# Testování softwaru (KI/TSW)

Pavel Beránek

15 Dubna, 2025

## 10. Akceptační testování

### 9.1. Robot framework

Robot framework je open-source nástroj, který pro testera představuje jednotné testovací centrum, do kterého může vložit různorodé testy přes nástroje jako je Selenium, Appium, Pytest, spouštět je a tvořit reporty. Převážně slouží jako nástroj pro tzv. akceptační testování, kterým se dokazuje klientovi, jak moc je software připraven pro předání. Umožňuje i psát vlastní testy pomocí .robot souborů, snadno vytvořit BDD testy a ovládat funkce OS (Robot Process Automation, RPA).

Vyzkoušíme si nejprve napsat jednoduchý test. Začněte tím, že si nainstalujete Robot framework a jeho modul pro Selenium. Kromě toho si nainstalujeme pytest, na kterém si ukážeme možnost propojení těchto dvou frameworků. Také budeme testovat webovou stránku, kterou si vytvořte. Doporučuji framework Flask pro jednoduchost.

```
pip install robotframework
pip install robotframework-seleniumlibrary
pip install pytest
pip install flask
```

## 9.2. Robot skripty

Pojďme nejprve prozkoumat, jak se píšou nativní skripty v Robot framework.

Nativní skripty vypadají takto:

```
*** Test Cases ***
Simple String Check
    Should Contain    Hello Robot Framework    Robot
```

- Test cases: pomocí 3 hvězdiček se uvozuje sekce. RF obsahuje různorodé sekce jako jsou testovací případy, proměnné, nastavení, klíčová slova. Zde vytváříme testovací případy.
- Simple String Check: název konkrétního testu
- Should Contain: klíčové slovo (funkce) RF, která se volá s dvěma argumenty
  - řetězec, ve kterém se hledá (Hello Robot Framework)
  - řetězec, který se má najít (Robot)

Skripty se spouští příkazem `robot test.robot`, kde `test.robot` je název našeho testovacího skriptu. Po spuštění bychom měli vidět v pracovním adresáři soubory: `report.html`, `log.html`, `output.xml`.

- output.xml: surová data o testech - struktura testů, průběh testování, výsledky, časování, proměnné atd.
- log.html: přehledný log o běhu testu ze souboru output.xml
- report.html: celkové statistiky s odkazy na log.html

Pojďme si ukázat ještě další jednotkové testy:

```
*** Test Cases ***

# 1. Test na porovnání čísel
Check That Five Is Greater Than Three
    Should Be True    ${5} > ${3}

# 2. Test rovnosti čísel
Check That Sum Is Correct
    ${result}=    Evaluate    2 + 2
    Should Be Equal As Numbers    ${result}    4

# 3. Test na text - kontrola obsahu
Text Contains Keyword
    Should Contain    This is a test string    test

# 4. Test na text - přesná rovnost
Check Exact Match
    Should Be Equal    Hello    Hello

# 5. Test na text - není obsaženo
Text Does Not Contain
    Should Not Contain    Hello world    goodbye

# 6. Test na délku seznamu
Check List Length
    ${mylist}=    Create List    apple    banana    orange
    Length Should Be    ${mylist}    3

# 7. Test, zda je hodnota v seznamu
Check Item In List
    ${mylist}=    Create List    apple    banana    orange
    List Should Contain Value    ${mylist}    banana

# 8. Test na přetypování a výpočet
Convert To Int And Multiply
    ${val}=    Convert To Integer    7
    ${result}=    Evaluate    ${val} * 3
    Should Be Equal As Numbers    ${result}    21

# 9. Test logické hodnoty
Boolean Test
    ${val}=    Set Variable    ${True}
    Should Be True    ${val}

# 10. Test s proměnnými
Check Concatenation
    ${first}=    Set Variable    Hello
    ${second}=   Set Variable    World
    ${joined}=   Catenate    SEPARATOR=    ${first}    ${second}
    Should Be Equal    ${joined}    HelloWorld
```

Jeden z testů selže. Podívejte se, jak teď vypadají výstupní soubory.

Testy lze sturkturovat pomocí sekcí tímto způsobem. BuiltIn jsou základní klíčová slova RF.

```
*** Settings ***
Library    BuiltIn

*** Variables ***
${TEXT}    Hello Robot Framework
${LIST}    apple    banana    cherry

*** Test Cases ***
My First Test
    Should Contain    ${TEXT}    Robot
```

Pokud chcete využít složitější struktury, pak musíte nahrát knihovny ze standardní knihovny RF, jako je Collections pro List.

```
*** Settings ***
Library    Collections

*** Test Cases ***
Check Item In List
    ${mylist}=    Create List    apple    banana    orange
    List Should Contain Value    ${mylist}    banana
```

Mezi nejdůležitější knihovny patří:

1. BuiltIn: automaticky načtená základní knihovna s klíčovými slovy jako Should Be Equal, Log, Evaluate, Sleep aj.
2. Collections: pro práci se seznamy a slovníky s klíčovými slovy jako Create List, Append To List, Dictionary Should Contain Key aj.
3. SeleniumLibrary: Nutné nainstalovat, slouží pro testy webových stránek pomocí klíčových slov Open Browser, Click Element, Input Text, Page Should Contain, Capture Page Screenshot aj.
4. OperatingSystem: pro práci s OS a shellem pomocí klíčových slov File Should Exist, Remove File, Run Process aj.
5. Process: pro spouštění externích skriptů pomocí Run Process, Start Process, Wait For Process aj.
6. String: typické operace nad řetězcem jako Replace String, Split String, Should Start With aj.
7. RequestLibrary: pro testování REST API skrze GET, POST, Response Should Contain aj.
8. JSONLibrary: pro testování vnitřku odpovědi v JSON formátu přes Load Json From File, Get Value From JSON, Convert String To JSON aj.
9. Dialog: debugující knihovny pro testy s interaktivním zadáváním vstupů a provádění kroků testů přes Get Value From User, Pause Execution, Execute Manual Step aj.

Zadání:

1. Opravte nefunkční test a ujistěte se, že report dává pozitivní výsledky.
2. Nainstalujte si `robotframework-jsonlibrary`.
3. Spusťte následující testy a pochopte jejich strukturu a logiku úpravou hodnot.

```
*** Test Cases ***
Should Be Equal Example
    ${a}=    Set Variable    42
    ${b}=    Evaluate    6 * 7
    Should Be Equal As Numbers    ${a}    ${b}

*** Settings ***
Library    String

*** Test Cases ***
Replace Substring Example
    ${text}=    Set Variable    Hello student
    ${new}=     Replace String    ${text}    student    teacher
    Should Be Equal    ${new}    Hello teacher

*** Settings ***
Library    Collections

*** Test Cases ***
List Contains Value Example
    ${fruits}=    Create List    apple    banana    cherry
    List Should Contain Value    ${fruits}    banana

*** Settings ***
Library    Dialogs

*** Test Cases ***
Ask User For Name
    ${name}=    Get Value From User    Please enter your name:
    Log    You entered: ${name}

*** Settings ***
Library    JSONLibrary
Library    Collections

*** Test Cases ***
Get Value From JSON Example
    ${json}=         Convert String To Json    {"name": "Alice", "age": 30}
    ${age_list}=     Get Value From Json       ${json}    $.age
    ${age}=          Get From List             ${age_list}    0
    Should Be Equal As Numbers    ${age}    30
```

## 9.3. Pytest v RF

Robot framework má vlastní skriptovací jazyk pro psaní jednotkových testů. Tento jazyk však nedisponuje všemi možnostmi testovacích frameworků zaměřených na jeden typ testů jako je např. Pytest. Pojďme si zavolat z robot skriptu pytest soubor s jednotkovými testy.

```
*** Settings ***
Library    Process

*** Test Cases ***
Run Pytest Test
    ${result}=    Run Process    pytest    test_calc.py    stdout=stdout    stderr=stderr
    Log           ${result.stdout}
    Should Contain    ${result.stdout}    1 passed
```

Osobně nedoporučuji míchat nástroje se "skoro" stejnými možnostmi dohromady, ale pokud k tomu máte dobrý důvod, tak cesta existuje.

Zadání:

1. Napište si jednoduchý kód pro kalkulačku Vámi vybraného souboru (např. `calc.py`).
2. Napište do Vámi vybraného souboru (např. `test_calc.py`) testy pro základní funkce kalkulačky.
3. Upravte můj skript v RF aby testy prošly. Doporučuji podívat se na Should Contain parametry.

## 9.4. Selenium v RF

V Robot framework lze využívat velké množství nástrojů, které je nutné doinstalovat jako modul. Do skriptů se pak importují pod klíčovým slovem library. Následující skript otevře definovanou URL v proměnné v prohlížeči, vyhledá tlačítko jazykem XPath, klikne na něj a zkontroluje, zda se na webu nachází text Odesláno.

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

*** Variables ***
${URL}    http://localhost:5000

*** Test Cases ***
Add New Task
    Open Browser           ${URL}    chrome
    Input Text             name=task    Napsat test
    Click Button           xpath=//input[@type="submit"]
    Page Should Contain    Odesláno
    Close Browser
```

Zadání:

1. Vytvořte si jednoduchou stránku, která testo test splňuje.
2. Napiště další libovolný test.
3. Nahlédněte do reportu.

Řešení:

```
from flask import Flask, request, redirect, render_template_string

app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def index():
    if request.method == "POST":
        task = request.form.get("task")
        return render_template_string("""
            <h1>Odesláno</h1>
            <p>Zadal jsi: {{ task }}</p>
        """, task=task)

    return render_template_string("""
        <h1>TODO Formulář</h1>
        <form method="post">
            <input name="task" placeholder="Zadej úkol">
            <input type="submit" value="Přidat">
        </form>
    """)

if __name__ == "__main__":
    app.run(debug=True)
```

## 9.5. BDD v RF

Skriptovací jazyk Robot frameworku umožňuje psát BDD testy v jazyce Gherkin. Jednotlivé BDD kroky si musíme nadefinovat ručně (nebo neznám lepší řešení, možná existuje knihovna). Příkladem může být následující skript:

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

*** Variables ***
${URL}    http://localhost:5000

*** Test Cases ***
Adding A Task Should Appear In List
    Given  Browser Is Open To Todo Page
    When   I Add Task With Text         Udělat kávu
    Then   I Should See Task In List    Udělat kávu

*** Keywords ***
Given
    [Arguments]    ${step_keyword}    @{args}
    Run Keyword    ${step_keyword}    @{args}

When
    [Arguments]    ${step_keyword}    @{args}
    Run Keyword    ${step_keyword}    @{args}

Then
    [Arguments]    ${step_keyword}    @{args}
    Run Keyword    ${step_keyword}    @{args}

Browser Is Open To Todo Page
    Open Browser    ${URL}    chrome

I Add Task With Text
    [Arguments]     ${text}
    Input Text      name=task    ${text}
    Click Button    xpath=//input[@type="submit"]

I Should See Task In List
    [Arguments]    ${text}
    Page Should Contain    ${text}
    Close Browser
```

Webová stránka splňující takový krok může vypadat takto:

```
from flask import Flask, request, redirect, render_template_string

app = Flask(__name__)
todos = []

@app.route("/", methods=["GET", "POST"])
def index():
    if request.method == "POST":
        task = request.form.get("task")
        if task:
            todos.append(task)
        return redirect("/")

    return render_template_string("""
        <h1>TODO Seznam</h1>
        <form method="post">
            <input name="task" placeholder="Zadej úkol">
            <input type="submit" value="Přidat">
        </form>
        <ul>
        {% for t in todos %}
            <li>{{ t }}</li>
        {% endfor %}
        </ul>
    """, todos=todos)

if __name__ == "__main__":
    app.run(debug=True)
```

Zadání:

1. Přepiště test z předchozího cvičení (9.4.) do BDD.
2. Stránku rozšiřte o jednu funkcionalitu a ověřte si ji novým BDD testem (nejprve kód, pak test).
3. Pak zkuste vymyslet ve své hlavě novou funkcionalitu, napište k ni BDD test a teprv poté dopište do stránky kód (nejprve test, pak kód - přístup iniciovaný testy).

## 9.6. Organizování a reporting v RF

Robot framework umožňuje organizaci testů na několika úrovních: 1. adresáře - organizace do adresářů se objevuje v reportech 2. názvy testů - vhodné názvy a jejich přípony/předpony nám dost pomohou pro snadnou orientaci 3. značky - slouží pro selektivní výběr testů ZDE 4. dokumentace - testy lze dokumentovat pomocí dokumentačních řetězců ZDE

Od RF existuje doporučená projektová struktura ZDE a styl psaní testů ZDE

## Domácí cvičení - Plný reporting

Napište si jednoduchou webovku alespoň o 3 endpointech se vstupním formulářem. Proveďte plné testování a vygenerujte z něj report. Organizujte si vhodně testy.