Behavior Driven Development with .NET Core and Visual Studio for Mac

Published by Gökhan Gökalp on May 15, 2019

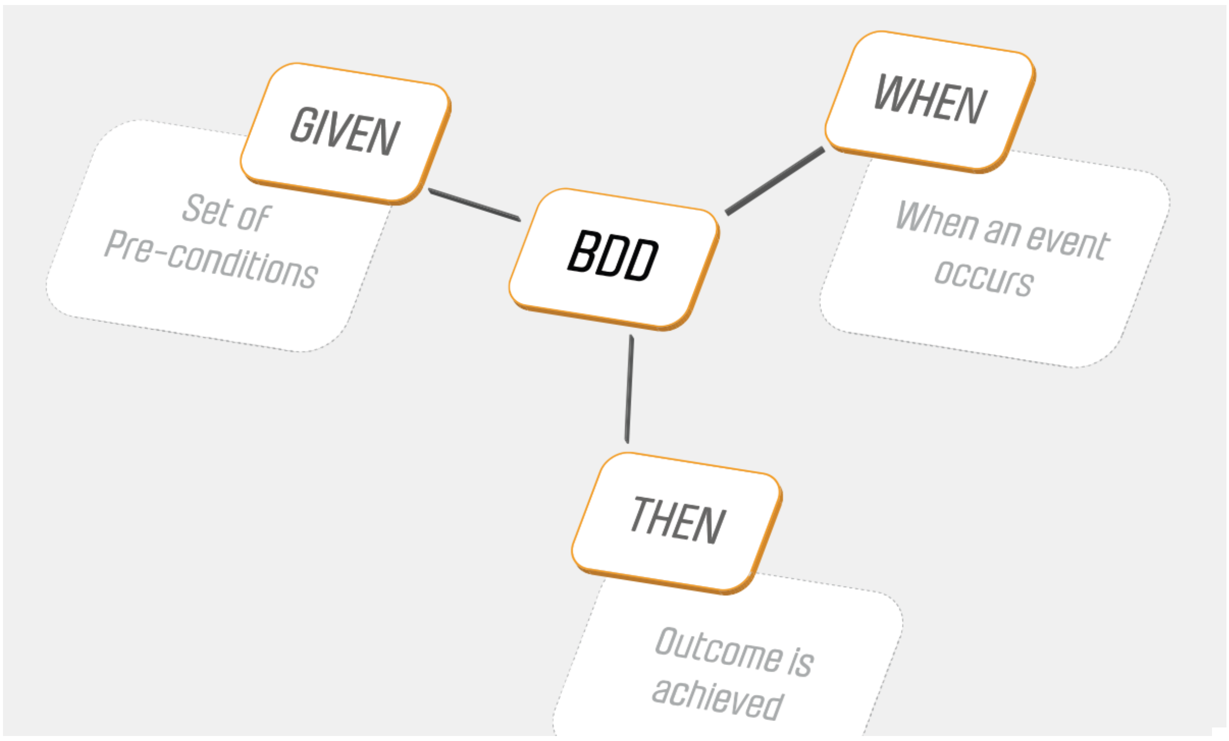
Let’s assume an agile development team, from Developer to Product Owner, from Scrum Master to Stake Holder, all of them working **collaboratively** on **product development**.

Sounds great, doesn’t it? But unfortunately, this is not always 100% possible.

So, today we will talk about ***Behavior Driven Development***.

Within the scope of this article, we will refresh our's knowledge about *BDD*, after that, we will try to develop an *API*, that contains basic functions, with *BDD* approach on ***macOS*** using ***.NET Core*** and ***Visual Studio***.

> Obviously, in order to complete this article, I have been waiting for *.NET Core* support of *SpecFlow* for a long time.



In the context of this article, we will refer to the following topics:

1. Briefly Remembering *BDD*
2. Benefits of *BDD*
3. *BDD* on *macOS* using *.NET Core* and *Visual Studio*

Briefly Remembering BDD

BDD, which is based on the *Test Driven Development* (TDD), is especially concentrating on the producing of "*quality code*".

Based on my ~2 years of experience, if I need to say that the great advantages of the *BDD* have provided us are, firstly it is an important tool against the difficulties that arise from the **communication-related** problems during the project development phase, and secondly, it provides us with the great project documentation.

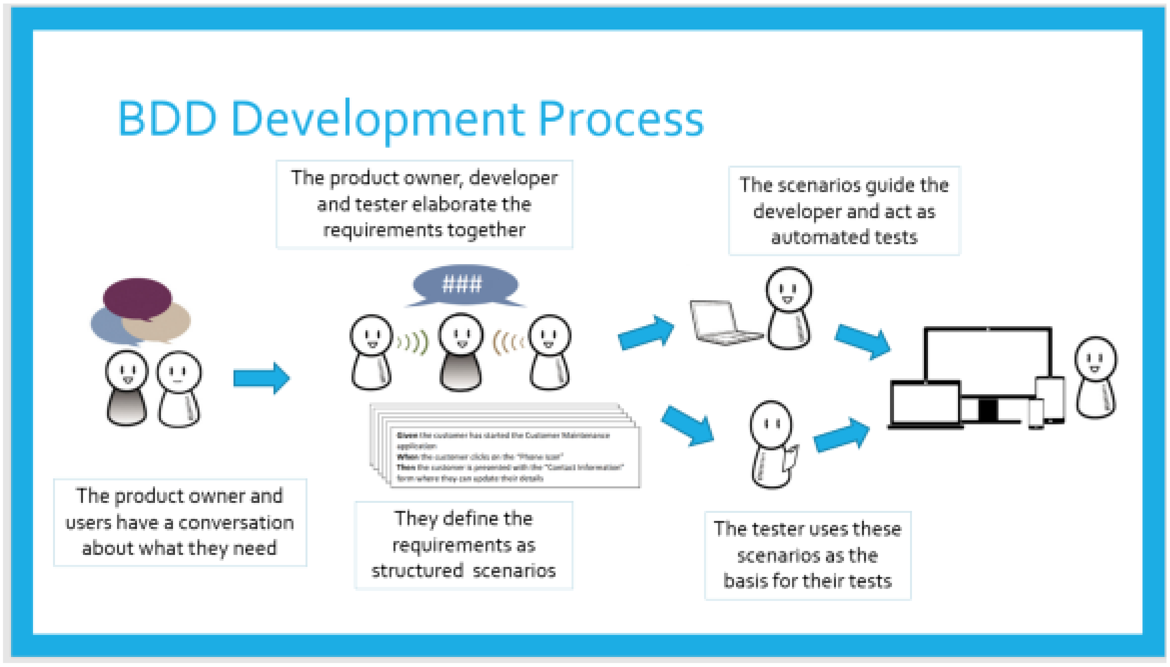
So, what are these communication-related difficulties and what are the problems?

Usually, a team, who will develop the project, is dependent on the business teams in order to understand all the needs of the customer correctly.

However, most of the time **poorly written code** and **incomplete needs** may appear due to the fact that business teams are far away from the technical side. At this point, *BDD* provides us a "common language" to solve these problems. In other words, it is acting as a **guide** to improve communication between developers, test teams and business teams and also to understand the requirements easily.

In addition, *BDD* clearly defines the **behaviors**, that will affect the **business outcome** directly, in a way that is accessible to the developers, test teams and the business teams. The focus of this definition in the *BDD* is to finding the requirements in the user story and writing the **acceptance** tests based on the requirements. That is, it draws a path for the development of the project in accordance with end-to-end acceptance criteria.

> ***NOTE***: In the *BDD*, customers also get involved in the development process.


BDD Development Process
The product owner, developer and tester elaborate the requirements together
The scenarios guide the developer and act as automated tests
###
Given the customer has started the Customer Maintenance application
When the customer clicks on the "Phone icon"
Then the customer is presented with the "Contact Information" form where they can update their details
The product owner and users have a conversation about what they need
They define the requirements as structured scenarios
The tester uses these scenarios as the basis for their tests

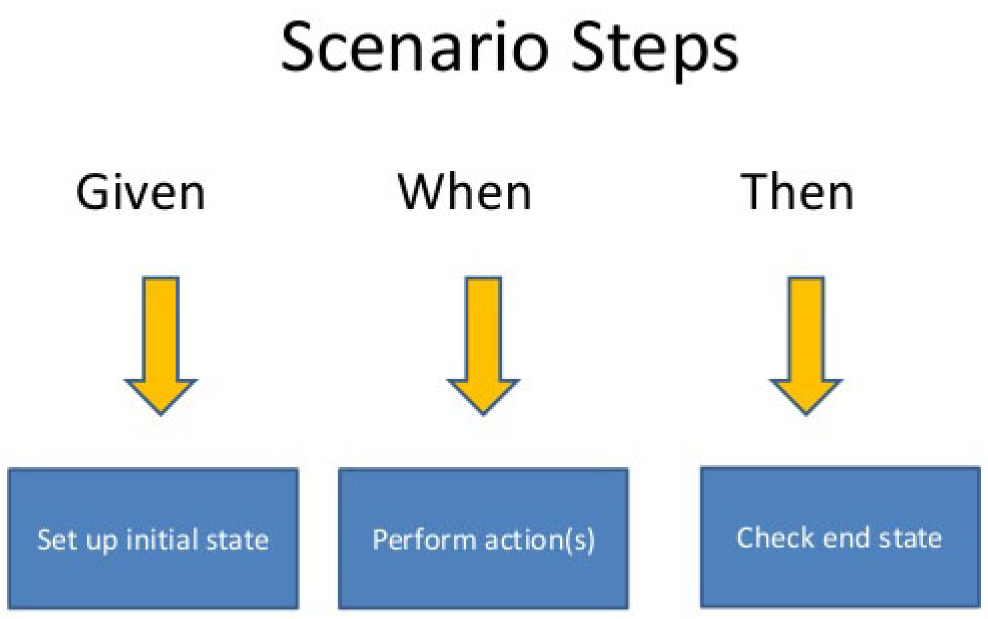
Scenarios

In the *BDD*, acceptance criteria are defined as "scenarios". The scenarios are structured and explain how a feature should behave in different situations or with different parameters.

E.g:

- X person entered the Google.
- He/she wrote "cat" in the search box.
- Search results related to "cat" are displayed.

In addition, the scenarios are written in a linguistic format called "*Gherkin*", which consists of the **Given**, **When** and **Then** sections.



- **Given**: Describes the context of the scenario.
- **When**: Defines the action.
- **Then**: In here, defines "what will happen", that is "outcome".

As we can see, the scenarios are written in a simple form of speaking language, this is making them easy to understand by all team members. It also has the characteristic of **documentation** for the project.

You can reach more detailed information about this topic from *here*.

Benefits of BDD

First of all, *BDD* is one of the methods used in test automation projects. In addition to using the test scenarios written in *Gherkin* format in the "automation process", it also provides living and up-to-date documentation of the project.

In general, the benefits are:

- It offers a simple and understandable language that can be used by each member of the team.
- Improves cooperation.
- The focus is on the customer and following the behavior of the application.
- It provides up-to-date documentation of the project.

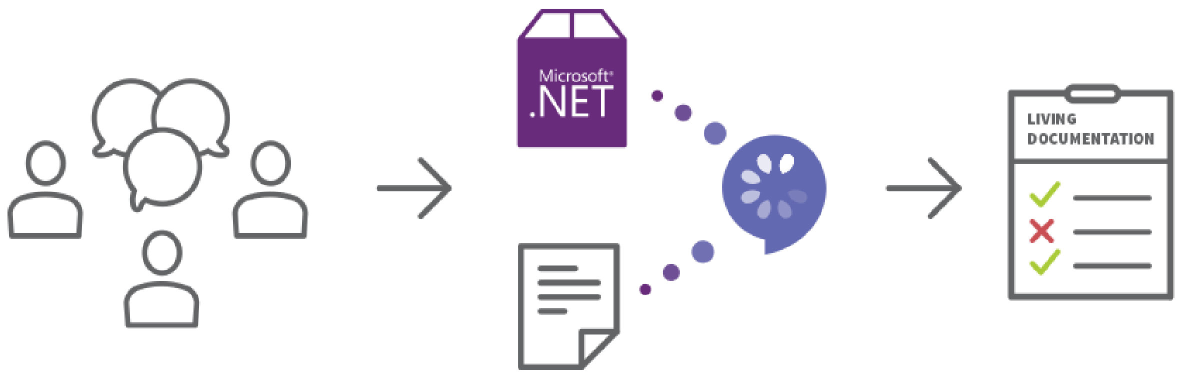
Besides, *BDD* significantly reduces the time spent on "*end user*" and "*user acceptance*" tests in the software development process.

BDD on *macOS using .NET Core* and *Visual Studio*

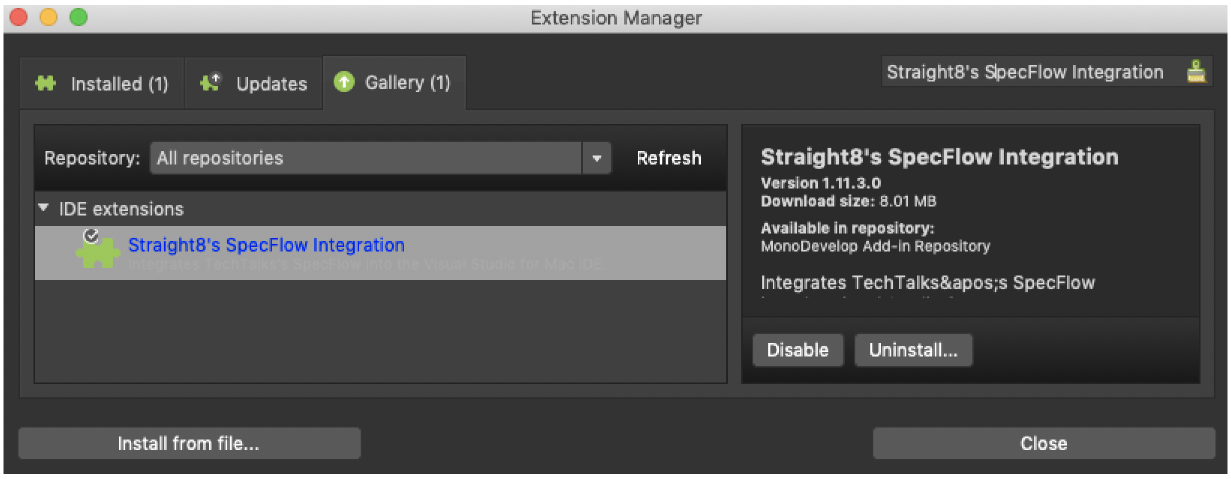
Let’s assume we are working in an e-commerce company. We are requested to develop an *API* so that users can add their favourite products to their favourite lists. Let’s develop this *API* with *BDD* and see how it works out.

I will develop the *API* on *macOS* using *Visual Studio* and *.NET Core 2.2*. Also, we will use *SpecFlow* as *BDD* framework.

SpecFlow is an open-source *Behavior Driven Design* framework that allows us to define and manage human-readable acceptance tests using the *Gherkin* parser.



Firstly, if you don’t have *Visual Studio* on *macOS*, you can download it *here*. After opening *Visual Studio*, let’s enter the “*Extensions*” tab then click to the “*Gallery*” tab. Now we need to type “*Straight8’s SpecFlow Integration*” in the search box and then install the related extension as follows.



Repository: All repositories

Straight8's SpecFlow Integration
Integrates TechTalks's SpecFlow into the Visual Studio for Mac IDE.

Straight8's SpecFlow Integration
Version 1.11.3.0
Download size: 8.01 MB
Available in repository: MonoDevelop Add-in Repository

Integrates TechTalks&apos;s SpecFlow

With this extension, we will be able to add features and step definitions to our project easily.

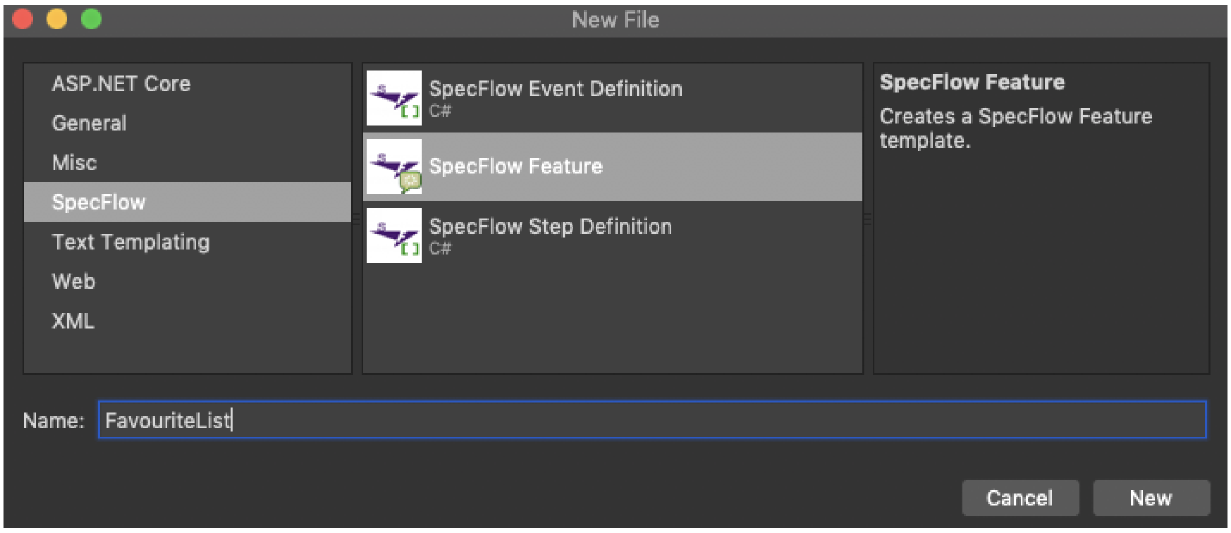
Now let's create a *.NET Core 2.2 NUnit Test Project* called "*MyFavouriteAPI.Tests*". Then, we need to include "*SpecFlow*", "*SpecFlow.Tools.MsBuild.Generation*" and "*SpecFlow.NUnit*" packages to the project via *NuGet*.

For general configuration options, we should create a configuration file called "*specflow.json*" as follows.

```
{
  "language": {
    "feature": "en-US"
  }
}
```

With this option, we specify that the feature files will be in English.

After creating the configuration file, let's create a new folder called "*Features*". Then add a new feature file in this folder called "*FavouriteList*" as follows.



Name: FavouriteList

The extension will create a template feature file as below.

```
Feature: Addition
    In order to avoid silly mistakes
    As a math idiot
    I want to be told the sum of two numbers

@mytag
Scenario: Add two numbers
    Given I have entered 50 into the calculator
    And I have entered 70 into the calculator
    When I press add
    Then the result should be 120 on the screen
```

Now let's create our own feature script by editing the contents of this template.

Defining the Scenario

What we want as a feature is that "*users can create own favourite list and add or remove products from the list in order to buy them later*".

So, let's edit the feature as follows.

```
Feature: Favourite List
    A simple favourite list that we can add or remove products in order to buy the
```

Now we can define our first scenario. Firstly, we need to create a favourite list. To do that, let's called the scenario part as "*create a new favourite list*".

So when this scenario will happen, what is the action here? For the definition of the action here, I think it is enough to say "*when I create a new favourite list*". Now, what will be the result of this operation, what is the outcome here?

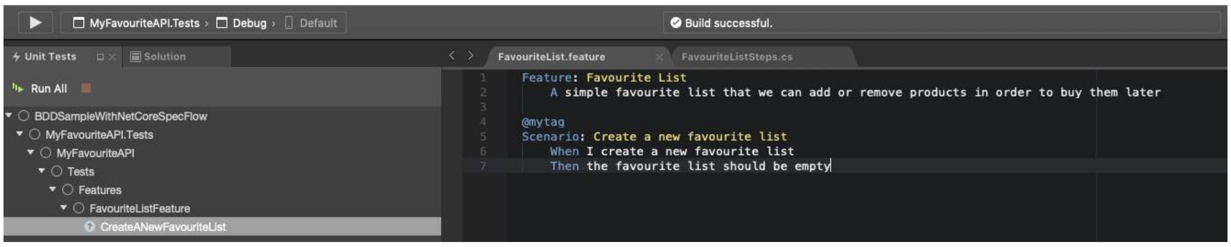
For this, we can say "*then the favourite list should be created as empty*". Based on this scenario, let's edit the feature file as follows.

```
Feature: Favourite List
    A simple favourite list that we can add or remove products in order to buy the

@mytag
Scenario: Create a new favourite list
    When I create a new favourite list
    Then the favourite list should be created as empty
```

This scenario, which we have described above, clearly explains how the work will be done, doesn't it? A simple language that can be used and understood by each member of the team.

Coding the Scenario

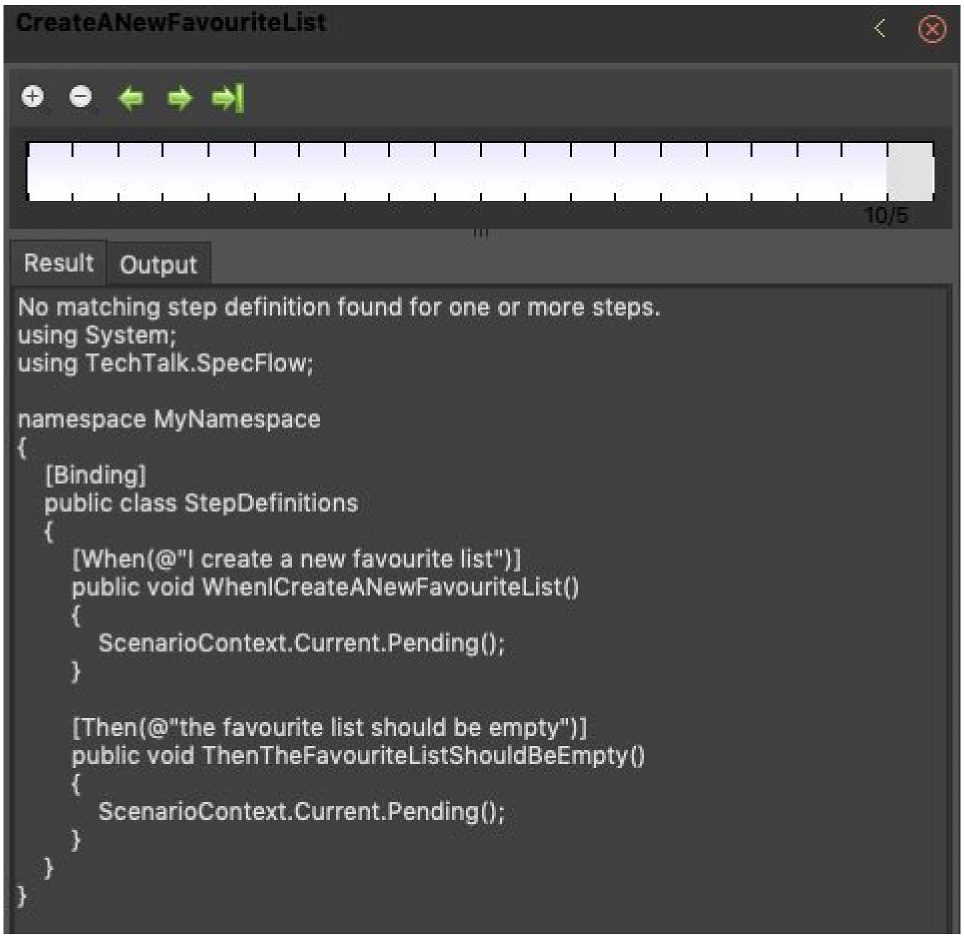
After defining the scenario, let's build the project and switch to the "*Unit Tests*" pad through the *IDE*.



```
Feature: Favourite List
    A simple favourite list that we can add or remove products in order to buy them later

@mytag
Scenario: Create a new favourite list
    When I create a new favourite list
    Then the favourite list should be empty
```

Ahha! After building the project, the extension has created the "*CreateANewFavouriteList*" test for us.

So, let's run the test and look at the result on the test result pad.



```
No matching step definition found for one or more steps.
using System;
using TechTalk.SpecFlow;

namespace MyNamespace
{
    [Binding]
    public class StepDefinitions
    {
        [When(@"I create a new favourite list")]
        public void WhenICreateANewFavouriteList()
        {
            ScenarioContext.Current.Pending();
        }

        [Then(@"the favourite list should be empty")]
        public void ThenTheFavouriteListShouldBeEmpty()
        {
            ScenarioContext.Current.Pending();
        }
    }
}
```

It gives us the "*No matching step definition found for one or more steps.*" message because we haven't written any code yet.

Now we need to define the step definitions related to our scenario. For this, we can use the sample code snippet that the extension gave us in the result pad.

Let's create a folder called "*StepDefinitions*" and define a class in this folder called "*FavouriteListSteps*".

Then copy the sample code snippet and paste it into the "*FavouriteListSteps*" class as follows.

> ***NOTE***: Don't forget to edit "*MyNamespace*" and "*StepDefinitions*" parts.

```
using System;
using TechTalk.SpecFlow;

namespace MyFavouriteAPI.Tests.StepDefinitions
{
    [Binding]
    public class FavouriteListSteps
    {
        [When(@"I create a new favourite list")]
        public void WhenICreateANewFavouriteList()
        {
            ScenarioContext.Current.Pending();
        }

        [Then(@"the favourite list should be empty")]
        public void ThenTheFavouriteListShouldBeEmpty()
        {
            ScenarioContext.Current.Pending();
        }
    }
}
```

It's pretty clear what we are supposed to do here, right?

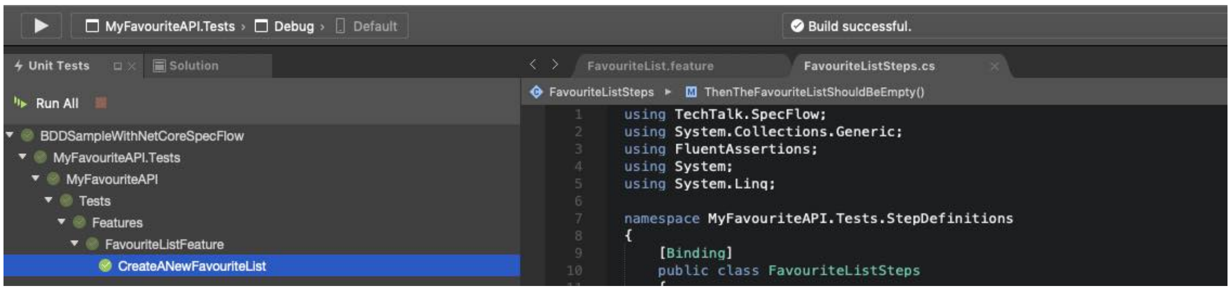
Before start to coding, we need to include the "*FluentAssertions*" package via *NuGet* to perform assertions easily.

Then, let's start coding to our sample scenario as follows.

```
using TechTalk.SpecFlow;
using System.Collections.Generic;
using FluentAssertions;
using System;
using System.Linq;

namespace MyFavouriteAPI.Tests.StepDefinitions
{
    [Binding]
    public class FavouriteListSteps
    {
        private readonly IFavouriteService _favouriteService;
        private int _favouriteListId;
        private readonly int _userId;

        public FavouriteListSteps()
        {
            _favouriteService = new FavouriteService();
            _userId = 1;
        }

        [When(@"I create a new favourite list")]
        public void WhenICreateANewFavouriteList()
        {
            _favouriteListId = _favouriteService.Create(_userId);
        }

        [Then(@"the favourite list should be empty")]
        public void ThenTheFavouriteListShouldBeEmpty()
        {
            FavouriteList favouriteList = _favouriteService.GetFavouriteList(_use

            favouriteList.Should().NotBeNull();
            favouriteList.FavouriteListId.Should().Be(_favouriteListId);
            favouriteList.ProductIds.Should().BeNull();

        }
    }

    public class FavouriteList
    {
        public int FavouriteListId { get; set; }
        public List<int> ProductIds { get; set; }

    }
    public interface IFavouriteService
    {
        int Create(int userId);
        FavouriteList GetFavouriteList(int userId, int favouriteListId);
    }

    public class FavouriteService : IFavouriteService
    {
        private readonly Dictionary<int, List<FavouriteList>> favouriteListStore

        public int Create(int userId)
        {
            int favouriteListId = new Random().Next(10);

            var newFavouriteList = new List<FavouriteList>
            {
                new FavouriteList { FavouriteListId = favouriteListId }
            };

            favouriteListStore.Add(userId, newFavouriteList);

            return favouriteListId;
        }

        public FavouriteList GetFavouriteList(int userId, int favouriteListId)
        {
            if (favouriteListStore.TryGetValue(userId, out List<FavouriteList> u
            {
                var favouriteList = userFavouriteList.FirstOrDefault(_ => _.Favo

                return favouriteList;
            }

            return null;
        }
    }
}
```

We simply coded "*When*" and "*Then*" steps. First, we have created a new favourite list for the user, then we verified that the favourite list is empty.

Now let's run the "*CreateANewFavouriteList*" test again.



```
using TechTalk.SpecFlow;
using System.Collections.Generic;
using FluentAssertions;
using System;
using System.Linq;

namespace MyFavouriteAPI.Tests.StepDefinitions
{
    [Binding]
    public class FavouriteListSteps
```

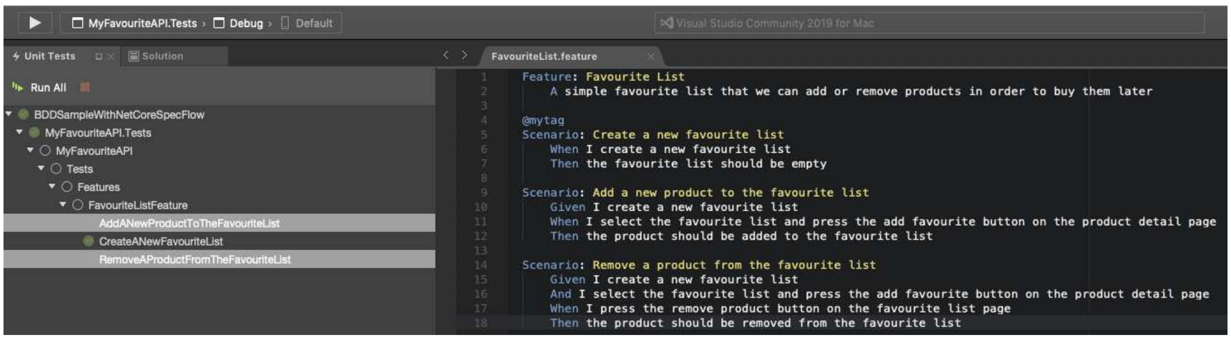
As we can see the test passed successfully.

Now we need to add one more new scenario to the feature. The user should be able to add products to his/her favourite list also delete them.

For that, let's extend our scenario as follows.

```
Feature: Favourite List
    A simple favourite list that we can add or remove products in order to buy tl

@mytag
Scenario: Create a new favourite list
    When I create a new favourite list
    Then the favourite list should be empty

Scenario: Add a new product to the favourite list
    Given I create a new favourite list
    When I select the favourite list and press the add favourite button on the p
    Then the product should be added to the favourite list

Scenario: Remove a product from the favourite list
    Given I create a new favourite list
    And I select the favourite list and press the add favourite button on the pro
    When I press the remove product button on the favourite list page
    Then the product should be removed from the favourite list
```

After saving the scenario, we need to rebuild the project and then take a look at the "*Unit Tests*" pad again.


MyFavouriteAPI.Tests › Debug › Default
Visual Studio Community 2019 for Mac
Unit Tests
Solution
Run All
BDDSampleWithNetCoreSpecFlow
MyFavouriteAPI.Tests
MyFavouriteAPI
Tests
Features
FavouriteListFeature
AddANewProductToTheFavouriteList
CreateANewFavouriteList
RemoveAProductFromTheFavouriteList
FavouriteList.feature

```
Feature: Favourite List
    A simple favourite list that we can add or remove products in order to buy them later

@mytag
Scenario: Create a new favourite list
    When I create a new favourite list
    Then the favourite list should be empty

Scenario: Add a new product to the favourite list
    Given I create a new favourite list
    When I select the favourite list and press the add favourite button on the product detail page
    Then the product should be added to the favourite list

Scenario: Remove a product from the favourite list
    Given I create a new favourite list
    And I select the favourite list and press the add favourite button on the product detail page
    When I press the remove product button on the favourite list page
    Then the product should be removed from the favourite list
```

After building the project, the extension has created the tests for the scenario, that we have defined newly, named "*AddANewProductToTheFavouriteList*" and "*RemoveAProductFromTheFavouriteList*".

Let's run the tests and take the sample method snippets on the test result pad again.

```
No matching step definition found for one or more steps.
using System;
using TechTalk.SpecFlow;

namespace MyNamespace
{
    [Binding]
    public class StepDefinitions
    {
        [Given(@"I create a new favourite list")]
        public void GivenICreateANewFavouriteList()
        {
            ScenarioContext.Current.Pending();
        }

        [When(@"I select the favourite list and press the add favourite button o
        public void WhenISelectTheFavouriteListAndPressTheAddFavouriteButtonOnThe
        {

            ScenarioContext.Current.Pending();
        }

        [Then(@"the product should be added to the favourite list")]
        public void ThenTheProductShouldBeAddedToTheFavouriteList()
        {
            ScenarioContext.Current.Pending();
        }
    }
}

No matching step definition found for one or more steps.
using System;
using TechTalk.SpecFlow;

namespace MyNamespace
{
    [Binding]
    public class StepDefinitions
    {
        [Given(@"I create a new favourite list")]
        public void GivenICreateANewFavouriteList()
        {
            ScenarioContext.Current.Pending();
        }

        [Given(@"I select the favourite list and press the add favourite button
        public void GivenISelectTheFavouriteListAndPressTheAddFavouriteButtonOnT
        {
            ScenarioContext.Current.Pending();
        }

        [When(@"I press the remove product button on the favourite list page")]
        public void WhenIPressTheRemoveProductButtonOnTheFavouriteListPage()
        {
            ScenarioContext.Current.Pending();
        }

        [Then(@"the product should be removed from the favourite list")]
        public void ThenTheProductShouldBeRemovedFromTheFavouriteList()
        {
            ScenarioContext.Current.Pending();
        }
    }
}
```

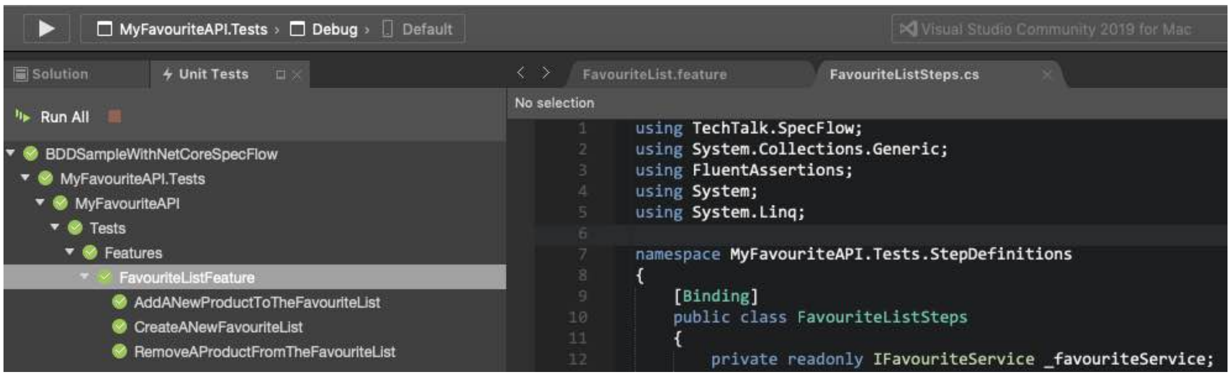
Now, in order to complete the feature, we need to implement these behaviors, which are expected from us, in the "*FavouriteListSteps*" class as follows.

```
using TechTalk.SpecFlow;
using System.Collections.Generic;
using FluentAssertions;
using System;
using System.Linq;

namespace MyFavouriteAPI.Tests.StepDefinitions
{
    [Binding]
    public class FavouriteListSteps
    {
        private readonly IFavouriteService _favouriteService;
        private int _favouriteListId;
        private readonly int _userId;
        private readonly int _productId;

        public FavouriteListSteps()
        {
            _favouriteService = new FavouriteService();
            _userId = 1;
            _productId = 1;
        }

        [Given(@"I create a new favourite list")]
        [When(@"I create a new favourite list")]
        public void WhenICreateANewFavouriteList()
        {
            _favouriteListId = _favouriteService.Create(_userId);
        }

        [Then(@"the favourite list should be empty")]
        public void ThenTheFavouriteListShouldBeEmpty()
        {
            FavouriteList favouriteList = _favouriteService.GetFavouriteList(_us

            favouriteList.Should().NotBeNull();
            favouriteList.FavouriteListId.Should().Be(_favouriteListId);
            favouriteList.ProductIds.Should().BeEmpty();

        }

        [Given(@"I select the favourite list and press the add favourite button
        [When(@"I select the favourite list and press the add favourite button o
        public void WhenISelectTheFavouriteListAndPressTheAddFavouriteButtonOnTh
        {
            _favouriteService.AddFavourite(_userId, _favouriteListId, _productId
        }

        [Then(@"the product should be added to the favourite list")]
        public void ThenTheProductShouldBeAddedToTheFavouriteList()
        {
            FavouriteList favouriteList = _favouriteService.GetFavouriteList(_us

            favouriteList.Should().NotBeNull();
            favouriteList.FavouriteListId.Should().Be(_favouriteListId);
            favouriteList.ProductIds.Should().Contain(_productId);
        }

        [When(@"I press the remove product button on the favourite list page")]
        public void WhenIPressTheRemoveProductButtonOnTheFavouriteListPage()
        {
            _favouriteService.RemoveProduct(_userId, _favouriteListId, _productI
        }

        [Then(@"the product should be removed from the favourite list")]
        public void ThenTheProductShouldBeRemovedFromTheFavouriteList()
        {
            FavouriteList favouriteList = _favouriteService.GetFavouriteList(_us

            favouriteList.Should().NotBeNull();
            favouriteList.FavouriteListId.Should().Be(_favouriteListId);
            favouriteList.ProductIds.Should().NotContain(_productId);
        }
    }

    public class FavouriteList
    {
        public int FavouriteListId { get; set; }
        public List<int> ProductIds { get; set; }
    }

    public interface IFavouriteService
    {
        void AddFavourite(int userId, int favouriteListId, int productId);
        int Create(int userId);

        FavouriteList GetFavouriteList(int userId, int favouriteListId);
        void RemoveProduct(int userId, int favouriteListId, int productId);
    }

    public class FavouriteService : IFavouriteService
    {
        private readonly Dictionary<int, List<FavouriteList>> favouriteListStore

        public void AddFavourite(int userId, int favouriteListId, int productId)
        {
            FavouriteList favouriteList = GetFavouriteList(userId, favouriteLis

            if(favouriteList != null)
            {
                favouriteList.ProductIds.Add(productId);
            }
        }

        public int Create(int userId)
        {
            int favouriteListId = new Random().Next(10);

            var newFavouriteList = new List<FavouriteList>
            {
                new FavouriteList
                {
                    FavouriteListId = favouriteListId,
                    ProductIds = new List<int>()
                }
            };

            favouriteListStore.Add(userId, newFavouriteList);

            return favouriteListId;
        }

        public FavouriteList GetFavouriteList(int userId, int favouriteListId)
        {
            if (favouriteListStore.TryGetValue(userId, out List<FavouriteList> u
            {
                var favouriteList = userFavouriteList.FirstOrDefault(_ => _.Fav

                return favouriteList;
            }

            return null;
        }

        public void RemoveProduct(int userId, int favouriteListId, int productI
        {
            FavouriteList favouriteList = GetFavouriteList(userId, favouriteLis

            if (favouriteList != null)
            {
                favouriteList.ProductIds.Remove(productId);
            }
        }
    }
}
```

I would like to mention a few points here. If there is a similar scenario that we have implemented before, we don't need to re-code it. All we have to do is add the "*Given*" context in the right place as in the feature file.

For example, to be able to add a new product into a favourite list or delete, first we need to create a favourite list. In order to do this, it will be enough to add "*[Given(@"I create a* *new favourite list")]*" attribute to the "*WhenICreateANewFavouriteList*" method we have implemented before.

Then, we have implemented the behaviours expected from us. Now let's go back to the "*Unit Tests*" pad and run all the tests.



```
using TechTalk.SpecFlow;
using System.Collections.Generic;
using FluentAssertions;
using System;
using System.Linq;

namespace MyFavouriteAPI.Tests.StepDefinitions
{
    [Binding]
    public class FavouriteListSteps
    {
        private readonly IFavouriteService _favouriteService;
```

Tada! All the scenarios' tests, that are required to complete the "*FavouriteList*" feature, are passed successfully.

When talking about the benefits of *BDD* at the beginning of this article, we have mentioned the following topics:

- It offers a simple and understandable language that can be used by each member of the team.
- Improves cooperation.
- The focus is on the customer and following the behavior of the application.
- It provides up-to-date documentation of the project.

Now let's take a look at the feature file that we created.

```
Feature: Favourite List
    A simple favourite list that we can add or remove products in order to buy t

@mytag
Scenario: Create a new favourite list
    When I create a new favourite list
    Then the favourite list should be empty

Scenario: Add a new product to the favourite list
    Given I create a new favourite list
    When I select the favourite list and press the add favourite button on the p
    Then the product should be added to the favourite list

Scenario: Remove a product from the favourite list
    Given I create a new favourite list
    And I select the favourite list and press the add favourite button on the pr
    When I press the remove product button on the favourite list page
    Then the product should be removed from the favourite list
```

This feature file has a simple and understandable language that can be used by each member of the team and also it is up-to-date documentation of the project. During the development process, it guided our code by following the behaviour of the application.

Conclusion

BDD is an important methodology that can be used especially when cooperation is needed in product development. It focuses on the user and the behaviour of the application, thus it minimizes maintenance and additional efforts. It also supports the test automation process by creating up-to-date documentation of the project.

Link: *https://github.com/GokGokalp/BDDSampleWithNetCoreSpecFlow*

References

> *https://specflow.org/getting-started/*
> *https://specflow.org/documentation/*
> *https://specflow.org/2018/specflow-3-public-preview-now-available/*

Bu makale toplam **(1369)** kez okunmuştur.

11 0

Published in .NET Core ASP.NET Core Behavior Driven Development Test Driven Development

.net core asp.net core bdd behavior driven development specflow tdd

Previous Post
Distributed Tracing with OpenTracing API of .NET Core Applications on Kubernetes

Next Post
Kubernetes-based Event Driven Autoscaling with KEDA, RabbitMQ and .NET Core

Be First to Comment

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name*

Email*

Website

Spam protection: Sum of 7 + 10 ?
*

Post Comment