**Testing Frameworks**

# Introduction

Testing is an essential part of the process of developing a software application. It essentially ensures that the features included function as expected and many of the bugs or issues that users may encounter are detected before delivery and addressed by developers. Furthermore, tests ensure that already existent finished features, developed and tested, have the same functionality event after other features are added, that may impact the current functionality of the already existing.

It is important to understand that this document will focus both on testing the back-end and the front-end functionality of the application, meaning that the user will not encounter bugs related to either the interface that he or she is interacting with, nor will there be any on the processing of the data that is received by the interface.

Contents

[Introduction 1](#_Toc98596841)

[1. Selection Criteria 2](#_Toc98596842)

[2. Back-End Testing 2](#_Toc98596843)

[2.1. Description 2](#_Toc98596844)

[2.2. Goals of the framework 2](#_Toc98596845)

[2.3. Selected Frameworks Comparison 3](#_Toc98596846)

[2.4. Conclusion 4](#_Toc98596847)

[3. Front-End Testing 4](#_Toc98596848)

[3.1. Description 4](#_Toc98596849)

[3.2. Goals of the framework 4](#_Toc98596850)

[3.3. Selected Frameworks Comparison 4](#_Toc98596851)

[3.4. Conclusion 5](#_Toc98596852)

[4. Conclusions 5](#_Toc98596853)

# Selection Criteria

As mentioned in the *Introduction* chapter, this document will focus on both back-end and front-end testing solutions. Moreover, it is very important to consider that the frameworks chosen for this project will be selected based on already used technologies within the project, respectively NestJS Back-End Framework (JavaScript based Testing Frameworks) and ReactJS Front-End Framework (JavaScript based Testing Frameworks). As both parts of the application are JavaScript based, it means that one framework might fulfill the criteria for the two, but this is not essential, even if it might represent and advantage during testing.

# Back-End Testing

## Description

Back-end testing is a method or technique that examines the server of database side of web applications or software. The primary motive of performing this test is to test the application layer or database layer to make the software defect-free, and prevent deadlock, data corruption, or data loss. Usually, databases are validated for: ACID functions, CRUD operations, Schema, Migration, business rule conformance, security purposes and performance. ([TutorialsPoint](https://www.tutorialspoint.com/backend-testing-tutorial-what-is-tools-examples))

## Goals of the framework

The goals of the framework are:

* Unit testing integration
* Test full CRUD Operations
* Test Migration Schemas
* Test Security Purposes
* Test Data Mapping
* Test Data Integrity

Moreover, we are looking for a *stable* testing framework, that is *continuously developed* and *updated* as these are few of the main points for future development and extensibility of the project. Another big plus of the framework is *documentation*, as this would improve the development process.

## Selected Frameworks Comparison

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Jest | Mocha | Jasmine |
| Documentation | Very Good | Good | Good |
| Ease of Learning | Harder (for new developers) | Normal | Easy |
| Active Development | √ | √ | √ |
| Unit Test Integration | √ | √ | √ |
| Advantages | * Customizable assertions * Built with ReactJS * Can be used for both front-end & back-end * No need for configuration * Testing support for asynchronous code * Can perform regression tests by comparing screenshots of previous versions of the application (useful for front-end testing) | * Clear and simple API * Simple async support, including promises * Test coverage reporting | * Flexible with lots of frameworks and libraries * A rich set of built-in matchers that can add asserts to the test cases * Various CI servers support the Jasmine framework * Does not rely on any JavaScript framework, DOM, or browsers |
| Disadvantages | * Not many libraries and tooling are supported by Jest, compared to other frameworks * Snapshot testing with Jest is not so feasible for larger snapshot files containing thousands of lines | * Mocha requires more configuration * Auto-mocking and snapshot testing are not easy | * Configuration is a bit more complicated. Users must select an assertion or mocking library before using it * Asynchronous testing is difficult to achieve with Jasmine * Expect a specific suffix to all test files (\*.spec.js by default) |

## Conclusion

# Front-End Testing

## Description

Front End Testing is a testing technique in which Graphical User Interface (GUI), functionality and usability of web applications or a software are tested. The goal of Front-End testing is testing overall functionalities to ensure the presentation layer of web applications, or a software is defect free with successive updates. E.g., If you enter your name into the frontend of application, numbers should not be accepted. Another example would be checking the alignment of GUI elements. [(Guru99)](https://www.guru99.com/frontend-testing.html)

## Goals of the framework

The goals of the front-end testing framework are to simulate the user experience, test the interface to be as expected, and validate that the user gets the intended information, and that the visual aspect is the one expected by the developers. Essentially, it performs end-to-end testing, simulating the end user’s experience, according to real scenarios validating the system.

## Selected Frameworks Comparison

|  |  |  |  |
| --- | --- | --- | --- |
| Property | Cypress IO | NightwatchJS | WebdriverIO |
| Documentation | Very Good | Good | Good |
| Ease of Learning | Normal | Normal | Normal |
| Active Development | √ | √ | √ |
| Snapshots | √ | Only with other frameworks/ libraries (e.g., Mocha) | Includes Visual Regression |
| Advantages | * Comes with its own dashboard * Works in the browser, therefore dev tools can be used side-by-side with Cypress * Built-in parallelization and load balancing, making debugging easier * Cypress has access to front & back parts that enable it to modify everything coming in & out of the browser | * Easy to setup * Supports page object model, custom commands, custom assertions, and globals.js | * Easy to setup * Highly Customizable * Can be used to test native mobile apps * Includes Cross-Origin Support * Multiple tab support * It supports various types of selectors including CSS |
| Disadvantages | * Cypress is only available for Chrome, Firefox, Edge, Brave, and Electron browsers * Cypress will never have support for handling multiple browser tabs * It only supports the JavaScript framework for the creation of test cases | * The syntax is not clear, and it is not possible for the users to write a clear description in case of each and every step | * Confusing syntax * Includes pixel-by-pixel comparison. This can be an disadvantage (Chrome updates to a newer version might require to change baseline due to font rendering differences * Typescript Integration might be time-consuming |

## Conclusion

# Conclusions