

**Assignments 1-3**

*Anh Nguyen* (*2301590*)

Software Testing (SOF012AS3AE)

Report

May 27, 2024

**Contents *[Remember to update the table of contents when finalizing]***

[1 Introduction 1](#_Toc133913054)

[2 Assignment 1 – Component Testing (max. 21 points) 2](#_Toc133913055)

[2.1 Specifications (No points) 2](#_Toc133913056)

[2.2 Test planning (max. 4 points) 2](#_Toc133913057)

[2.3 Test case design (max. 7 points) 2](#_Toc133913058)

[2.4 Test case implementation (max. 6 points) 3](#_Toc133913059)

[2.5 Test result analysis (max. 4 points) 3](#_Toc133913060)

[3 Assignment 2 – End to End Testing (max. 21 points) 4](#_Toc133913061)

[3.1 Requirements (No points) 4](#_Toc133913062)

[Use case diagram 4](#_Toc133913063)

[3.2 Test planning (max. 4 points) 5](#_Toc133913064)

[3.3 Test case design (max. 7 points) 5](#_Toc133913065)

[3.4 Test case implementation (max. 6 points) 6](#_Toc133913066)

[3.5 Test result analysis (max. 4 points) 6](#_Toc133913067)

[4 Assignment 3 – Exploratory Testing (max. 12 points) 7](#_Toc133913068)

[4.1 Background (No points) 7](#_Toc133913069)

[4.2 Exploratory Test 1 (max. 6 points) 7](#_Toc133913070)

[4.3 Exploratory Test 2 (max. 6 points) 7](#_Toc133913071)

# Introduction

*[In this document, there are instructions for Assignments 1-3 and templates for your answers where you can add your answers. The template texts are in square brackets and in italics font like this text. Remove the template texts and / or replace them by your own information and answers. When starting, remember to add your name, student number (optional) and date on the cover page and when finalizing, remember to update the table of contents.]*

This report is documentation of Assignments 1-3. The assignments are a part of a Software Testing implementation in Haaga-Helia University of Applied Sciences.

*[If you want to add something to the introduction, for example, if you want have notes about this work for yourself for later, you can do so here. This is not required, so can just remove this text if there is nothing you want to add here.]*

The report is has in three sections in addition to this Introduction. Each of the three sections contains one assignment.

# Assignment 1 – Component Testing (max. 21 points)

In Assignment 1, you will do component level testing using Jest for the Todolist web application used in the implementation. You are given specifications (2.1) and you will plan your testing (2.2), design the test cases (2.3), implement them (2.4) and analyze the results (2.5).

## Specifications (No points)

The following specifications have been defined for the functions **addTodo** and **removeTodo** in the file **TodoApp.tsx**:

The method removes the **ToDo** from the todolist. ied to be removed, the contents of the Todolist is not changed and false is returned.

The method adds the **ToDo** to the todolist. If an item with same description and date is already in the current todolist is tried to be added, the contents of the todolist is not changed and a warning message is shown.

## Test planning (max. 4 points)

You are to test that 1) the method **removeTodo** works correctly when an item that is not in the current Todolist is tried to be removed and that 2) the method **addTodo** works correctly when an item that is already in the current Todolist is tried to be added. Based on the specifications in 2.1, plan the testing by explaining how will you do the testing as component level testing - how many test cases will you need and what will they do.

**Test plan**

***Test Suite Description:*** The test suite is named "TodoApp removeTodo functionality", indicating its purpose to test the removal functionality within the TodoApp component.

**a. Test Case 1: Removing an Item Correctly:**

**- Description:** This test ensures that an item can be correctly removed from the todo list.

**- Steps:** Renders the TodoApp component 🡪 Adds two todos to the list using fireEvent 🡪 Verifies both todos are present 🡪 Attempts to remove the first todo 🡪 Checks if the first todo is removed while the second remains.

**- Assertions:** The first todo should no longer be in the document and the second todo should still be present.

**b. Test Case 2: Handling Non-Existing Item Removal:**

**- Description:** This test ensures the application handles the attempt to remove a non-existent item gracefully.

**- Steps:** Renders the TodoApp component 🡪 Adds a todo to the list 🡪 Attempts to remove a non-existent second todo.

**- Assertions:** The original todo should remain in the list and the removal function should return false to indicate the non-existence of the item.

I added one helper function called removeTodoFromList to remove a todo from the list based on its description. **Steps:** Retrieves all todos from the screen 🡪 Finds the index of the todo with the provided description 🡪 If the todo is found, removes it and returns true, else returns false.

## Test case design (max. 7 points)

Design the testing you planned in 2.2 in more detail by filling out the following test case design (detailed test plan).

**Test case design**

**Test targets:** **Notes:**

*[Add information.] [Add information if applicable.]*

**Coverage goal:**

100 % statement coverage

**Test cases:** *[Fill the table. Add rows as necessary.]*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Id | Description | Target(s) | Precondition | Input(s) | Expected outcome (postcondition) |
|  |  |  |  |  |  |

## Test case implementation (max. 6 points)

Implement the test cases you designed in 2.3 using Jest. Create a new test file (such as

**TodoListAssignment1**) to folder **src/test** and implement the test cases there. Add the source code of the test cases here.

**Implemented test cases**

*[Insert the source code of your class here as text or as a screenshot.]*

## Test result analysis (max. 4 points)

Run the test cases you implemented in 2.4 and analyze the results. Find out the answers to following questions:

1. For each test case, did it pass or fail?
2. What is the statement coverage of the test cases?
3. Does the coverage of the test cases meet the goal?

Explain the results of the analysis (answers to the questions) shortly and add screenshots of test report showing the pass/fail results and coverage report showing the coverage.

**Test results**

*[Add here your answers as text and the screenshots showing the results and coverage.]*

# Assignment 2 – End to End Testing (max. 21 points)

In Assignment 2, you will do end to end testing using Robot Framework for the Todolist web application used in the implementation. You are given requirements (3.1) and you will plan your testing (3.2), design the test cases (3.3), implement them (3.4) and analyze the results (3.5).

## Requirements (No points)

The use case diagram and use case description in Figure 1 have been defined based on a requirement “**The Actor shall be able to add a new item to the ToDo-list**.” This requirement is implemented in the Todolist app currently under testing. In addition, the user story “**As Actor, I want to see the number of items in the ToDo-list, because I want to have an idea on how much tasks I have to complete**.” has been added to the product backlog and is to be implemented in the next release. For this, a draft of the user interface for the next release has been designed as seen in Figure 2.

Use case diagram

|  |  |
| --- | --- |
| Precondition: | ToDo-list is populated with Act, Play and Imitate |
| Postcondition: | ToDo-list is populated with Act, Play and Imitate and the added ToDo |
| Normal flow: | 1. Actor opens the application 2. The Application shows the ToDo-list with Act, Play and Imitate in it 3. Actor types the description of the new item and selects Add 4. The Application adds the item to the ToDo-list |
| Alternative flows and Exceptions: | 4a) The ToDo is already in the list. ToDo is not added and the Application shows warning message “The entry is identical with an existing todo. Do you want to keep it?” |

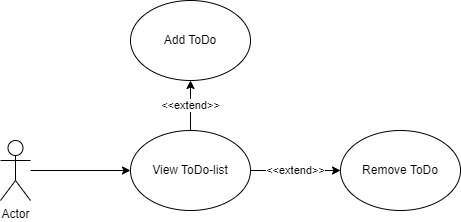


Figure 1. Use case diagram

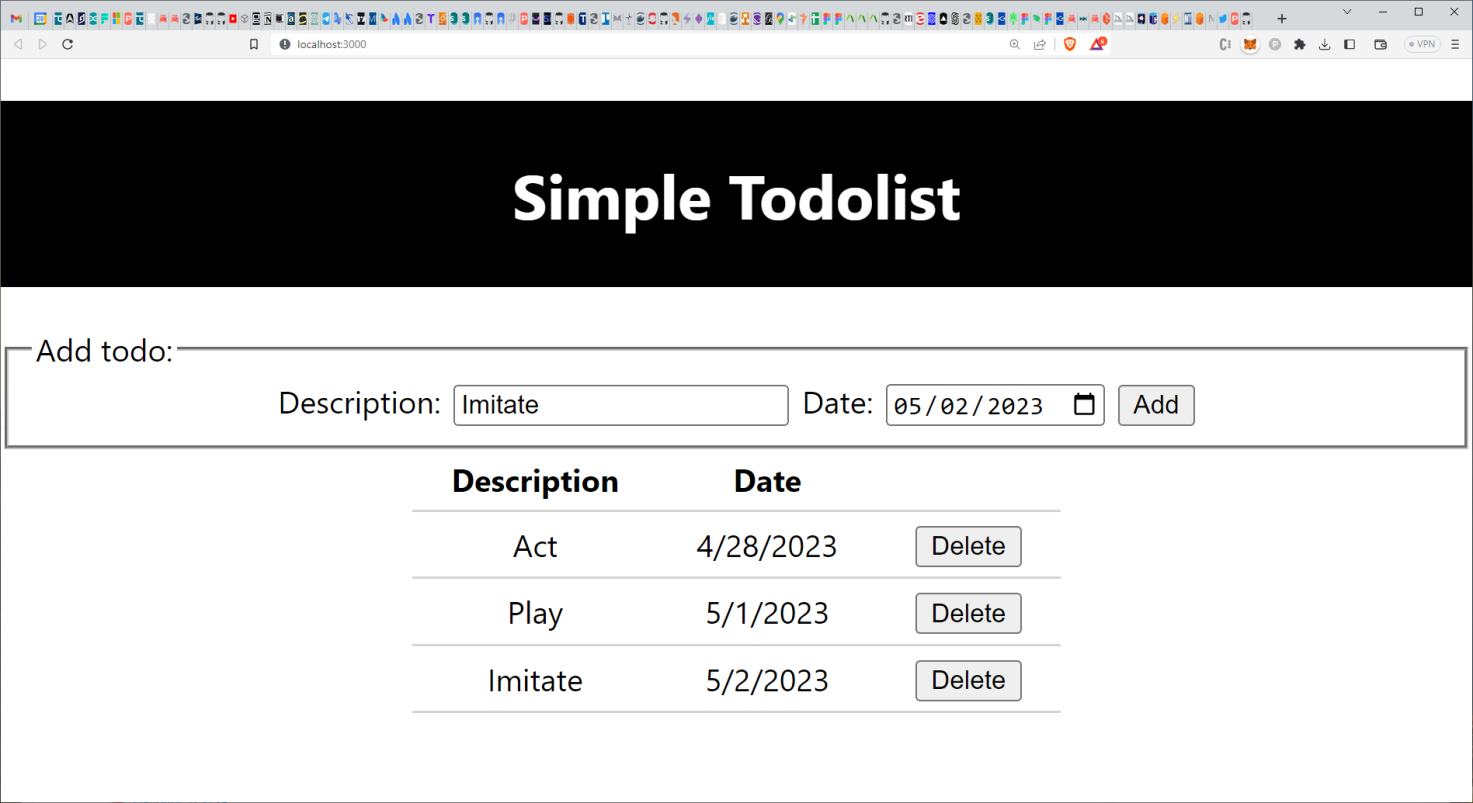


Figure 2. Draft of the user interface for the next release of the Todolist web application

## Test planning (max. 4 points)

You are to test that 1) adding a new item works as required. Also, you are to 2) plan testing for the new requirement of showing the number of item in the Todolist. Based on the information in 3.1, plan the testing by explaining how will you do the testing as end to end level testing - how many test cases will you need and what will they do.

**Test plan**

*[Add here (around paragraph or two of) text explaining your overall test plan in your own words.]*

## Test case design (max. 7 points)

Design the testing you planned in 3.2 in more detail by adding and filling out one test case design form (detailed test plan) for each of the test cases you planned.

**Test case design forms**

*[Fill the test case design form below for each of the text cases. Add forms as necessary.]*

**Form 1**

**Test case:** **Precondition:**

*[Add information.] [Add information.]*

**Target:** **Postcondition:**

*[Add information.]* *[Add information.]*

*[Fill the steps of the test case in the table. Add rows as necessary.]*

|  |  |  |  |
| --- | --- | --- | --- |
| # | Step | Input(s) | Expected outcome |
|  |  |  |  |

## Test case implementation (max. 6 points)

Implement the test cases you designed in 3.3 using Robot Framework. Create a new test script in a file (such as TodoListAssignment2.robot) and implement the test cases there. Add the source code of the script containing the test cases here.

Hint: You can implement the following test cases to add “Imitate” to the input field.

Add Duplicate Items To Todolist And Keep Changes

Add Duplicate Items To Todolist And Decline Changes

**Implemented test cases**

[Insert the source code of your script here as text or as a screenshot.]

## Test result analysis (max. 4 points)

Run the test cases you implemented in 3.4 and analyze the results. Find out the answers to following questions:

1) For each test case, did it pass or fail?

2) What is the statement coverage of the test cases?

3) Does the coverage of the test cases meet the goal?

Explain the results of the analysis (answers to the questions) shortly and add screenshot of test report showing the pass/fail results.

**Test results**

*[Add here your answers as text and the screenshot showing the results.]*

# Assignment 3 – Exploratory Testing (max. 12 points)

In Assignment 3, you will do manual exploratory testing for the Todolist web application used in the implementation. You are given background (4.1) and you will do two exploratory tests (4.2 and 4.3).

## Background (No points)

The Todolist is a web application, so there are, among other, the following two relevant thigs to test: 1) concurrent users (session handling) and 2) cross-site scripting (XSS). For 1), each of the concurrent (simultaneous) users (or browser sessions) should have their own Todolist, so it should be tested that concurrent users do not impact each other’s Todolists when adding and removing items. For 2), the inputs given to the application should be filtered, so that, for example, inputting a script such as

**%3Cscript%3Ealert(%27hello%27)%3C/script%3E**

does not execute and open an alert window in a browser.

## Exploratory Test 1 (max. 6 points)

Concurrent users can be emulated by using, for example, several windows of the same browser or different browsers at the same time. Based on the background given in 4.1, explain how you can test concurrent users (session handling) of the Todolist web application. Try it out and report your findings - what happened and does it look like the application works correctly with concurrent users (meaning that the session handling works)? What other performance test cases can you create based on course material? Create similar list test cases than in chapter 2 and 3.

*[Add here (around paragraph or two of) text explaining your approach and findings in your own words. You can use screenshot(s) to illustrate the testing and results.]*

## Exploratory Test 2 (max. 6 points)

Based on the background given in 4.1, explain how you can test if the Todolist web application is vulnerable to cross-site scripting. Try it out and report your findings - what happened and does it look like the cross-site scripting could work? What other security test cases can you create based on course material? Create similar list test cases than in chapter 2 and 3.

*[Add here (around paragraph or two of) text explaining your approach and findings in your own words. You can use screenshot(s) to illustrate the testing and results.]*