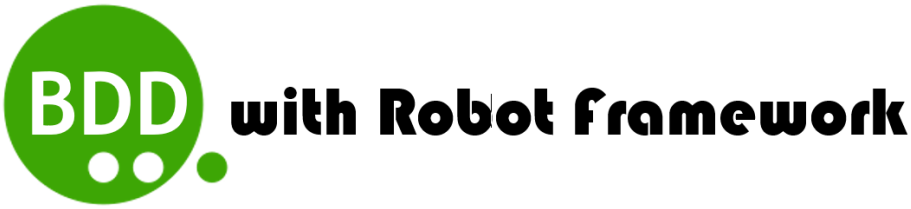

Automation with Robot Framework

Workbook for attendees

Gáspár Nagy

coach, trainer and BDD addict,
creator of SpecFlow

gaspar@specsolutions.eu
http://gasparnagy.com
@gasparnagy

A1: Automate Menu scenario

> Automate scenario “Only active pizza menu items are listed on the menu page” using API automation.

- Open the `features\home.robot` file and check the scenario “Only active pizza menu items are listed on the menu page”.
- Imagine what you would do to verify the scenario manually: Think it over how you would automate the different steps using controller automation.
- As you can see from the IDE color of the steps, the “Given” step has been automated already. You will need no to automate the “When” and “Then” steps using the following steps.
- Copy the name of the steps (without the “When” and “Then” keywords) to the end of the `keywords\geekpizza_keywords.robot` file (in the `*** Keywords ***` section). Replace parameters values with parameters in `${parameter_name}` format (see other keywords as example).
- Automate “The client checks the menu page” keyword by making a GET request to the `/api/menu` endpoint.
- Automate the other keyword by asserting on the count of the “`items`” array of the returned JSON object. The count can be checked using the “`Length Should Be`” keyword.
- Run the test. It should fail: proves that the business rule illustrated by the scenario is not implemented yet!
- Now simulate that the expectation is implemented in the application by clicking on the “*[Complete A1 Tutorial]*” link in the app. The test should pass now.
- *Bonus:* Take a look at the provided automation for the “Given” step. (You can navigate to the step definition from the feature file with *Go to Definition*.) It sets the menu items by invoking the admin endpoint. Would it be possible to simply save these records to the database? *Would that be a good idea? What are the disadvantages of it?*

Hints

- You can check the returned JSON manually as well by navigating to the `/api/menu` endpoint of the application in the browser.
- How to call an API controller method? Check the step definition of the “The client checks the home page” keyword!
- How to work with a result returned by a GET endpoint? Check the keywords of the “When the client checks the home page” and the “Then the home page main message should be: "Welcome to Geek Pizza!"” steps!

A2: AUTOMATE DATATABLE ASSERTION OF THE MENU SCENARIO

Automate the missing "Then" step of the scenario "The pizzas are listed in alphabetically on the menu page".

- Imagine what you would do to verify the scenario manually.
- The "Given" and "When" steps have been defined (automated) already. You only need to work on the "Then" step. Review how the "Given" step is defined, you can see how you can work with tabular structures using the DataTables.py helper class!
- Add a keyword for the "Then" step to the `keywords\geekpizza_keywords.robot` file.
- Take a cell list as argument and convert it to a table in the same way how we do it in the keyword of the "Given" step. Set it to a variable named `${expected_menu_items}`.
- Implement the rest of the keyword using the following code first:

```
1. Set Local Variable  $actual_menu_items
   ${menu_response.json()}[items]
2. ${expected_count} =  Get Length  ${expected_menu_items}
3. Length Should Be  ${actual_menu_items}  ${expected_count}
4. FOR  ${expected_menu_item}  ${actual_menu_item}  IN ZIP
   ${expected_menu_items}  ${actual_menu_items}
5.     #TODO compare name key of ${expected_menu_item} and
   ${actual_menu_item}
6. END
```

- Complete the code with an assertion in the FOR loop body and check if it fails.
- Again, simulate that the expectation is implemented in the application by clicking on the "*[Complete A2 Tutorial]*" link in the app. The test should pass now.
- Now the test pass, but the logic you wrote into the "Then" step keyword is pretty complex. Are you sure that it is bug-free? Test the test by making the scenario wrong and see if you really got an error! Try to change pizza names in the scenario or add/remove pizzas. Why do we need the line 3 that compares the counts? Have you found any case when the scenario is wrong but the scenario passes (false negative)?
- Change the assertion to the following. Why would this be better?

```
1. Dictionary Should Contain Sub Dictionary  ${actual_menu_item}
   ${expected_menu_item}
```

- Make the scenario wrong again and see how new assertion reports the error. Try to add an "ingredients" column to the data table in the feature file and see whether it is checked.

HINTS

- A table can be created from a cell list using the `${table} = Create DataTable @{cell_list}` keyword (implemented by `support\DataTables.py`).
- A row of the table can be retrieved by `${table}[0]`.
- A cell within the row can be retrieved by `${row}[header]`.

A3: SPLIT FEATURE AND KEYWORD ROBOT FILES

Split feature and keyword robot files to ensure a logical structure

- This is a refactoring exercise, so make sure the tests are passing before you would make any changes.
- Move Menu-related scenarios to a new robot file: `menu.robot`. *Do they still pass? Did you have to make any changes in the keywords to make it pass? Why?*
- The first two keywords in `keywords\geekpizza_keywords.robot` are all related to the "admin" interface of the application. Extract them to a separate keyword file: `admin_keywords.robot`. *Where do you have to "register" the new file to make the tests pass again?*
- Split the remaining keywords in `geekpizza_keywords.robot` to `home_keywords.robot` and `menu_keywords.robot` accordingly.

A4: AUTOMATE SCENARIO USING AN ATTEMPT-ACTION PATTERN

Complete the automation of the keywords of the registration.robot scenario in the user_keywords.robot file.

- Check the automation of the "Given the client is logged in" step in the `home.robot` file. The related keyword logs in the user by invoking a POST request.
- Now look at the scenario in the `registration.robot` file. This needs to perform a similar action (make a POST request to perform registration), but the assertion will be in the "Then" step.
- Try to complete the automation of the two registration steps in the `user_keywords.robot` file. The POST request has to go to the `/api/user` endpoint.
- If the scenario passes, try changing the password in the scenario to "`123`", that is too short. Do you get an error? Did you get it from the step you expected? For step details, you can also check the `log.html` in the exercise folder.
- *Hint:* Check the `expected_status` argument of the `POST On Session` keyword in the documentation.

A5: Practice using Data-driven Tests

> Convert the scenarios in the rule "The password should be valid and verified" of registration.robot to a data-driven test.

- There are three scenarios within the rule "The password should be valid and verified" of the `registration.robot` file that check the password validation of the registration process.
- Replace the three scenarios with a single "Scenario Outline" with three examples:
    1. Password was not provided
    2. The re-entered password is different
    3. Password is too short
- In order to create a data-driven test, you need to create a test that uses a template with multiple data sets. See the "Hints" section for an example of the syntax.
- Add an additional first argument (e.g. "`${description}`") to the template to describe the purpose of the individual examples.
- *Bonus:* How could you include the scenario "User name was not provided" as well into your scenario outline. What problems you see with the approach you have chosen?

Hints

- Data-driven test syntax

```
*** Test Cases ***
My data-driven test
    [Template]  My Test Template
    Orange  12
    Apple   8

*** Keywords ***
My Test Template
    [Arguments]  ${fruit}  ${price}
        Some Keyword  ${fruit}
        Some Other Keyword  ${price}
```

C1: Driver for Admin API call (admin_driver)

Extract automation logic of the admin_keywords.robot to use to a driver.

- Create a driver for the admin functions in `admin_keywords.robot`. Name the new driver `admin_driever.robot` and save it to the `drivers` folder. You can use the `user_driver.robot` class as an example.
- Update the keywords in `admin_keywords.robot` to use the driver object.

C2: Driver for GET API call (home_driver)

Extract the automation logic of the home_keywords.robot call to a driver.

- Create a driver for the home page automation in `home_keywords.robot`. Name the new driver `home_driver.robot` and save it to the `drivers` folder.
- *Bonus:* Refactor the remaining keywords that directly call HTTP requests into drivers. (The C2 folder in the `Solutions` folder contain implementations for these: `menu_driever.robot` and `auth_driver.robot`.)