Automation with SpecFlow Tutorial
Workbook for attendees

Gáspár Nagy

coach, trainer and BDD addict,
creator of SpecFlow

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

A1: AUTOMATE MENU SCENARIO WITH CONTROLLER AUTOMATION

> Automate scenario "Only active pizza menu items are listed on the menu page" using controller automation.

- 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 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.

- Generate step definition skeletons for the "When" and "Then" steps and add them to the existing `StepDefinitions\GeekPizzaStepDefinitions.cs` file. (To add step definitions to an existing file, you can use the "*Copy methods to clipboard*" button of the "*Define Steps*" dialog.)

- Implement step definitions to make the test fail: proves that the business rule illustrated by the scenario is not implemented yet! See Hints below, if you need help.

- Now (and only now) we can try to implement the application logic to satisfy the scenario. Check the `MenuController.GetPizzaMenu()` method, and you will see how this could be done.

- *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 (F12)*.) It saves the menu items directly to the database. *Is this a good idea? What are the disadvantages of it?*

- *Bonus:* If you have still some time, you can try automating the Given step using controller-automation (by using the `AdminController` class). Feel free to cleanup the code duplications as well in the Given step definition. You can also do this as a homework.

HINTS

- The items on the menu page are retrieved using the "`GetPizzaMenu`" API controller method of the "`MenuController`".

- How to call an API controller method? Check the step definition of the "When the client checks the home page" step!

- The method returns the items in the "`Items`" property of a "`PizzaMenuModel`" structure.

- How to work with a result returned by an API controller method? Check the step definition of the "When the client checks the home page" and the "Then the home page main message should be: "Welcome to Geek Pizza!"" steps!

- To compare the result with the expected count, you can use the "`Assert.AreEqual(expected, actual)`" assertion method.

spec**solutions**
given.when.then.

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" using controller automation.

- 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 DataTables (`Table` class) in SpecFlow!

- Generate step definition skeleton for the "Then" step and add them to the existing `StepDefinitions\GeekPizzaStepDefinitions.cs` file.

- Rename "table" parameter to "`expectedMenuItemsTable`".

- Implement the step definition using the following code first:

```
1. Assert.AreEqual(expectedMenuItemsTable.RowCount,
   _menuModel.Items.Count);
2. for (int i = 0; i < expectedMenuItemsTable.RowCount; i++)
3. {
4.     Assert.AreEqual(expectedMenuItemsTable.Rows[i]["name"],
   _menuModel.Items[i].Name);
5. }
```

- Implement the application logic to satisfy the scenario. It will be close to the one you did for the previous exercise.

- Now the test pass, but the logic you wrote into the "Then" step 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 first line that compares the counts? Have you found any case when the scenario is wrong but the scenario passes (false negative)?

- Complex table-based assertions can also be done using the CompareToSet helper method from the SpecFlow "Assist" helper library. Comment out the code in the step definition and replace it with:

```
1. expectedMenuItemsTable.CompareToSet(_menuModel.Items,
   sequentialEquality: true);
```

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

HINTS

- The length of the SpecFlow Table can be retrieved by `myTable.RowCount`.

- A row of the SpecFlow Table can be retrieved by `myTable.Rows[i]`.

- A cell within the row can be retrieved by `myRow["header"]` (or `myTable.Rows[i]["header"]`).

A3: SPLIT FEATURE FILES AND STEP DEFINITION CLASSES

Split feature file and step definition class 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 feature file: `Menu.feature`. *Do they still pass? Did you have to make any changes in the step definitions to make it pass? Why?*

- The two "Given" step definitions are all related to the "admin" interface of the application. Extract them to a separate step definition class: `AdminStepDefinitions.cs`. *How do you have to decorate the new class to make the tests pass again?*

- Split the remaining step definitions in `GeekPizzaStepDefinitions` to `HomeStepDefinitions` and `MenuStepDefinitions` accordingly.

A4: SHARE DATA BETWEEN STEP DEFINITIONS IN DIFFERENT CLASSES

Practice data sharing between step definitions placed in different classes.

- This is a refactoring exercise, so make sure the tests are passing before you would make any changes.

- The step definitions that are related to the same features of the system should be kept in the same step definition class. We have seen this in `HomeStepDefinitions` (the `HomePageModel` was shared) and also in `MenuStepDefinitions` (the list of pizzas was shared).

- In this exercise, there is a new scenario introduced in `Home.feature`: "The logged-in user name is shown on home page". The related step definitions are currently automated in the `HomeStepDefinitions` class, but to keep a healthy structure, we would like to move the `GivenTheClientIsLoggedIn` method to the `AuthStepDefinitions`, because that does not belong to the home page but to the authentication.

- We have learned how to move step definitions to another class, but this one shares data (the authentication token, stored in the `_authToken` field) with the `WhenTheClientChecksTheHomePage` method, that supposed to remain in `HomeStepDefinitions`. To fix this, we will use the **Context Injection** feature of SpecFlow by performing the following steps:

  1. Review the context class that has been prepared for you already: `AuthContext`. (Normally you would need to create such a class with properties for the data you want to share.)

  2. Inject the context into *both step definition classes*, using constructor injection based on the following example:
     ```
     public class YourStepDefinitions
     {
         private readonly AuthContext _authContext;
         public YourStepDefinitions(AuthContext authContext)
     ```

```
        {
            _authContext = authContext;
        }
    }
```

3. Wherever the code used the `_authToken` field (for reading or writing), use the `AuthToken` property of the injected context class instead: `_authContext.AuthToken`

4. The scenarios are passing again! This means that SpecFlow created instances of the `AuthContext` class whenever it was needed. *How many `AuthContext` instance are created if you run all scenarios. Why?*

- *Bonus:* The `ThenTheUserNameOfTheClientShouldBeOnTheHomePage` method of the `HomeStepDefinitions` has the user name "Marvin" hard-coded. We need that because that is the user name we use for login in the `AuthStepDefinitions`. Can you change the automation code in a way that it only mentions "Marvin" once – in the `AuthStepDefinitions`? Hint: You need to extend the `AuthContext` class.

A5: PRACTICE USING SCENARIO OUTLINES

> Convert the scenarios in the rule "The password should be valid and verified" of Registration.feature to a Scenario Outline.

- There are three scenarios within the rule "The password should be valid and verified" of the Registration.feature 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

- You can find the Scenario Outline syntax on the Gherkin Cheat Sheet.

- Add an additional first column (e.g. "description") to the Examples table to describe the purpose of the individual examples. Check the name of the generated tests!

- *Bonus:* Add a second "Examples" block and provide further test cases. What is the difference of a "test case" and an "illustrative scenario"?

- *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?

B1: REVIEW SCENARIO AUTOMATED USING WEB API AUTOMATION

> Review the automation of the scenario "The user name should be shown on the home page if logged in" to find potential maintenance and scaling problems.

- We have switched over to a new automation solution that is in the `BddWithSpecFlow.GeekPizza.API.Specs` project. For the sake of simplicity, the scenarios we have automated using the API controllers have been removed.

- The current codebase contains a single scenario: "The user name should be shown on the home page if logged in" in the `Home.feature`. The scenario has been automated already using the step definitions in the `WebApiStepDefinitions` class. The scenario should be passing (make sure it does).

- The current automation solution is a "simplest possible" approach and while it makes the scenario pass it has a few problems. *Review the step definitions and make a note of the ones you find.*

- The following orientation questions might help:

  1. Is the code clean? Are there any code duplications?

  2. Assuming that the automation solution will contain many other similar GET and POST requests, would you use these automation blocks as samples for those? Why?

  3. Do these step definitions support reusability (reusing the steps in other scenarios, potentially in different order)?

  4. Does the current structure of the code support structure the step definitions into different step definition classes?

- *How could you fix these issues?* (You should not start fixing it, just make notes so that we can discuss together.)

- *Bonus:* What is the difference between sanity and functional checks? Why are they separated?

B2: AUTOMATE SCENARIOS USING WEB API AUTOMATION

Automate "Menu" and "Registration" scenarios using Web API automation.

- The "Given" step has been already automated in `AdminStepDefinitions.cs` (it adds the expected pizzas to the database).

- Automate the remaining steps by making GET and POST requests. The exact method, endpoint and input/output data you can find in the feature files as `#TODO` comments.

- The (empty) step definition classes are prepared (`MenuStepDefinitions` and `RegistrationStepDefinitions`) and the `WebApiContext` has been injected already.

- Check `HomeStepDefinitions.WhenTheClientChecksTheHomePage` and `AuthStepDefinitions.GivenTheClientIsLoggedIn` methods to see how you can use the WebApi Context to make a GET and POST request.

- *Was it easy? Did you enjoy the separation of the infrastructural details from the functional parts of the automation?*