APIs

The QA tool, as previously mentioned, includes a backend (BE) part with several APIs. Specifically, the backend part is created with the programming language “**Node.js**”. With the usage of the **Node.js** programming language, several APIs, based on the QA tool’s pages, are created. These APIs are separated into the following categories:

- Header APIs
- Tests APIs
- Configurations APIs
- Results APIs
- Feature APIs

The first category includes all the APIs that are related to the “**Header**” component of the site. Similarly, the rest of the categories include respectively, APIs for the “**Tests**” page, the “**Configurations**” page, the “**Results**” page, and the “**Feature**” page. The APIs are separated in this way, in order for the developers to ease their handling and editing.

There are several types of APIs, created in this project. In particular, “**GET**” APIs are created in order to get information. Moreover, “**POST**” APIs are created in order to send information. Finally, “**PUT**” and “**DELETE**” APIs are created in order to update and delete data respectively.

Due to low storage demands the BE part of this project is using JSON files and not a database. In these files, crucial information for the QA Tool’s processes are stored. These files are being used and edited from the abovementioned APIs.

### **3.3.3 Frontend – UI**

As far as the frontend (FE) part of the QA tool is concerned, a library of the programming language JavaScript, named “**React.js**” is used. This framework is very useful in order to create user interfaces. The whole project of the User Interface (UI) part, is based on this library and consists of several pages. Each one of these pages serve a different handling of the testing project.

This JavaScript framework is selected because of its advantages. Its most important advantage is the capability to separate the whole project into components. The whole FE part of the project is separated into components. Each component includes HTML elements with several functionalities. Some components represent pages, while some smaller represent elements. It is feasible for a component to include other components (bigger or smaller).

This structure is used because provides the developers with the possibility to create reusable components, handle easier the source code and, finally, create a maintainable project.

Like the BE part, the components are separated into categories. There are:

- Header APIs
- Tests APIs
- Configurations APIs
- Results APIs
- Feature APIs

The mindset behind this categorization is the same as the one that was mentioned in the BE part. Every category contains a main component that represents a page and several other components that are included to the main one.

### **3.3.4 Functionality**

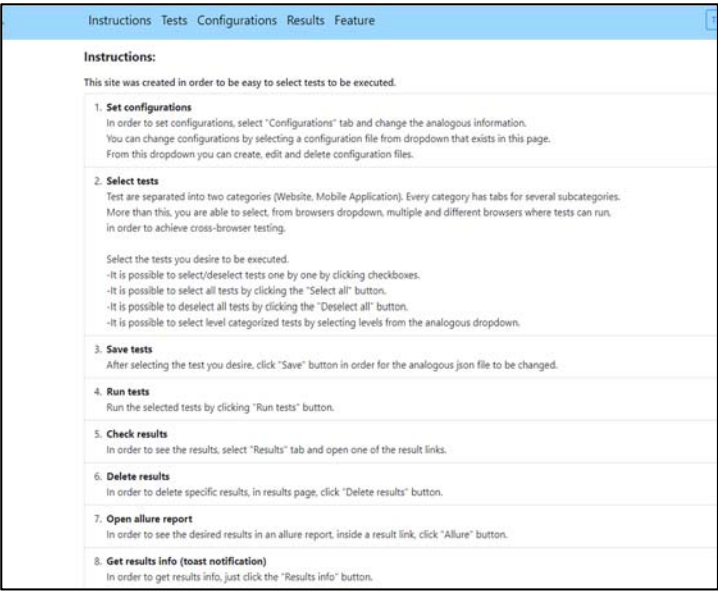
It the updated version of the testing project the QA Tool was changed and adapted to the new technologies, new programming languages, and new testing methodologies which accompany the new testing project. The initial QA tool was totally changed, except its structure, technology stack and User Interface (UI). Further the features of the updated QA tool and its functionalities shall be described in details.

The updated QA tool consists of five main pages and for each one of them, several APIs exist. In particular, these five main pages are the following:

- ✓ Instructions
- ✓ Tests
- ✓ Configurations
- ✓ Results
- ✓ Feature

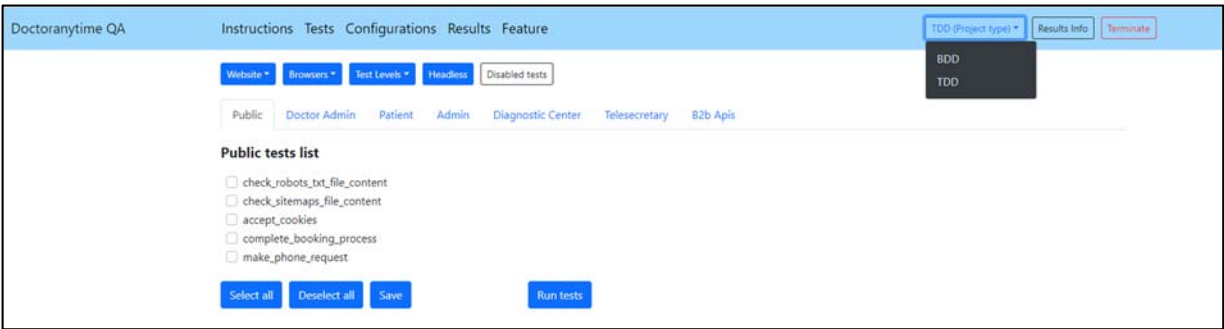


Firstly, the “**Instructions**” page, which is shown below, includes several instructions that help the user to handle the QA tool (**figure 3.8**).

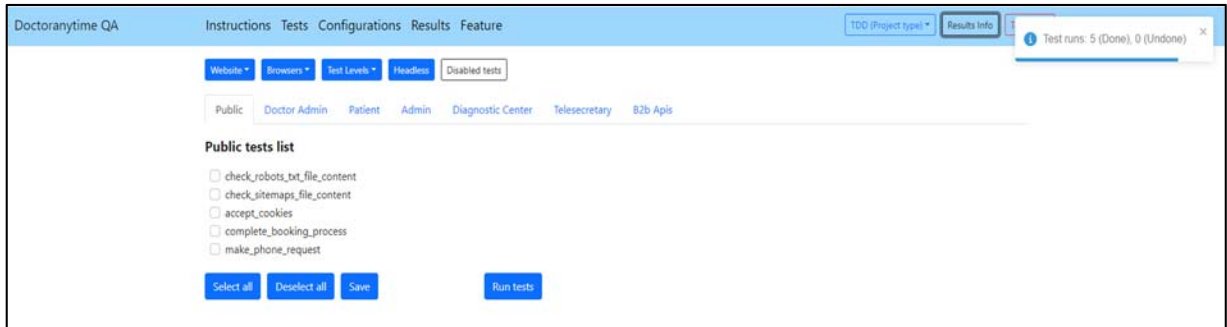


**Figure 3.8 “Instructions” page**

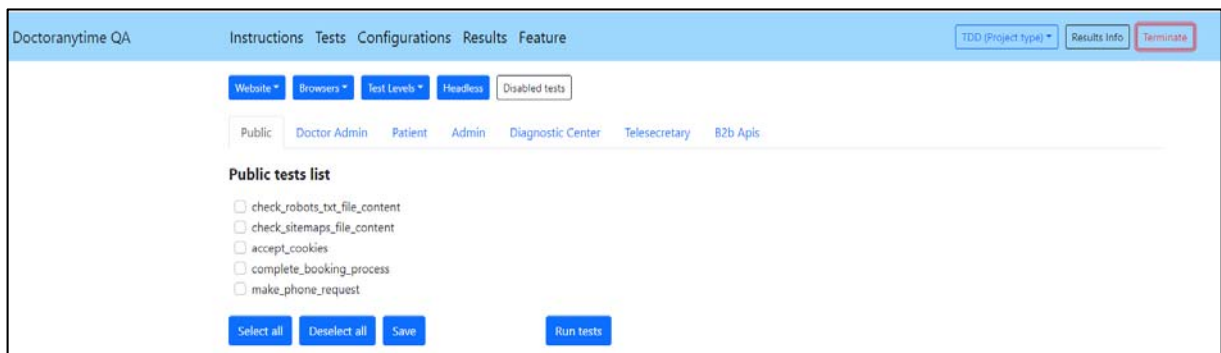
In the up and right corner of the tool, being presented below, there are three buttons. The first one is related to the testing methodology that the user desires to use (**figure 3.9**). The second button is responsible for informing the user about the status of the results (pending and done) with a toast (**figure 3.10**). The last button is responsible for terminating the program when the user clicks it (**figure 3.11**).



**Figure 3.9 Testing methodology dropdown button**



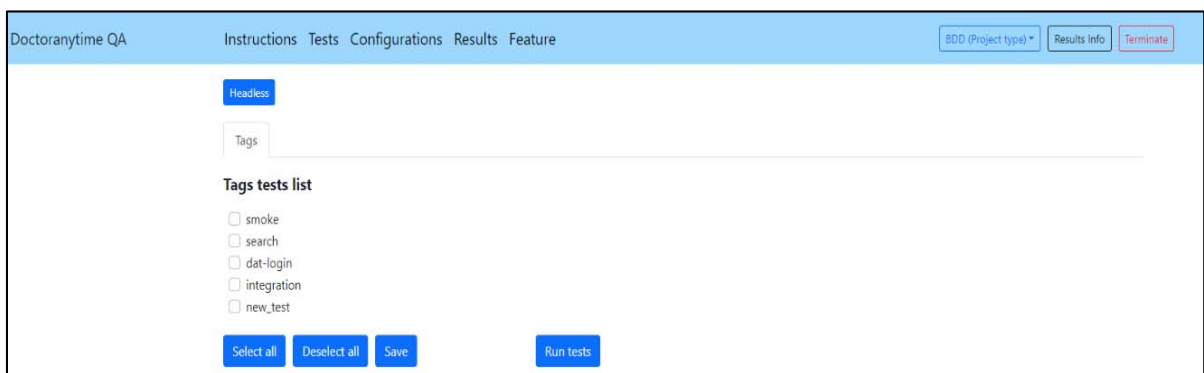
**Figure 3.10 Results information button**



**Figure 3.11 Terminate program button**

From “**Tests**” page, the user of the QA tool has the possibility to select to execute its desired tests. This page differs according to the selected testing methodology.

More specifically, when BDD methodology is selected, the page includes all the available tags of the tests. Thus, the tests that are going to be executed are the ones that include the tags that the user has selected. Moreover, from “Tests” page, the user is able to choose whether the tests are going to be executed in headless or non-headless mode (**figure 3.12**).



**Figure 3.12 BDD “Tests” page**

When the TDD methodology is selected, the “**Tests**” page has a whole different UI. In this case, the page includes the names of all the available tests. These tests are grouped by categories according to their testing content (**figure 3.13**).

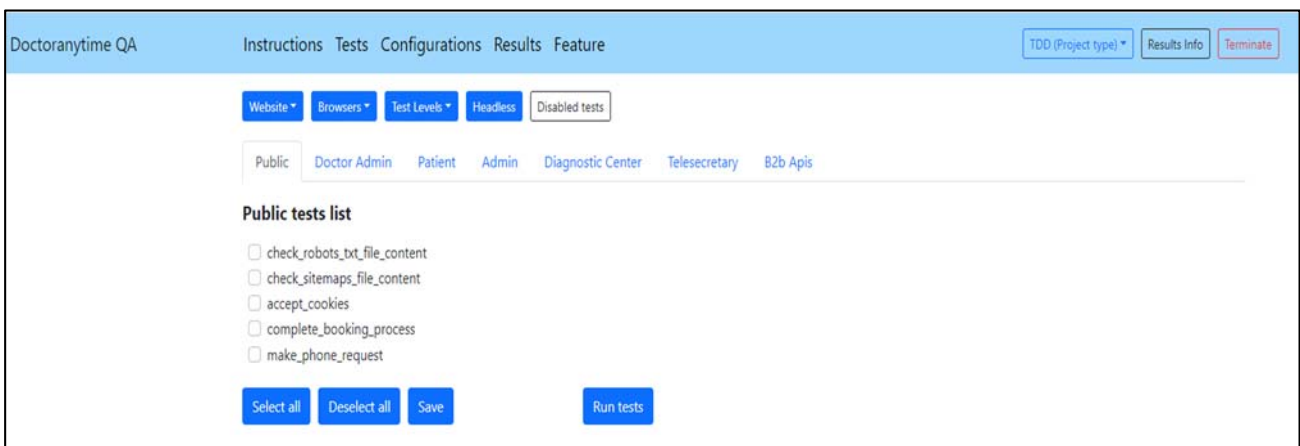


Figure 3.13 TDD “Tests” page

The selection of tests takes place either by selecting the respective checkboxes or by selecting the desired “Test Levels” from the dropdown list (**figure 3.14**).

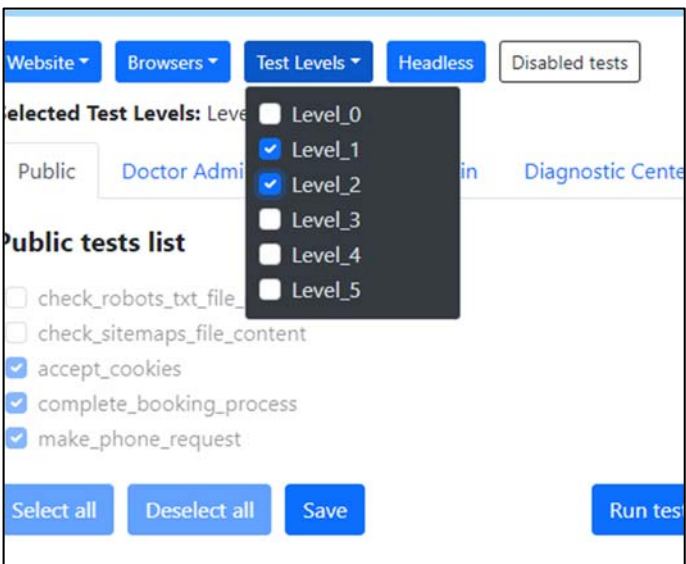
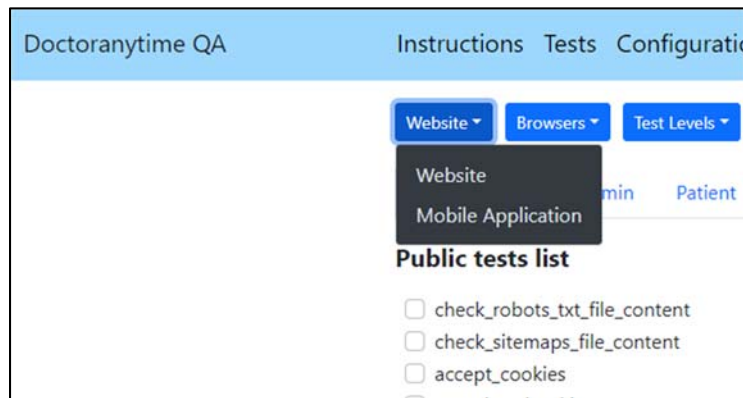


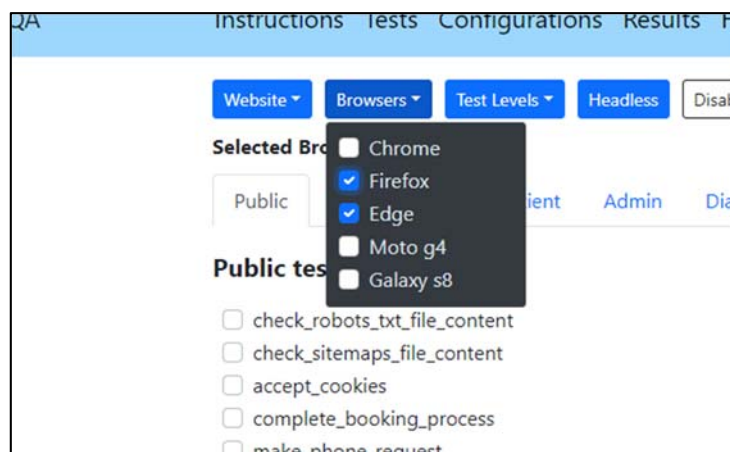
Figure 3.14 Testing levels dropdown

The tests are, also, categorized into mobile application and desktop tests. The user has the possibility to navigate into these categories, from a dropdown list (**figure 3.15**).



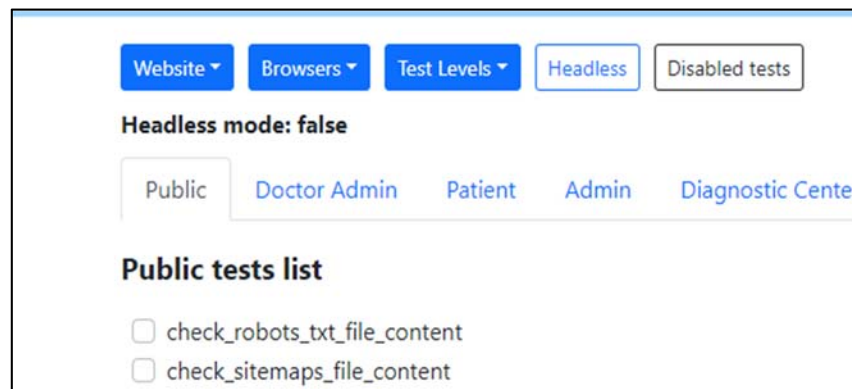
**Figure 3.15 Device dropdown**

More than this, from the same page, the user has the possibility to select the browsers and the mobile emulators where the tests are going to be executed (**figure 3.16**).



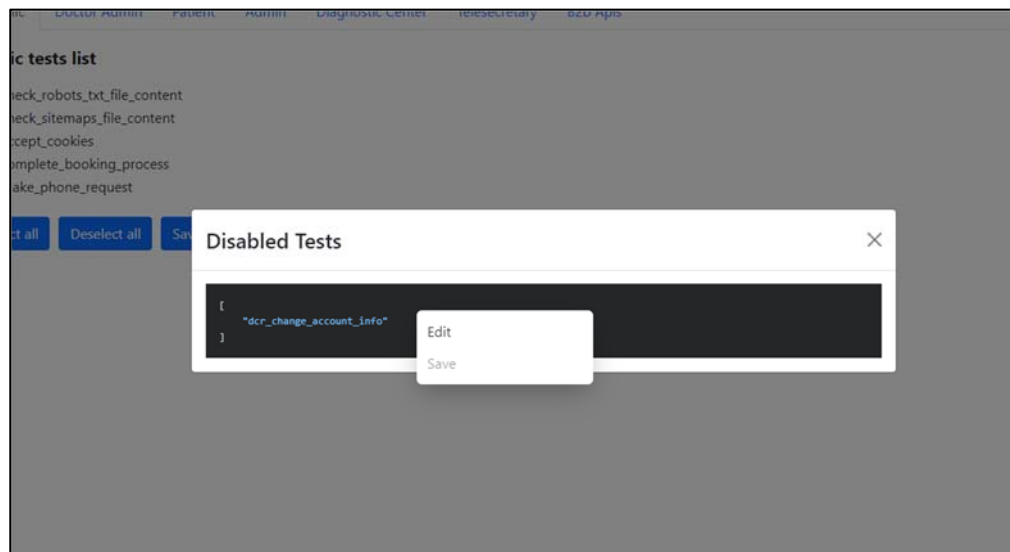
**Figure 3.16 Browser-Emulators dropdown**

Moreover, from this page, the user is able to select whether the tests are going to be executed in headless or non-headless mode (as in the BDD “Tests” page) (**figure 3.17**).



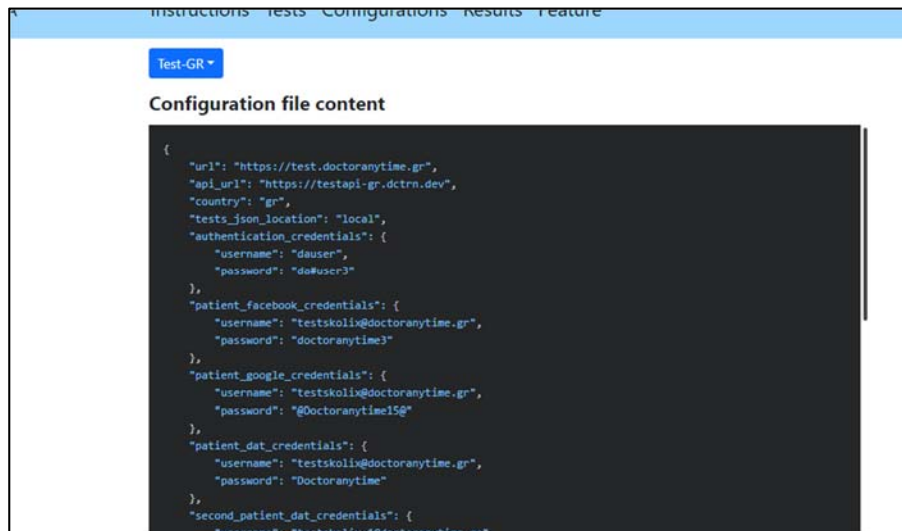
**Figure 3.17 Headless mode button**

Finally, from this page, the user has the possibility to disable and enable tests (**figure 3.18**).



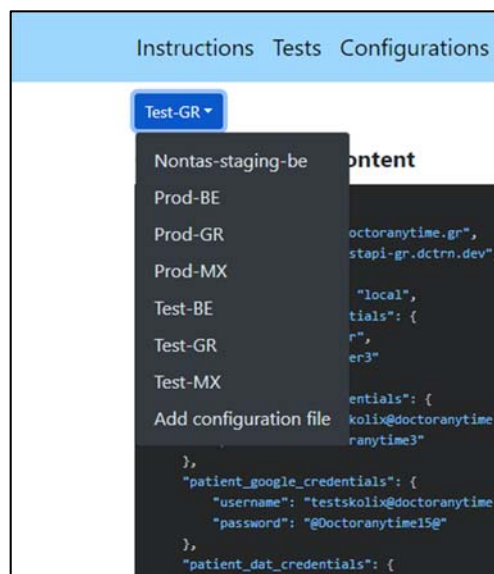
**Figure 3.18 Disable/Enable tests button**

From “**Configurations**” page, the user of the QA tool has the possibility to handle the configurations of the tests’ executions. These configurations include credentials of accounts and information about the site that is going to be tested (URL, API URL, country, etc.) (**figure 3.19**).



**Figure 3.19 “Configurations” page**

In particular, the user is able to select an already created configuration file from a dropdown list (**figure 3.20**).



**Figure 3.20 Configurations dropdown**

Moreover, there are the possibilities to create a new configuration file (**figure 3.21**), edit and delete an already created configuration file (**figure 3.22**).

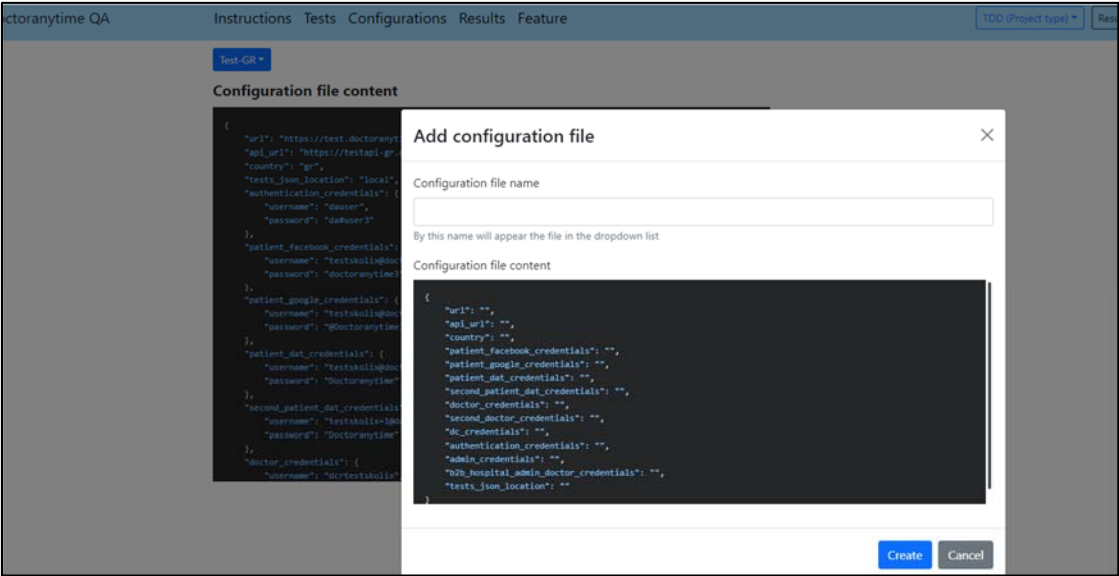


Figure 3.21 Add configuration file modal

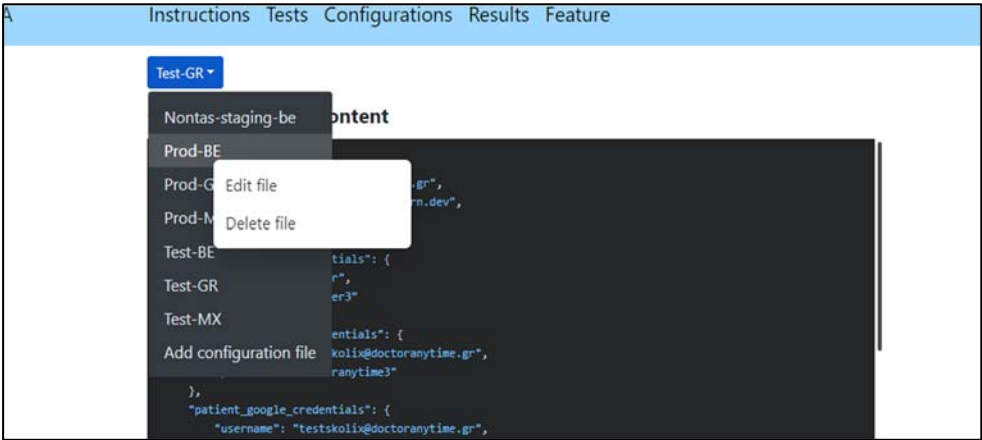
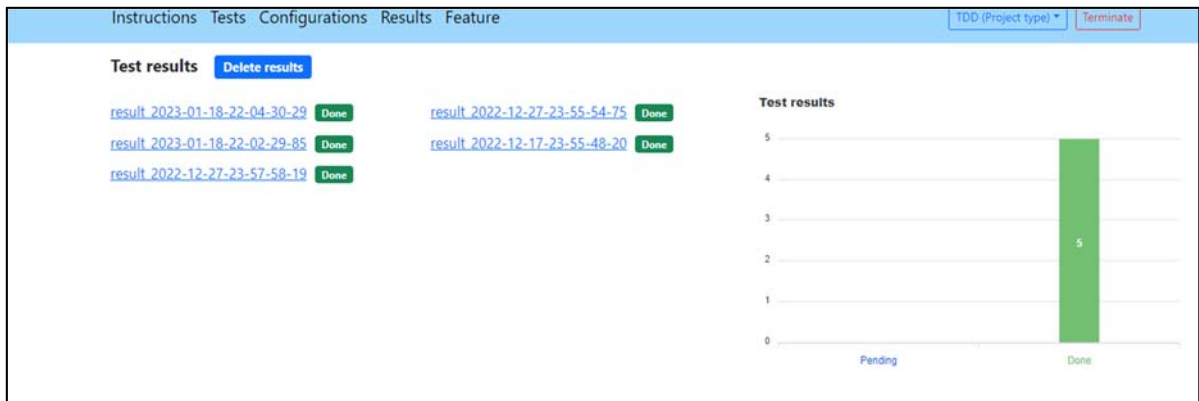


Figure 3.22 Edit/Delete configuration file

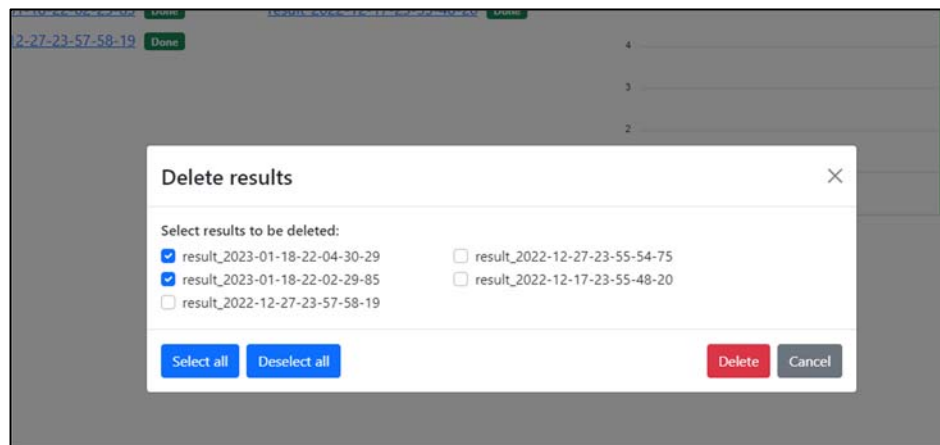
The “**Results**” page has the same structure for both testing methodologies. The user of the QA tool has the possibility to see and check all the results of the tests’ runs. This page contains a chart with the “pending” and the “done” results. The results are shown in this page with the form of a hyperlink, accompanied by their respective running status

(“pending” or “done”). Finally, a “delete results” button exists, in case the user desires to delete some or all of the results (**figure 3.23**).



**Figure 3.23 “Results” page**

Clicking the “Delete results” button (**figure 3.23**), a modal appears with all the results providing the possibility to the user to select some or all of them and, then, delete them (**figure 3.24**).



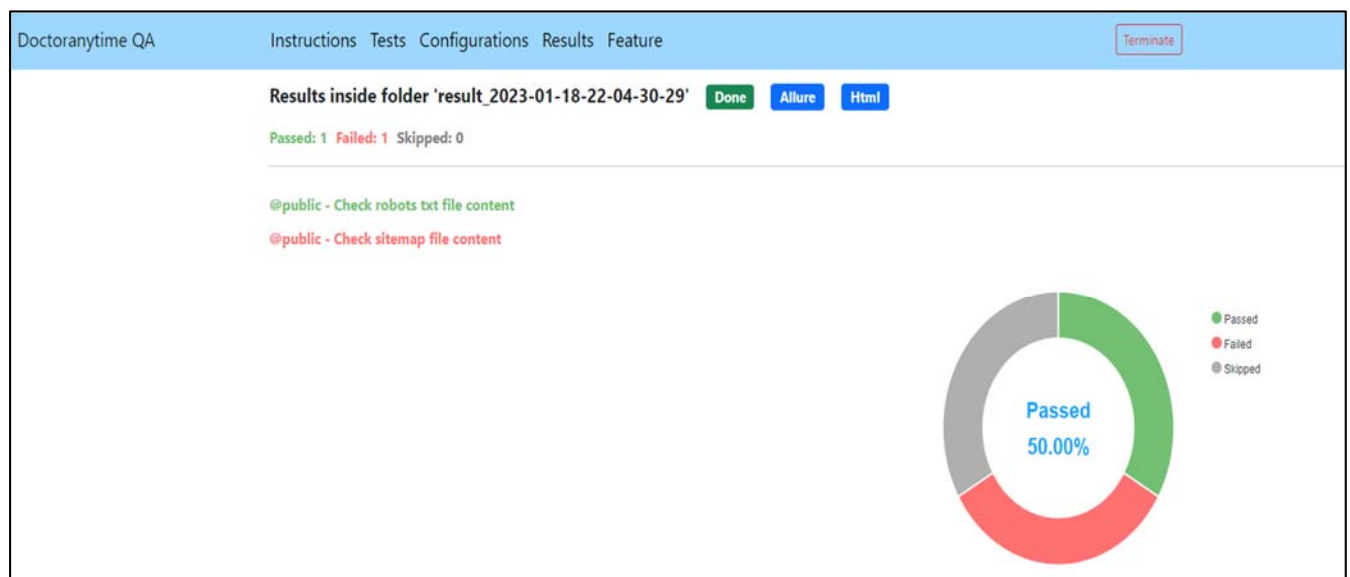
**Figure 3.24 Delete results modal**



Clicking a result hyperlink, which was mentioned before (**figure 3.23**), the user is redirected to the selected tests' execution results page, in which the user shall find a pie chart presenting the success rate of the tests' run. Moreover, the user shall find several information about the number of the passed, broken and skipped tests, as well as, the status of the tests' execution ("pending" or "done") (**figure 3.25**).

Furthermore, two buttons are available for the user, the "Allure" and the "Html". Each button opens a new page with a specific type and format of results' report (**figure 3.25**).

Last but not least, the user can find the tests' names when the selected testing methodology is TDD, or the scenarios' names when the selected testing methodology is BDD. These names will have a different color according to their result's value, i.e. green for passed, red for broken, grey for skipped (**figure 3.25**).



**Figure 3.25 Tests' execution results page**

Clicking the “Allure” button (**figure 3.25**), a new page will open with an Allure report of the selected tests’ execution results (**figure 3.26**).

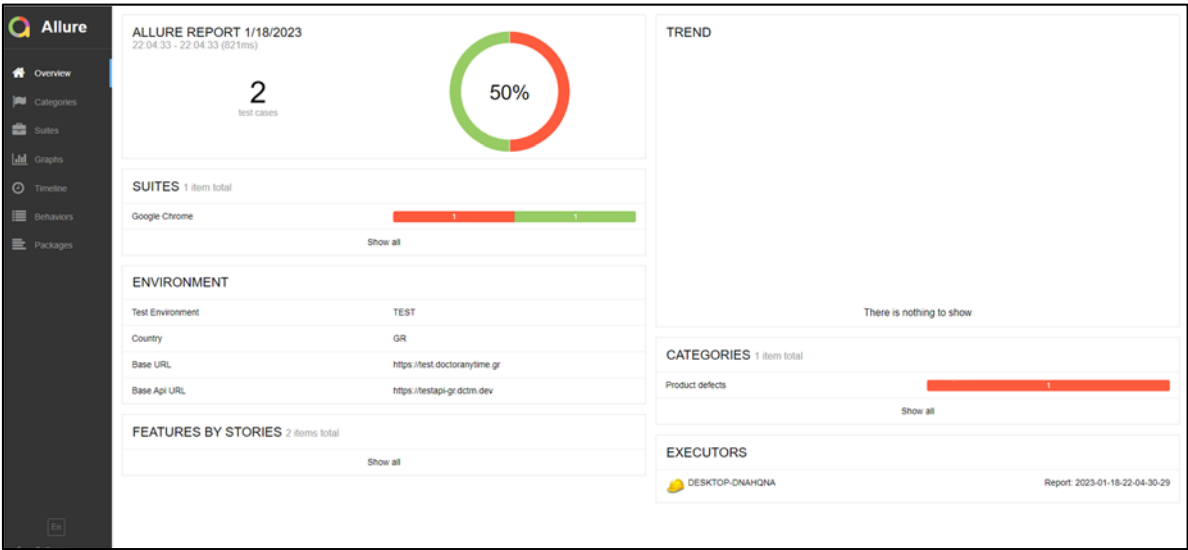


Figure 3.26 “Allure” report

Likewise, clicking the “Html” button, a new page will open with an Html report of the selected tests’ execution results. In this case, the report will not be the same for both testing methodologies. For TDD, the Html report will be the report which is provided by the Playwright framework (**figure 3.27**) (**figure 3.28**) and for BDD, the Html report will be the report which is provided by the Cucumber framework (**figure 3.29**).

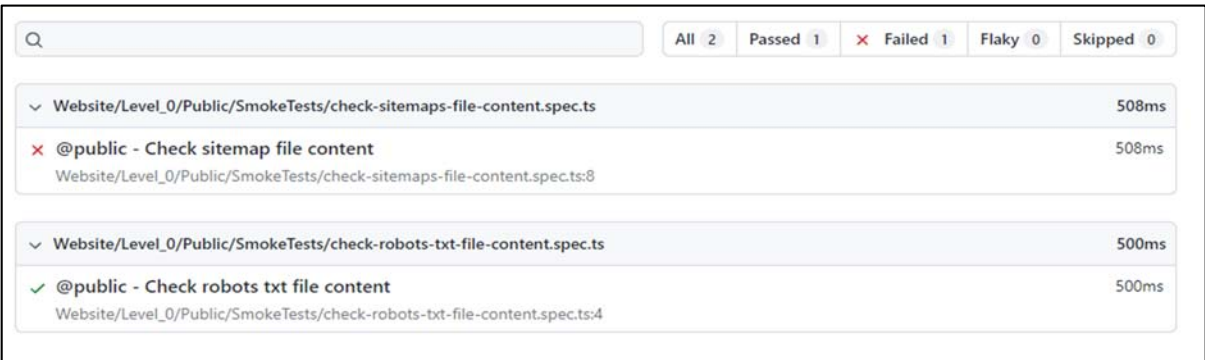


Figure 3.27 “Playwright” report (a)

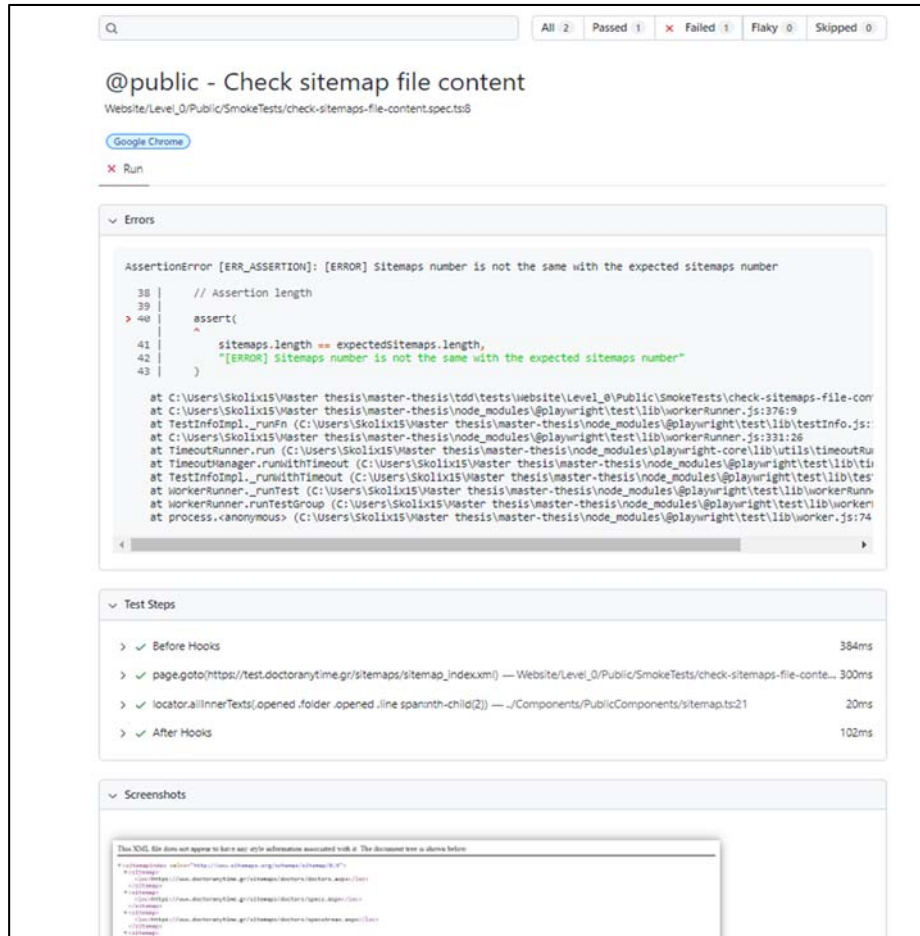


Figure 3.28 “Playwright” report (b)

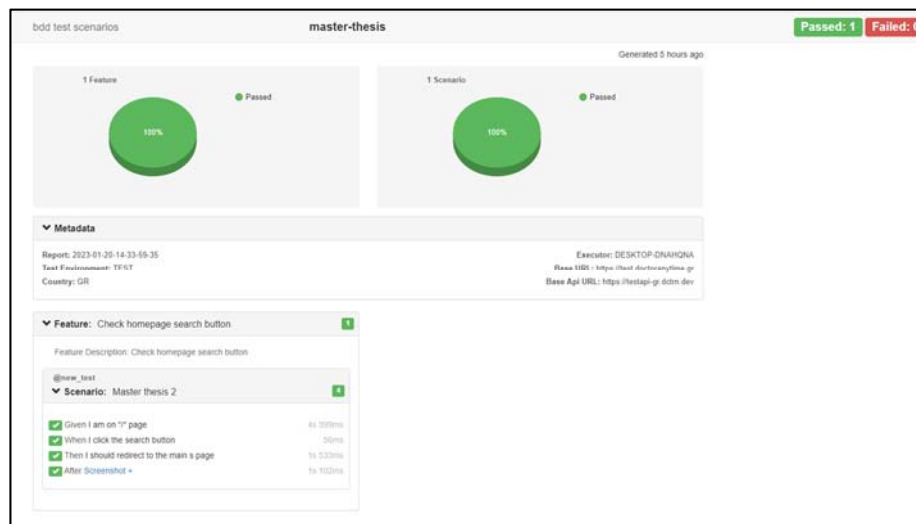


Figure 3.29 “Cucumber” report

Finally, the **“Feature”** page shall be presented and analyzed. In this page, the core value and usefulness of the present master thesis is effortlessly shown. This page connects the non-developers with the testers of a company. The non-developers shall be able to create feature files and, then, send them to the testers, in order to include them in the testing project. These feature files shall be generated by the usage of predefined steps.

In the same page, non-developers can enter the feature file name, the feature name and the feature description in the respective inputs. Moreover, they can enter the feature file’s content in the main text area. Also, they are able, also, to save temporarily the feature’s information in order to be able to navigate inside the QA tool without losing the written data. The content of the feature file can be selected from a dropdown list with the predefined steps (**figure 3.30**).

Doctoranytime QA    Instructions   Tests   Configurations   Results   Feature    TDD (Project type)   Results Info   Terminate

**Create feature**   Clear feature

**Feature file information**

Feature file name   Feature name   Feature description   Save feature info

**Feature file content**

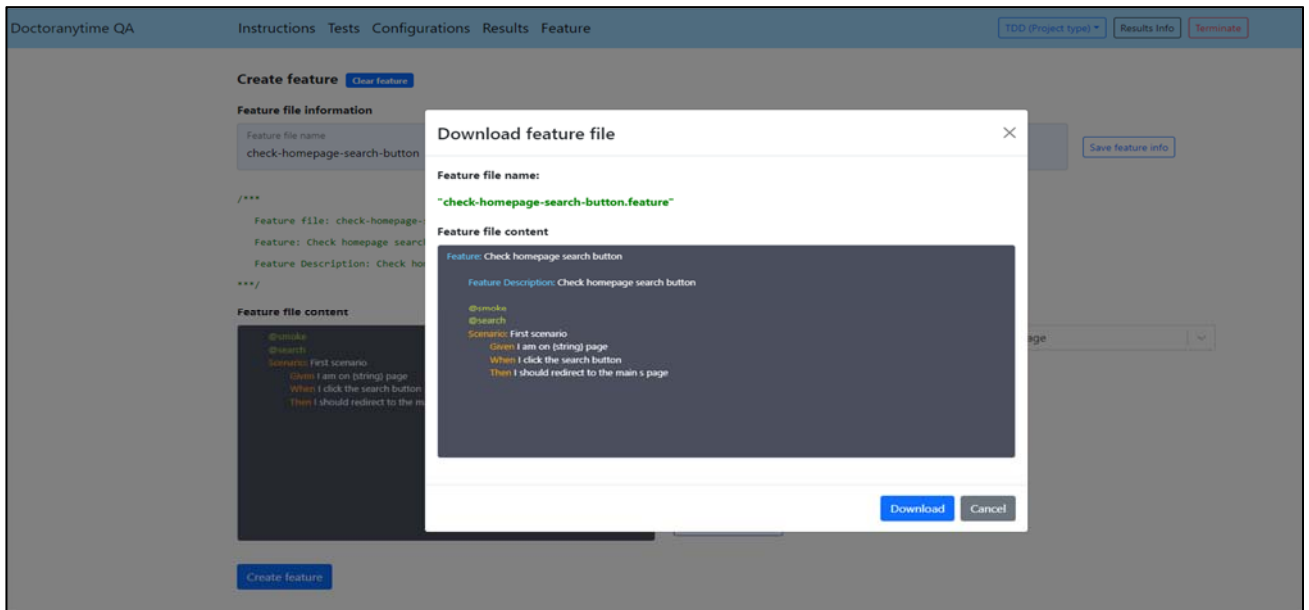
@search  
@smoke  
Scenario: Check homepage search button  
  Given I am on homepage  
  When I click the search button  
  Then I should redirect to the main's page

Add feature scenario   Select...   Save feature content

Create feature

**Figure 3.30 “Feature” page**

Finally, by clicking the “Create feature” button (**figure 3.30**), a confirmation modal appears containing the information, that was already added by the user, for the new feature file. At this stage, the user receives the final format of the feature file that shall be created. By clicking the “Download” button of this modal, the feature file shall be downloaded to the “Downloads” folder of the user’s computer (**figure 3.31**)



**Figure 3.31** Feature's confirmation modal

### 3.3.5 Tests execution example

In this chapter, a real example is going to be mentioned, in order for the reader to understand the usage of this tool. If the user desires to execute several tests for the Greek site of Doctoranytime's product and then check the respective results, he/she has the possibility either to execute tests on production site (the one that end-users can see), or the test site (the one that is used by the employees of the company in order to ensure that the new features and changes, work as expected). In this example, the tests are going to be executed in the test environment. The procedure that the user has to follow shall be explained.

To begin with, the user who wants to use this tool, all he/she has to do is double-click the tool's icon existing as a shortcut at the user's computer (**figure 3.32**).



*Figure 3.32 QA Tool icon*

The tool is auto-updated. Thus, every time the tool is executed, it runs the necessary updates, as shown in the image below (**figure 3.33**).

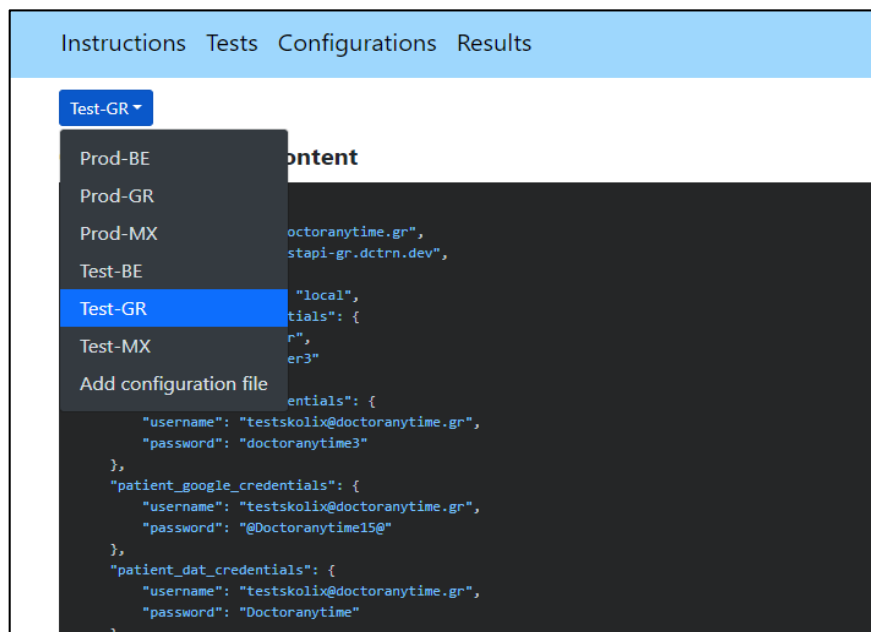
```
npm install
Current working directory: C:\Users\Skolix15\Master thesis\master-thesis
Requirement already satisfied: pipenv in c:\users\skolix15\appdata\local\programs\python\python310\lib\site-packages (20.23.2.4)
Requirement already satisfied: virtualenv-clone>=0.2.5 in c:\users\skolix15\appdata\local\programs\python\python310\lib\site-packages (from pipenv) (0.5.7)
Requirement already satisfied: certifi in c:\users\skolix15\appdata\local\programs\python\python310\lib\site-packages (from pipenv) (2022.12.7)
Requirement already satisfied: virtualenv in c:\users\skolix15\appdata\local\programs\python\python310\lib\site-packages (from pipenv) (20.17.1)
Requirement already satisfied: setuptools>=36.2.1 in c:\users\skolix15\appdata\local\programs\python\python310\lib\site-packages (from pipenv) (63.2.0)
Requirement already satisfied: distlib<1,>=0.3.6 in c:\users\skolix15\appdata\local\programs\python\python310\lib\site-packages (from virtualenv->pipenv) (0.3.6)
Requirement already satisfied: platformdirs<3,>=2.4 in c:\users\skolix15\appdata\local\programs\python\python310\lib\site-packages (from virtualenv->pipenv) (2.6.0)
Requirement already satisfied: filelock<4,>=3.4.1 in c:\users\skolix15\appdata\local\programs\python\python310\lib\site-packages (from virtualenv->pipenv) (3.8.2)
Current working directory: C:\Users\Skolix15\Master thesis\master-thesis
Current working directory: C:\Users\Skolix15\Master thesis\master-thesis
HEAD is now at d580e12 fixes
Already up to date.
C:\Users\Skolix15\virtualenvs\master-thesis-NLST3GDi\Scripts\python.exe: No module named pipenv
Current working directory: C:\Users\Skolix15\Master thesis\master-thesis
Current working directory: C:\Users\Skolix15\Master thesis\master-thesis\QA_tool
Current working directory: C:\Users\Skolix15\Master thesis\master-thesis\QA_tool\FrontEnd
[ ] / reify:fsevents: sill reify mark deleted [
```

*Figure 3.33 Auto-updates of QA Tool*

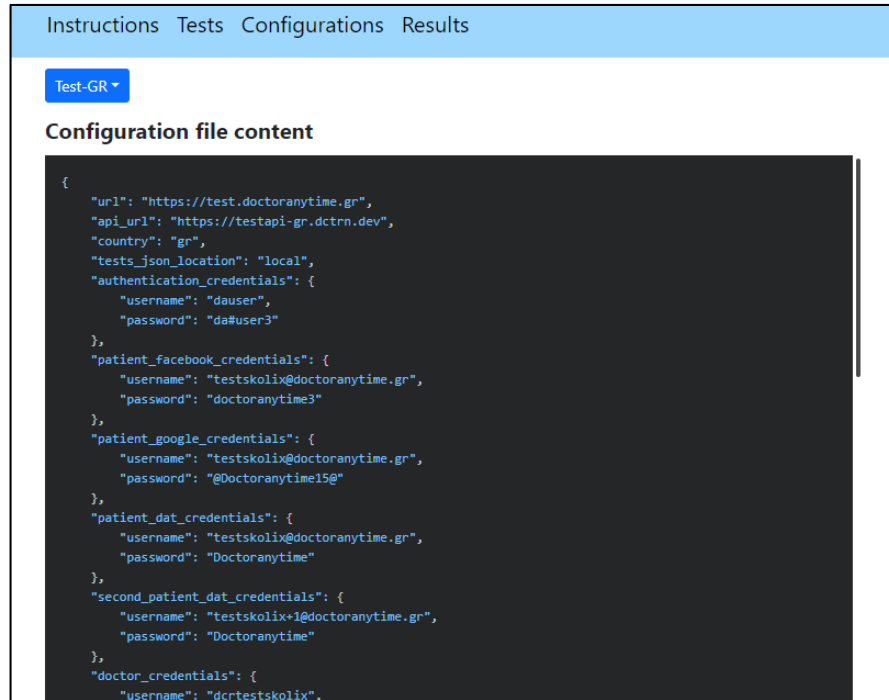
When the tool is ready, the first page that will open is the “**Tests**” page.

Firstly, the user needs to set the desired configurations. For this example, as it was mentioned before, the user has to choose from the configurations dropdown the option that is related to the Greek site and more specifically, the one related to the test environment of it.

By clicking from the header of the site the “Configurations” option, the user shall be redirected to the respective page. In this page, the user is able to click the configurations dropdown list and then choose the desired configuration option (**figure 3.34**) (**figure 3.35**).

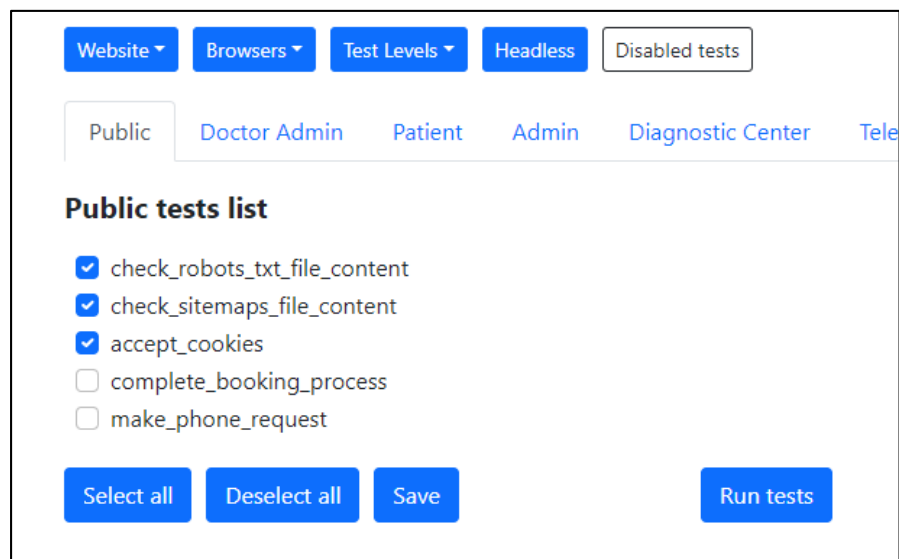


**Figure 3.34 Selecting configurations (a)**



**Figure 3.35 Selecting configurations (b)**

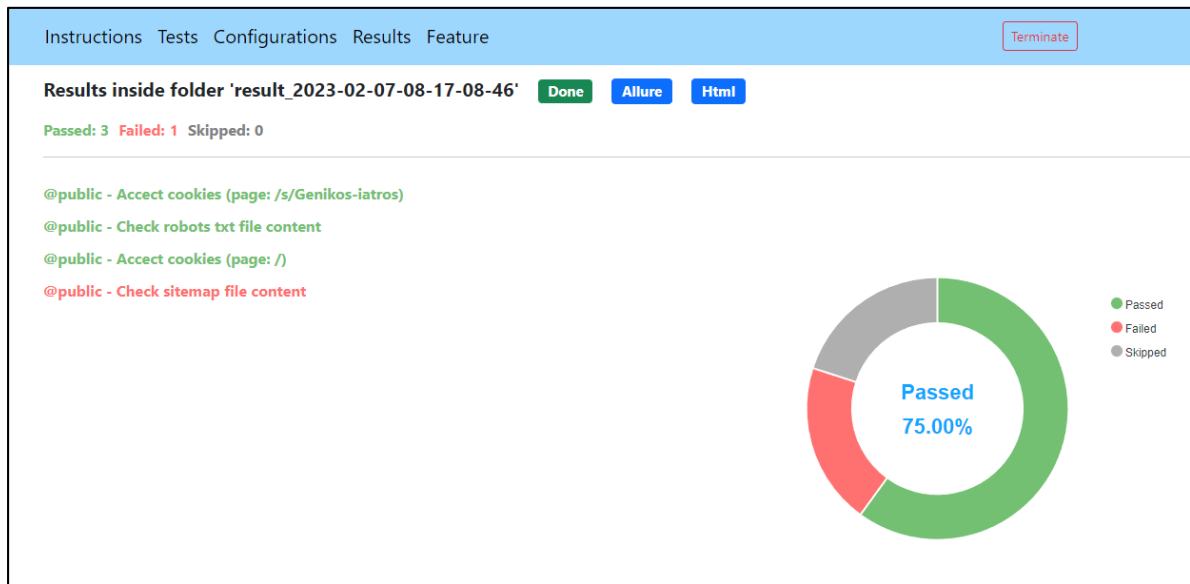
Then, the user should go back to the **“Tests”** page and select the tests for execution. The ways of selecting and filtering the tests have been described before. In the present example, the user selects individually three tests for execution (**figure 3.36**).



**Figure 3.36 Selecting tests**

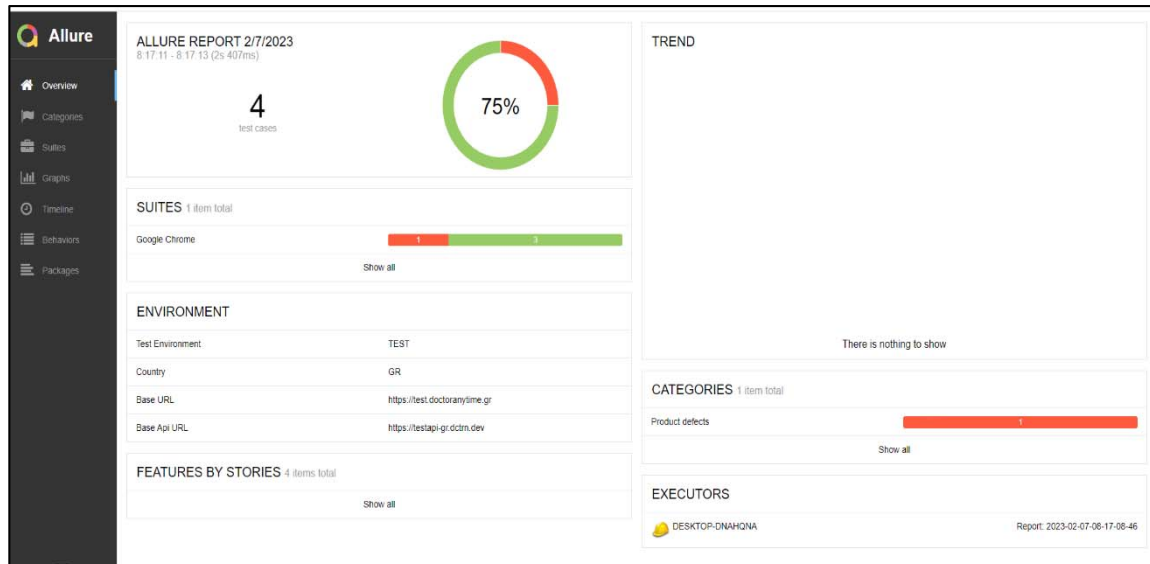


When the tests' execution is finished, a "Results" page with the respective results shall open automatically. From there, the user shall be able to check which and how many tests have passed, failed or skipped and their rate in a pie chart. (**figure 3.37**).

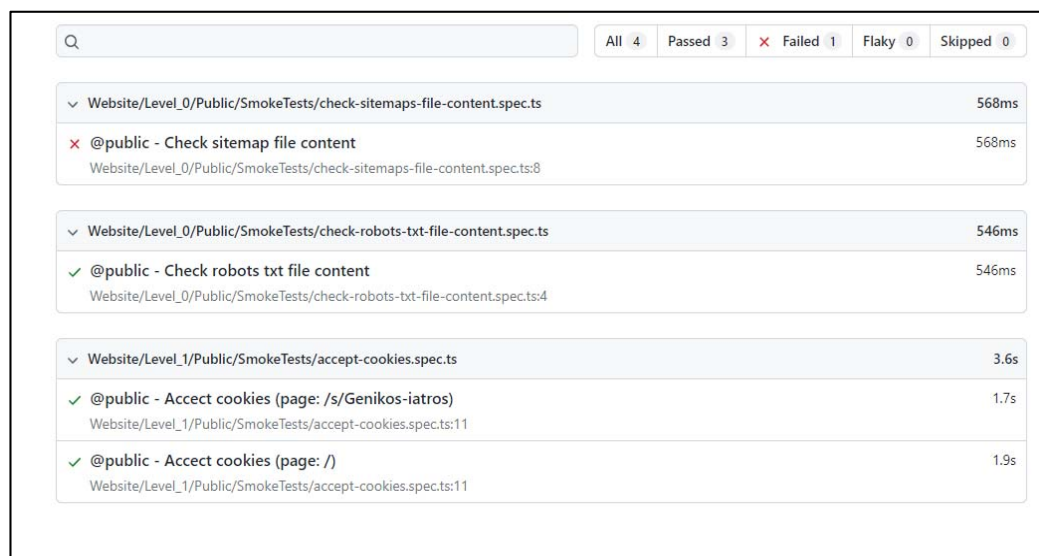


**Figure 3.37 Tests execution results**

Finally, in the case that the user desires to access a more analytical report, he/she has the choice to select between two extra kinds of report, the "**Allure**" and the "**HTML**". In the images below, these two reports appear. More specifically, in the first image the "Allure" report appears, which includes plenty of details for the tests' execution (screenshots, parameters, execution time, messages, charts, timelines, etc.) (**figure 3.38**). In the second image the "HTML" report appears, which also contains many details related to the tests' execution (**figure 3.39**).



**Figure 3.38 Results in “Allure” report**



**Figure 3.39 Results in “HTML” report**

A tests’ execution example with the usage of the QA Tool was provided. As it is evident, this tool has many functionalities and possibilities, which enable the user to operate successfully and easily the Testing Project.

### 3.3.6 Feature creation example

In another case, where the user is not a tester or a developer, but an employee of the higher levels of the company (for example a Product Owner, Leader, Manager, etc.), the QA Tool provides some extra possibilities. In particular, in case the user desires to test a new feature that is going to be uploaded on the production site, he/she can open the QA Tool as it was mentioned in the above chapter, and navigate into the **“Feature”** page of this tool.

On this page the user is able to create a new feature file with several steps, the ones that he/she evaluates as necessary, in order to check the functionality of the new feature that has been implemented. The available steps that can be selected by the user, appear in a dropdown list. These steps are already created, i.e. “translated”, as a source code in the testing project. After fulfilling the desired feature file’s information (Feature file’s name, feature’s name, feature’s description), the user can create a test scenario of his/her desire. In particular, by clicking the button “Add feature scenario”, the user is able to add a new scenario with several tags. It is worth to be mentioned that the content of a tag, that can be filled by a user, has no limitation. This tag will characterize the test scenario and will make it easy for the responsible employees to understand its type and content. Then, the user is able to select the necessary steps for the test scenario. (**figure 3.40**). Finally, by clicking the “Create feature” button, the user is able to see the final format of the feature file and download it (**figure 3.41**).

Create feature Clear feature

Feature file information

Feature file name  
check-homepage-search-button

Feature name  
Check homepage search button

Feature description  
Check homepage search button

Save feature info

```

/***
  Feature file: check-homepage-search-button.feature
  Feature: Check homepage search button
  Feature Description: Check homepage search button
  ***
  
```

Feature file content

```

@smoke
@search
Scenario: Check homepage search button
  Given I am on (string) page
  When I click the search button
  Then I should redirect to the main s page
  
```

Add feature scenario

Save feature content

Feature file steps

Then I should redirect to the main s page

Create feature

Figure 3.40 Pre-submitted “Feature” page

Download feature file

Feature file name:

“check-homepage-search-button.feature”

Feature file content

```

Feature: Check homepage search button
  Feature Description: Check homepage search button

@smoke
@search
Scenario: Check homepage search button
  Given I am on (string) page
  When I click the search button
  Then I should redirect to the main s page
  
```

Download

Cancel

Figure 3.41 “Feature” page final step

### **3.3.7 Conclusion**

To sum up, the QA Tool contains all the aforementioned pages, as they were shown in the previous images. From these pages, the user of this tool has the possibility to set and handle configurations and to filter, select and execute tests. More than this, the user is able to see and evaluate results and finally, to create a feature file that will be included in the project, in order to increase its test scenarios and test cases.

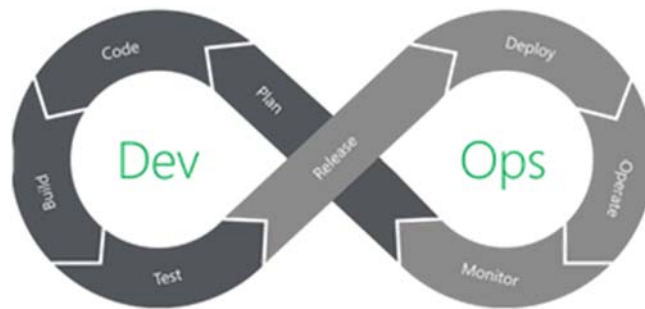
The usage of this tool (QA tool) makes feasible the immediate communication and collaboration between the higher and lower departments of a company. Moreover, the user of this tool has the possibility to handle in an easier way the testing project and achieve its improvement. Finally, the usage of this tool can save time for the employees of a company, increase their productivity and, of course, to increase the Quality Assurance (QA) of the company's product.

## 3.4 DevOps in QA

### 3.4.1 Pipelines

The whole testing project was connected with some pipelines in order to succeed the “**Continues Delivery**” operation. The tests are not only able to be executed through the QA tool, but also at an external computer (a virtual machine) on a predetermined and daily base and of course, before and after every release.

This operation is automated and it is the beginning of the processes needed in order for the company to succeed the “**Continues Integration**” operation.



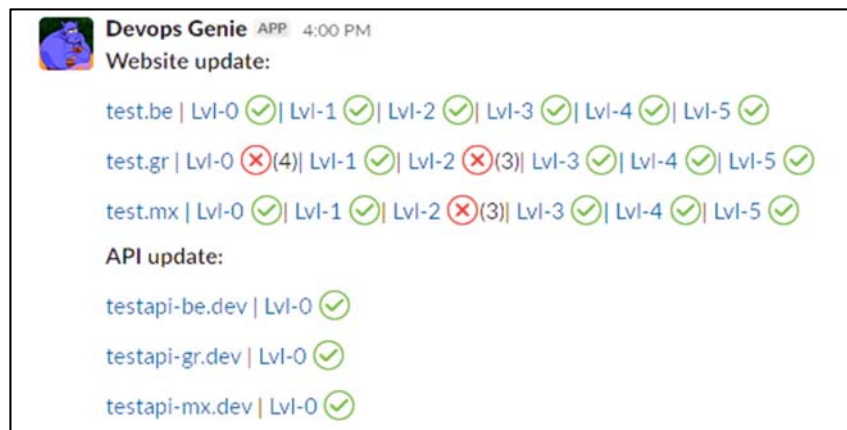
**Figure 3.42 DevOps**

[<https://medium.com/taptuit/the-eight-phases-of-a-devops-pipeline-fda53ec9bba>]

### 3.4.2 Results – Notifications

The results of the tests that were executed by the pipelines are stored in the cloud. In this way, the employees of the company have the possibility to view and evaluate the results at any time. More than this, in this way, a history of the tests' results is stored.

The “**DevOps**” department has, also, created a notification process. In particular, the results' generation part of the testing project, contained in the pipelines, was connected with an application named “**Slack**”. There, all the results appear and every employee of the company is able to view them (**figure 3.43**).



*Figure 3.43 Slack results notification*

## 4. Conclusions

### 4.1 Summary and conclusions

In the present master thesis, an analysis of the meanings of **Testing**, **Quality Assurance**, and **Automated Testing** and their characteristics, took place. The testing process is very important for the companies that desire to grow and be a part of the global market. In particular, the testing process is the only way to achieve the aforementioned goal. Testers, manual or automated, are able to find errors and bugs of a product, way sooner than the user or the client of this product. In this way, the Quality Assurance of a company's product is going to be achieved, the image and reliability of this company in the market shall be established and its incomes shall be increased. Despite the fact that manual testing is very important even in our days, the key to achieve the aforementioned goals is the automated testing.

There are several types of automated testing. Each company chooses which of these types would implement and, of course, the number of tests that are going to create for each testing type. This decision is related to the needs of each company. For example, some companies that are smaller and their product is new in the market, it is possible that they would need more smoke tests and less integration and end-to-end tests. On the other hand, bigger companies it is possible to need plenty of tests for each testing type.

Like the test types, there are different testing methodologies. The most frequently used of the testing methodologies is the one named **Test Driven Development** (TDD). In this testing methodology, only developers and testers interact. In order for the collaboration between higher and lower departments of the company, in the testing process, to be achieved, the testing methodology named **Behavior Driver Development** (BDD) should be implemented.

In this process, the behavior of a test scenario or test case is written in a human language inside a file, named feature file. This file includes several steps. Each of these steps will be “translated” by testers into a language comprehensible by a computer (source code).



Someone can understand better this theoretical part, by reading the practical part of this master thesis. A testing project with automated software testing processes has been created with the usage of the aforementioned testing technologies.

In the testing project that has been created, the communication and collaboration of the testers and non-developers of a company was indeed successful. The result of the above was to increase the Quality Assurance of the company's (**Doctoranytime**) product.

Besides the testing project that has been created, the implementation of a tool has, also, been achieved (QA Tool) that gives the possibility to testers and non-testers to easily operate and handle the testing project. This tool embeds into its processes both TDD and BDD testing methodologies.

## **4.2 Future work**

Testing has no boundaries. Thus, the QA part of this project can accomplish a great improvement in the future. The testing framework that has been selected for this project, "Playwright", consists a technology able to succeed this improvement. More than this, there is still room for improvement of the QA Tool and the DevOps part of this project. The QA tool can increase its quality and usage by adding plenty of new features. Among these new features, the features related to the Behavior Driven Development (BDD) testing methodology, should take priority. Finally, regarding the DevOps part of the testing project, the process of "Continuous Delivery" has been created. The main aim of this testing project is still ahead and it is called "Continues Integration".

## Bibliography

- [1] A. ISTQB team, "/en/search/," 2023. [Online]. Available: <https://glossary.istqb.org/en/search/>. [Accessed 12 February 2023].
- [2] L. A. C. Gómez, "Analysis of the impact of test based development techniques (TDD, BDD, AND ATDD) to the software life cycle," IC-Online, Leiria, 2018.
- [3] A. ISTQB team, Certified Tester, Foundation Level Syllabus, 2018 V3.1 ed., Belgium: International Software Testing Qualifications Board (ISTQB), 2019, p. 93.
- [4] M. Rehkopf, "/continuous-delivery/software-testing/automated-testing," 2023. [Online]. Available: <https://www.atlassian.com/continuous-delivery/software-testing/automated-testing>. [Accessed 7 February 2023].
- [5] P. Vuollet, "/blog/what-is-test-automation/," August 2019. [Online]. Available: <https://www.testim.io/blog/what-is-test-automation/>. [Accessed 27 December 2022].
- [6] N. Sharma, "AN EXPLORATORY STUDY ON WEB APPLICATION AUTOMATION TESTING," 43, 2020.
- [7] AWS, "/what-is/api," 6 February 2023. [Online]. Available: <https://aws.amazon.com/what-is/api/>. [Accessed 5 February 2023].
- [8] J. Juviler, "/website/modal-web-design," 1 April 2022. [Online]. Available: <https://blog.hubspot.com/website/modal-web-design>. [Accessed 11 February 2023].
- [9] A. Wikipedia contributors, "/wiki/Software\_feature#cite\_note-2," 6 2 2023. [Online]. Available: [https://en.wikipedia.org/wiki/Software\\_feature#cite\\_note-2](https://en.wikipedia.org/wiki/Software_feature#cite_note-2). [Accessed 12 February 2023].
- [10] A. Marget, "/blog/development-test-environments," 20 January 2023. [Online]. Available: <https://www.unitrends.com/blog/development-test-environments>. [Accessed 11 February 2023].
- [11] K. T. Hanna, "/searchsoftwarequality/definition/release," March 2022. [Online]. Available: <https://www.techtarget.com/searchsoftwarequality/definition/release>. [Accessed 8 February 2023].
- [12] A. GitLab team, "/topics/devops," 2023. [Online]. Available: <https://about.gitlab.com/topics/devops/>. [Accessed 10 February 2023].

- [13] T. Hall, "/devops/devops-tools/devops-pipeline," 2023. [Online]. Available: <https://www.atlassian.com/devops/devops-tools/devops-pipeline>. [Accessed 10 February 2023].
- [14] A. IBM team, "/topics/continuous-integration," 2023. [Online]. Available: <https://www.ibm.com/topics/continuous-integration>. [Accessed 10 February 2023].
- [15] A. IBM team, "/topics/continuous-delivery," 2023. [Online]. Available: <https://www.ibm.com/topics/continuous-delivery>. [Accessed 9 February 2023].
- [16] L.-O. Damm, "Early and cost-effective software fault detection measurement and implementation in an industrial setting," Blekinge Institute of Technology, 2007.
- [17] Mark Fewster, Dorothy Graham, Software Test Automation, Effective use of test execution tools, Boston: Addison-Wesley Professional, 1999, p. 574.
- [18] C. Software, "@concisesoftware/everything-you-should-know-about-qa-in-software-development-the-beginners-guide," 17 September 2019. [Online]. Available: <https://medium.com/@concisesoftware/everything-you-should-know-about-qa-in-software-development-the-beginners-guide-3e7afacf607c>. [Accessed 29 December 2022].
- [19] K. Yasar, "/whatis/definition/software-testing," August 2022. [Online]. Available: <https://www.techtarget.com/whatis/definition/software-testing>. [Accessed 26 December 2022].
- [20] Ilze, "/blog/importance-of-software-testing," 3 July 2018. [Online]. Available: <https://www.testdevlab.com/blog/importance-of-software-testing>. [Accessed 5 February 2023].
- [21] P. Parthiban, "/blog/why-software-testing/," 14 April 2021. [Online]. Available: <https://www.indiumsoftware.com/blog/why-software-testing/>. [Accessed 1 February 2023].
- [22] E. Papadopoulos, "Synthetic Transaction and Up-time Monitoring Systems in IT," Thessaloniki, 2021.
- [23] S. Pittet, "/continuous-delivery/software-testing/types-of-software-testing," 2023. [Online]. Available: <https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing>. [Accessed 3 February 2023].
- [24] S. R. Choudhary, "Detecting Cross-browser Issues in Web Applications," *Georgia Institute of Technology, Atlanta, GA*, p. 3, 21 May 2011.
- [25] C.-B. D. C. System, "Marti Kaljuve," Tartu University Library, Tartu, 2013.

- [26] Mattia Fazzini, Alessandro Orso, Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, Urbana-Champaign IL USA: IEEE Press, 2017, p. 318.
- [27] M. Shi, "Software Functional Testing from the Perspective of Business Practice," *Ccsenet*, vol. 3, no. 4, p. 4, November 2010.
- [28] Isha Rana, Pooja Goswami, Himani Maheshwari, Sameer Mohammad, "A REVIEW OF TOOLS AND TECHNIQUES USED IN SOFTWARE TESTING," *Jetir*, vol. 6, no. 4, p. 6, April 2019.
- [29] V. Shinde, "/automation-testing-tutorial-2," 24 January 2023. [Online]. Available: <https://www.softwaretestinghelp.com/automation-testing-tutorial-2/>. [Accessed 27 December 2022].
- [30] Rasneet Kaur Chauhan, Iqbal Singh, "Latest Research and Development on Software Testing Techniques and Tools," *Inpressco (International Press Corporation)*, vol. 4, no. 4, p. 5, 1 August 2014.
- [31] R. Bellairs, "/blog/sca/what-static-analysis," 10 February 2020. [Online]. Available: <https://www.perforce.com/blog/sca/what-static-analysis>. [Accessed 4 February 2023].
- [32] Alexander S. Gillis, "/searchsoftwarequality/definition/End-to-end-testing," February 2018. [Online]. Available: <https://www.techtarget.com/searchsoftwarequality/definition/End-to-end-testing>. [Accessed 5 February 2023].
- [33] Wei-Tek Tsai, Xiaoying Bai, Raymond A. Paul, Weiguang Shao, Vishal Agarwal, "End-To-End Integration Testing Design.," *DBLP*, p. 7, January 2001.
- [34] K. Beck, Test Driven Development: By Example, Boston: Addison-Wesley Professional, 2002, p. 216 .
- [35] J. F. Smart, BDD IN ACTION, Behavior-Driven Development for the whole software lifecycle, Manning Publications, 2014, p. 384.