Assignments 1-3

Victor Cherkasov

Software Testing (SOF012AS3AE)

Report

5.4.2024

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

1 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.

2 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).

2.1 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.

2.2 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

1) removeTodo function:

Test Case 1: Firstly I'll test that the method delete the correct ToDo from list when item have to be in list

Input: a particular ToDo item that exists in the todo list.

Expected Output: a particular ToDo item is deleted from the todo list, and the function must return true as expected.

Test Case 2: Secondly, I'll test that the method does not delete any item from the todo list when the item is missing.

Input: A ToDo item which is not in todo list

Expected Output: The todo list remains unchanged, and the function must return false.

2) addTodo function:

Test Case 1: I must ensure method adds the ToDo to the todo list when the item is not in list.

Input: A ToDo item which is not in the todo list.

Expected Output: The ToDo item is added to the list without any warning messages.

Test Case 2: I must ensure that method detect any duplicate of ToDo item and does not add it to the todo list, in addition shows a warning message about duplicate problem.

Input: a ToDo item that is already in the todo list

Expected Output: The todo list is not changed, and a warning message shows with message that ToDo item exists

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:

Detailed test plan for testing scenarios in previous section

Notes:

Coverage goal:

100 % statement coverage

Test cases:

Id	Description	Target(s)	Precondition	Input(s)	Expected outcome (postcondition)
1	Verifying that removeTodo function removes Todo item correctly from todo list	Focusing on removeTodo function in TodoApp.tsx	ToDo items exist in todo list	Certain ToDo item that is present in ToDo list	The certain ToDo item is removed from todo list. Function returns true
2	Verifying that removeTodo function returns false when function trying to remove non-exists ToDo item	Focusing on removeTodo function in TodoApp.tsx	ToDo items exist in todo list	Certain ToDo item that does not in the todo list	The todo list staying unchanged. Function returns false
3	Verifying that addTodo function correctly adds a new ToDo item to the todo list when item is not exists in the list	Focusing on addTodo function in TodoApp.tsx	Empty list or with existing ToDo items	A certain ToDo item which is not in the todo list	The certain ToDo item is added correctly to the todo list without any warning messages
4	Verifying that that addTodo function does not add a duplicate ToDo item to the list and shows warning message	Focusing on addTodo function in TodoApp.tsx	List with existing ToDo items	A ToDo item which is present in todo list	The todo list remains the same. A warning message alert with telling that item exists in the list

2.3 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

src > test > JS TodoListAssignment1.test.js > describe('addTodo function test') callback > test('add duplicate ToDo item to the

```
import { fireEvent, render, screen } from '@testing-library/react';
import '@testing-library/jest-dom/extend-expect';
import TodoInput from '../components/TodoInput';
import TodoApp from '../components/TodoApp';
import React from 'react';


describe('removeTodo function test', () => {
    test('remove ToDo item from list (Test Case 1)', () => {
        render(<TodoApp remove={(remove) => console.log("Remove", remove)} />)
        const deleteButtons = screen.getAllByRole('button', { name: /delete/i })
        const deleteButton = deleteButtons[0]
        fireEvent.click(deleteButton)
        expect(deleteButton).not.toBeInTheDocument()
    })
    test('does not delete any item from the todo list when the item is missing (Test Case 2)', () => {
        render(<TodoApp />)
        const deleteButton = screen.getByRole('button', { name: /delete/i })
        fireEvent.click(deleteButton);

    })

})

```

```
describe('addTodo function test', () => {

    test('add ToDo item to the todo list without warning message', () => {
        render(<TodoInput onValue={(todo) => console.log("Todo:", todo)} />);
        const inputElement = screen.getByRole('textbox', { name: /desc/i });
        const dateInputElement = screen.getByPlaceholderText('Enter date')
        const addButton = screen.getByRole('button', { name: /add/i });

        fireEvent.change(inputElement, { target: { value: 'Hello' } })
        fireEvent.change(dateInputElement, { target: { value: '2024-04-15' } });
        fireEvent.click(addButton);


        expect(inputElement.value).toBe('Hello')
        expect(dateInputElement.value).toBe('2024-04-15');
        expect(addButton).toBeEnabled()
    });
    test('add duplicate ToDo item to the todo list with warning message', () => {
        const modalRef = { current: { showModal: jest.fn() } }

        render(<TodoApp modal={modalRef} />)

        const inputElement = screen.getByRole('textbox', { name: /desc/i });
        const dateInputElement = screen.getByPlaceholderText('Enter date')
        const addButton = screen.getByRole('button', { name: /add/i });

        fireEvent.change(inputElement, { target: { value: 'Default Todo 1' } })
        fireEvent.change(dateInputElement, { target: { value: '2024-04-15' } });
        fireEvent.click(addButton);

        fireEvent.change(inputElement, { target: { value: 'Default Todo 1' } });
        fireEvent.change(dateInputElement, { target: { value: '2024-04-15' } });
        fireEvent.click(addButton);
        expect(modalRef.current.showModal).toHaveBeenCalled();
    })
})
```

2.4 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:

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

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

Three test cases passed and last one fails (test case 2 in addTodo function test).

removeTodo function test:

Test case 1:

```
 PASS  src/test/TodoListAssignment1.test.js
  removeTodo function test
    ✓ remove ToDo item from list (Test Case 1) (81 ms)
    ○ skipped does not delete any item from the todo list when the item is missing (Test Case 2)
  addTodo function test
    ○ skipped add ToDo item to the todo list without warning message
    ○ skipped add duplicate ToDo item to the todo list with warning message

Test Suites: 1 passed, 1 total
Tests:       3 skipped, 1 passed, 4 total
Snapshots:   0 total
Time:        0.832 s, estimated 1 s
Ran all test suites matching /src\/test\/TodoListAssignment1.test.js/i.

Active Filters: filename /src/test/TodoListAssignment1.test.js/
 › Press c to clear filters.

Watch Usage
 › Press a to run all tests.
 › Press f to run only failed tests.
 › Press o to only run tests related to changed files.
 › Press q to quit watch mode.
 › Press p to filter by a filename regex pattern.
 › Press t to filter by a test name regex pattern.
 › Press Enter to trigger a test run.
```

Test case 2:

```
 PASS  src/test/TodoListAssignment1.test.js
  removeTodo function test
    ✓ does not delete any item from the todo list when the item is missing (Test Case 2) (41 ms)
    ○ skipped remove ToDo item from list (Test Case 1)
  addTodo function test
    ○ skipped add ToDo item to the todo list without warning message
    ○ skipped add duplicate ToDo item to the todo list with warning message

Test Suites: 1 passed, 1 total
Tests:       3 skipped, 1 passed, 4 total
Snapshots:   0 total
Time:        0.393 s, estimated 1 s
Ran all test suites matching /src\/test\/TodoListAssignment1.test.js/i.

Watch Usage: Press w to show more.
```

addTodo function test:

Test case 1:

```
console.log
  Todo: { date: '2024-04-15', desc: 'Hello' }

    at onValue (src/test/TodoListAssignment1.test.js:28:54)

PASS  src/test/TodoListAssignment1.test.js
  removeTodo function test
    ○ skipped remove ToDo item from list (Test Case 1)
    ○ skipped does not delete any item from the todo list when the item is missing (Test Case 2)
  addTodo function test
    ✓ add ToDo item to the todo list without warning message (62 ms)
    ○ skipped add duplicate ToDo item to the todo list with warning message

Test Suites: 1 passed, 1 total
Tests:       3 skipped, 1 passed, 4 total
Snapshots:   0 total
Time:        0.633 s, estimated 1 s
Ran all test suites matching /src\/test\/TodoListAssignment1.test.js/i.

Active Filters: filename /src/test/TodoListAssignment1.test.js/
 › Press c to clear filters.

Watch Usage
 › Press a to run all tests.
 › Press f to run only failed tests.
 › Press o to only run tests related to changed files.
 › Press q to quit watch mode.
 › Press p to filter by a filename regex pattern.
 › Press t to filter by a test name regex pattern.
 › Press Enter to trigger a test run.
```

Test case 2:

```
● addTodo function test › add duplicate ToDo item to the todo list with warning message

  expect(jest.fn()).toHaveBeenCalled()

  Expected number of calls: >= 1
  Received number of calls:    0

    56 |         fireEvent.change(dateInputElement, { target: { value: '2024-04-15' } });
    57 |         fireEvent.click(addButton);
  > 58 |         expect(modalRef.current.showModal).toHaveBeenCalled();
       |                                            ^
    59 |     })
    60 | })
    61 |

    at Object.<anonymous> (src/test/TodoListAssignment1.test.js:58:44)
    at TestScheduler.scheduleTests (node_modules/react-scripts/node_modules/@jest/core/build/T
    at runJest (node_modules/react-scripts/node_modules/@jest/core/build/runJest.js:404:19)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 3 skipped, 4 total
Snapshots:   0 total
Time:        0.878 s, estimated 1 s
Ran all test suites matching /src\/test\/TodoListAssignment1.test.js/i.

Active Filters: filename /src/test/TodoListAssignment1.test.js/
```

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

I used formula to calculate statement coverage: Statement coverage = (Number of executed statements / Total number of statements in source code) * 100 (https://www.geeksforgeeks.org/statement-coverage-testing/)

(6 / 8) * 100 = 75 %

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

Unfortunately not last test case ran into fail with warning screen detecting. Other three test cases ran without errors and test web application as expected in test design table.

3 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).

3.1 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

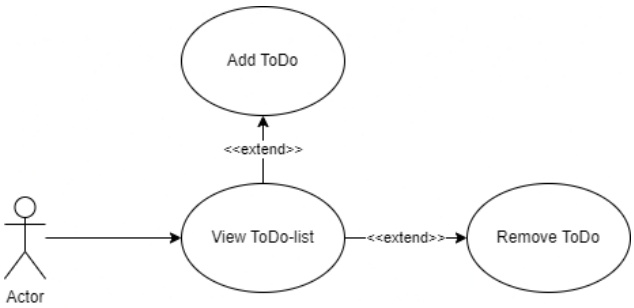


Figure 1. Use case diagram

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

3.2 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

Test Case 1: Ensure successful adding to the ToDo-list.

Description: First test case must ensure that new ToDo item can be added to the todo list.

Steps:

1. Launch the todolist application

2. Enter a new item in the Description and Date fields.

3. Click on the “Add” button.

4. Make sure that the new todo item is displayed in the todo list

5. Verify that the item count increased by one in todo list.

Test Case 2: Ensure numbers of items in todo list displayed correctly.

Description: Second test case check the numbers of items in the todo list without troubles and with correct amount

Steps:

1. Launch the todolist application.

2. Check count of items is displayed on the user interface.

3. Add 1-2 items to the todo list.

4. Make sure that the number of items in the list is the same as the number that is displayed.

3.3 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

Form 1
Test case:
Ensure successful adding to the ToDo-list.

Precondition:
The ToDo-list application launched and accessible.

Target:

must ensure that new ToDo item can be added to the todo list.

Postcondition:
Todo item is added to todo list and amount of items is increased by one.

#	Step	Input(s)	Expected outcome
1	Launch the todolist application	npm start (terminal)	The application is launched without problems.
2	Enter a new item in the Description and Date fields.	New item for description and date	Items added to the fields successfully.
3	Click on the “Add” button		Items added to the list successfully.
4	Make sure that the new todo item is displayed in the todo list		Last todo item is listed in the todo list.
5	Verify that the item count increased by one in todo list.		Amount of todo items is increased by one.

Form 2

Test case:

Ensure numbers of items in todo list displayed correctly.

Precondition:

The ToDo-list application launched and accessible.

Target:

Check the numbers of items in the todo list without troubles and with correct amount.

Postcondition:

Amount of items is correct

#	Step	Input(s)	Expected outcome
1	Launch the todolist application	npm start (terminal)	The application is launched without problems.
2	Check count of items is displayed on the user interface.		Todo items can be founded in user interface
3	Add 1-2 items to the todo list.	Insert information to description and date field	Items added to the list successfully.
4	Make sure that the number of items in the list is the same as the number that is displayed.		List have to be the correct amount of the items.

3.4 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 the following command to add “Imitate” to the input field.

```
Input Text          new-item-title      Imitate
```

Implemented test cases

src > TodoListAssignment2.robot

```
*** Settings ***
Library     SeleniumLibrary
Library     OperatingSystem

*** Variables ***
${BROWSER}          Chrome
${URL}              http://localhost:3000/
${NewItemDisc_Field}    id:todo_description_input
${item_description1}     New Test Item
${item_description2}     New Test Item 2
${NewItemDate_Field}    id:todo_date_input
${Date_To_Enter}        20/04/2024
${Add_Button}       xpath=//button[contains(text(),'Add')]
${Item_Count}       id:item-count

*** Test Cases ***
Verify Successful Addition of a New Item
    Open Browser    ${URL}    ${BROWSER}

    Click Element   ${NewItemDisc_Field}
    Input Text  ${NewItemDisc_Field}    ${item_description1}

    ${day}    ${month}    ${year}=    Split Date    ${Date_To_Enter}
    Input Text    ${NewItemDate_Field}    ${day}.${month}.${year}

    Click Button    ${Add_Button}

   # Verify the new item is displayed in the todo list
    ${new_item_xpath}=    Set Variable    //tr[td[@id="todo_list"][contains(text(),"${item_description1}")]]/td[contains(text(),"${Date_To_Enter}")]
    Wait Until Element Is Visible    ${new_item_xpath}
    Page Should Contain Element    ${new_item_xpath}

    Close Browser
```

src > TodoListAssignment2.robot

```
*** Test Cases ***
Ensure numbers of items in todo list displayed correctly
    Open Browser    ${URL}    ${BROWSER}

    ${items}        Get WebElements     xpath=//tr[td[@id="todo_list"]]
    ${initial_count}        Get Length      ${items}
    Log     Initial count of items: ${initial_count}

    Click Element   ${NewItemDisc_Field}
    Input Text  ${NewItemDisc_Field}    ${item_description1}

    ${day}    ${month}    ${year}=    Split Date    ${Date_To_Enter}
    Input Text    ${NewItemDate_Field}    ${day}.${month}.${year}

    Click Button    ${Add_Button}

    Click Element   ${NewItemDisc_Field}
    Input Text  ${NewItemDisc_Field}    ${item_description2}

    ${day}    ${month}    ${year}=    Split Date    ${Date_To_Enter}
    Input Text    ${NewItemDate_Field}    ${day}.${month}.${year}

    Click Button    ${Add_Button}

    ${items_after_adding}    Get WebElements    xpath=//tr[td[@id="todo_list"]]
    ${current_count}    Get Length    ${items_after_adding}
    Should Be Equal As Integers    ${current_count}    ${initial_count + 2}

    Close Browser

*** Keywords ***
Split Date
    [Arguments]    ${date}
```

```
src > TodoListAssignment2.robot
*** Keywords ***
Split Date
    [Arguments]    ${date}
    ${day}    ${month}    ${year}=    Evaluate    '${date}'.split('/')
    [Return]    ${day}    ${month}    ${year}
```

3.5 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:

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

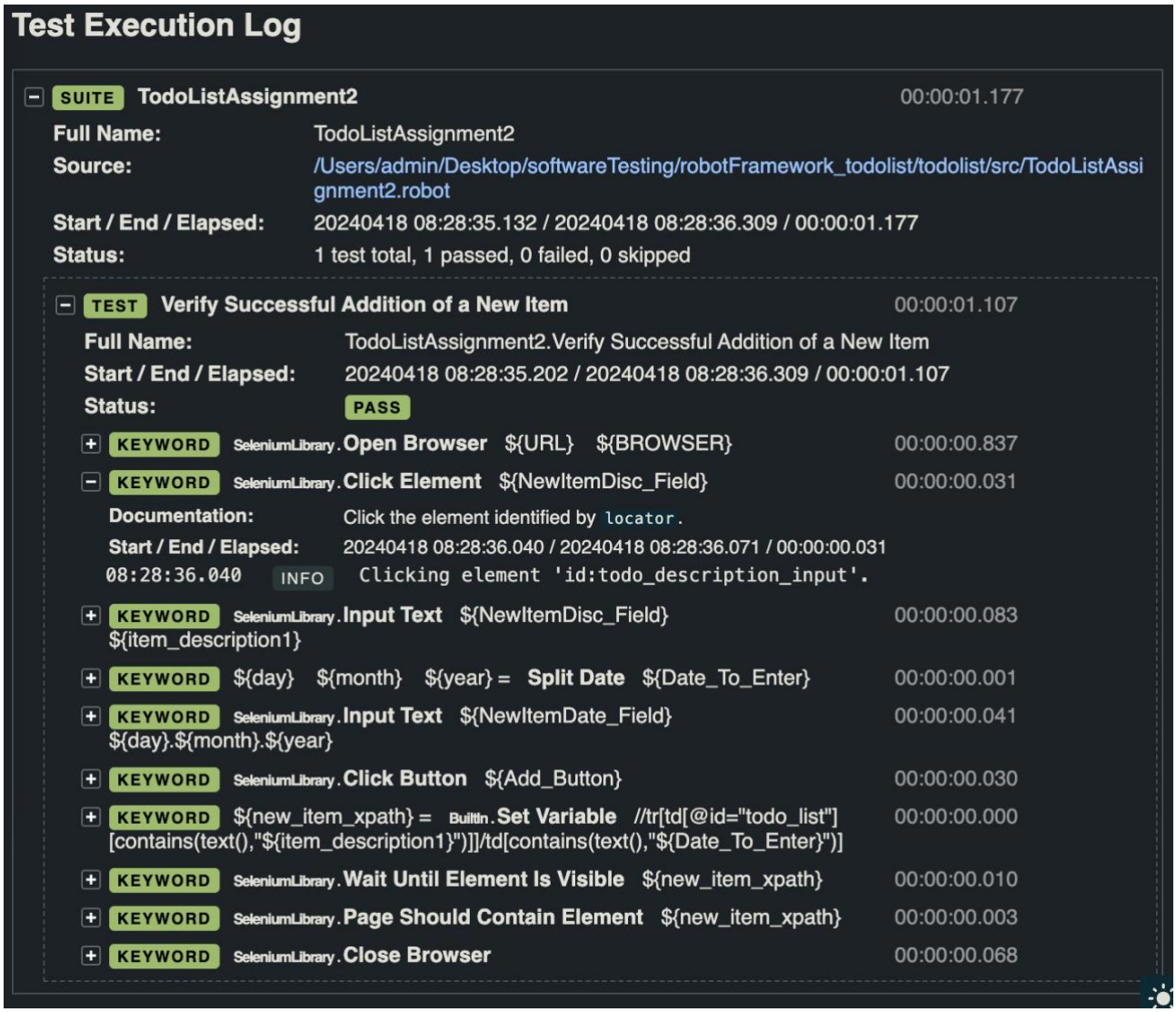
Test results

Both test cases passed.

Test case 1:

First test case verifies successful addition of a new todo item.

Firstly, test case opens browser (Chrome) on port :3000. Secondly find input element by id todo_description_input and insert short text to the field. After that test case has to split date (20/04/2024) to day, month and year by "/" (otherwise whole date inserts only to yyyy) and insert to the input element with id todo_date_input. Once description field and data field filled test case has to find and press Add button by text path xpath=//button[contains(text(),'Add')]. Lastly setting variable new_item_xpath that it contains description and date on website with //tr[td[@id="todo_list"][contains(text(),"${item_description1}")]]/td[contains(text(),"${Date_To_Enter}")] and after that wait until element is visible and contain variable new_item_xpath.



Test case 2:

Second case ensures numbers of items in todo list displayed correctly.

As precondition I added one todo item for positive count (I mean count starts from 1 not 0).

This test case starts same as first test case with opening same browser on the same port. Then test case identify todo list by path (xpath=//tr[td[@id="todo_list"]]) and get length which is 1 currently.

```
KEYWORD  ${initial_count} =  BuiltIn.Get Length  ${items}
Documentation:        Returns and logs the length of the given item as an integer.
Start / End / Elapsed:  20240418 08:46:05.315 / 20240418 08:46:05.315 / 00:00:00.000
08:46:05.315  INFO  Length is 1.
08:46:05.315  INFO  ${initial_count} = 1
```

When test case got the length of the list it will add some new todo items with same technic which is provided in test case 1. After that test case get webelements from xpath=//tr[td[@id="todo_list"]] and gives information to items_after_adding variable. Then test case gives items_after_adding length to current_count variable and tests that variables should be equal with initial_count + 2 (which got length at first of the test code)

```
Test Execution Log
SUITE TodoListAssignment2                                          00:00:01.313
  Full Name:          TodoListAssignment2
  Source:             /Users/admin/Desktop/softwareTesting/robotFramework_todolist/todolist/src/TodoListAssignment2.robot
  Start / End / Elapsed: 20240418 08:46:04.393 / 20240418 08:46:05.706 / 00:00:01.313
  Status:             1 test total, 1 passed, 0 failed, 0 skipped
  TEST Ensure numbers of items in todo list displayed correctly       00:00:01.256
    Full Name:          TodoListAssignment2.Ensure numbers of items in todo list displayed correctly
    Start / End / Elapsed: 20240418 08:46:04.450 / 20240418 08:46:05.706 / 00:00:01.256
    Status:             PASS
    KEYWORD SeleniumLibrary.Open Browser  ${URL}  ${BROWSER}          00:00:00.856
    KEYWORD ${items} = SeleniumLibrary.Get WebElements                00:00:00.008
      xpath=//tr[td[@id="todo_list"]]
      Documentation:      Returns a list of WebElement objects matching the locator.
      Start / End / Elapsed: 20240418 08:46:05.307 / 20240418 08:46:05.315 / 00:00:00.008
      08:46:05.314  INFO  ${items} = [<selenium.webdriver.remote.webelement.WebElement
                          (session="ab122e04d7697350c1fb5d682f096a54",
                          element="f.E3B72A11FB10472FA9EB467CA227BC92.d.46DE82F2BF0C47720023A5E28A53E32F.e.4")>]
    KEYWORD ${initial_count} = BuiltIn.Get Length  ${items}           00:00:00.000
      Documentation:      Returns and logs the length of the given item as an integer.
      Start / End / Elapsed: 20240418 08:46:05.315 / 20240418 08:46:05.315 / 00:00:00.000
      08:46:05.315  INFO  Length is 1.
      08:46:05.315  INFO  ${initial_count} = 1
    KEYWORD BuiltIn.Log  Initial count of items: ${initial_count}     00:00:00.000
      Documentation:      Logs the given message with the given level.
      Start / End / Elapsed: 20240418 08:46:05.315 / 20240418 08:46:05.315 / 00:00:00.000
      08:46:05.315  INFO  Initial count of items: 1
    KEYWORD SeleniumLibrary.Click Element  ${NewItemDisc_Field}       00:00:00.031
    KEYWORD SeleniumLibrary.Input Text  ${NewItemDisc_Field}          00:00:00.090
      ${item_description1}
    KEYWORD ${day}  ${month}  ${year} = Split Date  ${Date_To_Enter}  00:00:00.001
    KEYWORD SeleniumLibrary.Input Text  ${NewItemDate_Field}          00:00:00.041
      ${day}.${month}.${year}
```

```
KEYWORD  SeleniumLibrary.Click Button  ${Add_Button}                                   00:00:00.032
KEYWORD  SeleniumLibrary.Click Element  ${NewItemDisc_Field}                           00:00:00.022
KEYWORD  SeleniumLibrary.Input Text  ${NewItemDisc_Field}                              00:00:00.040
         ${item_description2}
KEYWORD  ${day}  ${month}  ${year} =  Split Date  ${Date_To_Enter}                     00:00:00.001
KEYWORD  SeleniumLibrary.Input Text  ${NewItemDate_Field}                              00:00:00.034
         ${day}.${month}.${year}
KEYWORD  SeleniumLibrary.Click Button  ${Add_Button}                                   00:00:00.024
KEYWORD  ${items_after_adding} =  SeleniumLibrary.Get WebElements                      00:00:00.004
         xpath=//tr[td[@id="todo_list"]]
KEYWORD  ${current_count} =  BuiltIn.Get Length  ${items_after_adding}                 00:00:00.000
KEYWORD  BuiltIn.Should Be Equal As Integers  ${current_count}                         00:00:00.000
         ${initial_count + 2}
KEYWORD  SeleniumLibrary.Close Browser                                                 00:00:00.068
```

4 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).

4.1 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

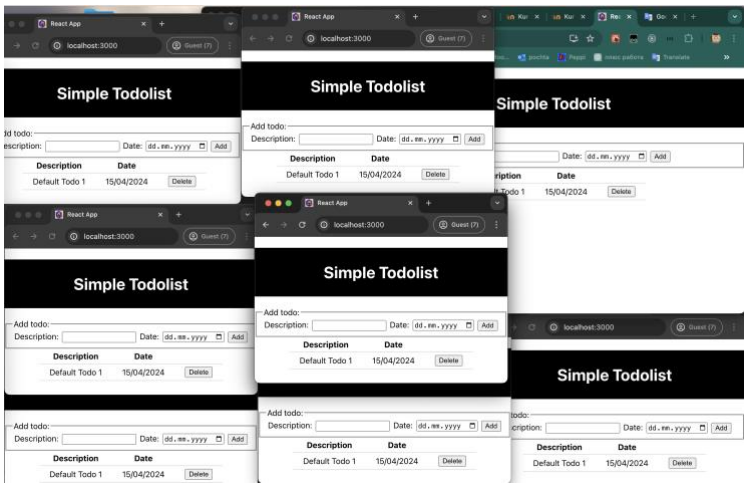
`%3Cscript%3Ealert(%27hello%27)%3C/script%3E`

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

4.2 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)?

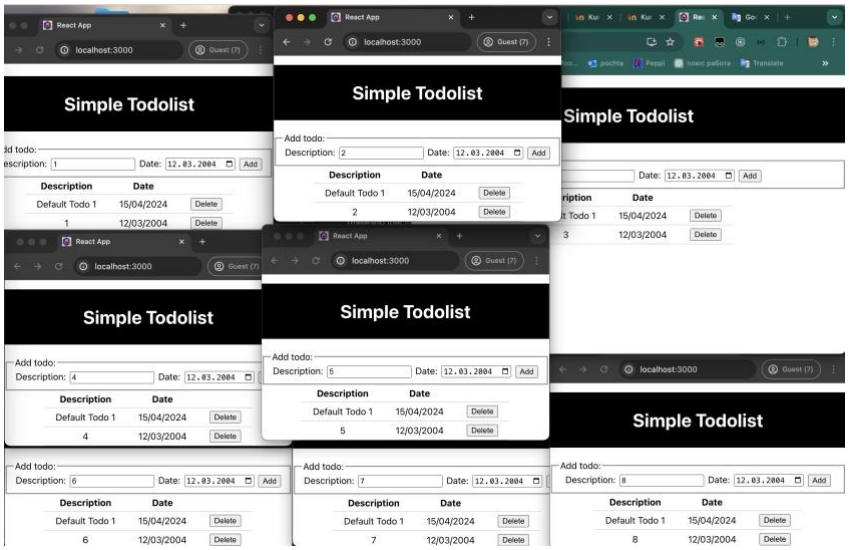
For begging I launched the web application and opened eight windows of the same browser(chrome), seven of them are guest users and last one is my own account with green navbar.



After begging I inserts basic data to each window. For description field I used numbers from 1 to 8 and as a date I used same date for every window (12.03.2024).

Summary about add function on different windows in same time:

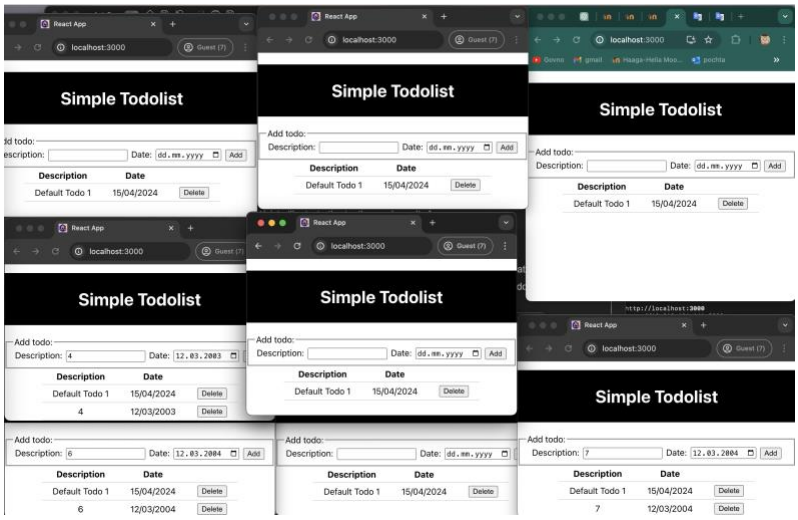
As expected, different sessions don’t interfere to each other and works by themselves when users are adding todo items.



After add method also I decided to test remove functions. So, I started to remove by one todo item and verify that no errors comes from terminal and to the other session windows.

Summary about remove function on different windows in same time:

As expected, different sessions don’t interfere to each other and works by themselves when users are removing todo items.



4.3 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?

[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.]