**Разработка тестов SpecFlow**

1 Основные понятия

**Specflow** - это “исполняемая спецификация” .NET фреймворк с открытым исходным кодом, ориентированный на следующие подходы к разработке:

Behavior Driven Development (BDD) - разработка управляемая через поведение

Acceptance test–driven development (ATDD) - разработка на основе приемочных испытаний

**Specflow тесты** - Это тесты написанные “максимально естественным языком” для каких либо испытаний для приемки конечного продукта. В Specflow для этого используется язык Gherkin (см. далее), пример теста с разбором **здесь**

**Gherkin** – это структурированный естественный язык (Natural Language), который используется для описания поведения системы по заданному сценарию. Может использоваться в рамках Agile-практики Behavior Driven Development (BDD)

**Selenium WebDriver** — это инструмент для автоматизации действий веб-браузера. В большинстве случаев используется для тестирования Web-приложений, но этим не ограничивается. В частности, он может быть использован для решения рутинных задач администрирования сайта или регулярного получения данных из различных источников (сайтов).

2 Понятия тестов Specflow

**Scenario** - это конкретный тест (сценарий) состоящий из шагов (Step - см. далее)

**Step** - это шаг конкретного теста (сценария), выполняющий определенное действие/проверку - примеры:

*And Click on element with class 'zoom-in-btn'* - Нажать на кнопку с классом zoom-in-btn

*Then Zoom indicator value should be '300%' -* Проверяет что значение индикатора zoom должно быть 300%

***Тэг*** *- это директива которая начинается с @ для того чтобы определить область (scope) для выполнения шагов(steps)/хуков (hooks) тестов*

*Ставится в самом начале файла Feature*

Например:

@webApp - определяет тесты которые относятся к веб-приложению (будет запущен браузер)

@api - определяет тесты для тестирования API (браузер не будет запущен)

Тэги можно использовать не только для общего разделения тестов frontend или backend, а например для задания какого-то специфического общего контекста (см. Далее FeatureContext, ScenarioContext) между тестами или для всей группы тестов (для всей Feature)

3 Что могут тестировать тесты Specflow принцип работы

Specflow тесты могут тестировать не только frontend (UI), можно тестировать и бэкенд (запросы и ответы к серверу) api. Общий принцип работы такой:

* В файлах \*.feature есть текст с текстами написанными на языке Gherkin
* Специальный компилятор от Specflow генерирует по данным файлам исходный код на языке C# с тестами NUnit
* Исходный код на языке C# компилируется компилятором языка C#
* на движке Specflow выполняются тесты (Scenario) написанные на языке Gherkin
* **Каждый шаг (Step) в тесте описан реализацией на C# и языке .NET**
* Все взаимодействие с браузером происходит через **Selenium Webdriver**

3.1 Разбор группы тестов (конкретного файла Feature), их шагов и их реализации

На рис.1 представлен фрагмент файла GroupDocs.Viewer.web.Document.Viewer.feature, **на линии 1** один находится тэг

Рис.1 - Файл GroupDocs.Viewer.web.Document.Viewer.feature

с группой тестов (Feature)

@webapp что указывает что тесты относятся к тестам веб-приложения и что нужно запустить веб-браузер для выполнения тестов, как это реализовано в коде будет указано чуть позднее.

**Далее на линии 2** ключевое слово **Feature** начинает собственно Feature - группу тестов, через пробел идет текстовое описание “GroupDocs Viewer Web viewer check” - в данной Feature находятся тесты тестирующие интерфейс, конкретно - кнопки экрана отображения документа.

**Далее на линии 4** ключевое слово **Background** начинает специальную секцию (Background секцию) **которая будет выполняться перед каждым тестом** (Scenario) в данной Feature.

**На линии 5** - имеется **ШАГ (Step)**

Given Successfully rendered '6-pages.docx' document

То есть “Дан успешно прорисованный ‘6-pages.docx’ документ” - то есть проверяется что документ был загружен и отображен на экране.

Посмотрим на реализацию данного шага (в студии для просмотра можно нажать F12):

3.1.1 Разбор реализации шага успешного отображения документа

Как видно из кода выше шаг GivenSuccessfullyRenderedDocument включает в себя вызов 4 функций, которые тоже являются шагами, которые можно использовать отдельно. Пример такого использования представлен далее:

По функциям (шагам):

3.1.1.1 Шаг Given Opened GroupDocs Viewer Web Application

Дано открытое веб-приложение GroupDocs Viewer - то есть в уже открытом браузере (запускается перед тестами) и открывается заданный URL:

То есть происходит вызов функции OpenPage в объекте браузер где далее проверяется что страница с заданным URL не открыта и далее происходит переход по данному URL и для удобства проверки была добавлена задержка в 5 секунд.

3.1.1.2 Шаг Given Document is uploaded

В данном шаге выполняется и проверяется загрузка документа, то что он будет отображен - это будет проверено позднее, в данном шаге только строго загрузка.

В данном шаге составляется и берется путь файла из каталога ресурсов, а далее браузеру дается команда щелкнуть на input с классом uploadfileinput, подождать когда появится элемент с классом top-toolbar - панелью отображения документ.

Рис. 2 - элемент uploadfileinput, загрузчик файла для отображения

Посмотрим как реализована собственно собственно загрузка файла:

Через свойство WebDriver сначала вызывается функция нахождения элемента input по локатору By.ClassName(“uploadfileinput”), далее посылаются нажатия кнопок - ПОЛНЫЙ путь к файлу который будет грузится, для удобства отладки тестирования добавлена задержка в 5 секунд.

Данный загрузчик на странице устроен таким образом что кнопку загрузки файла нажимать не нужно поэтому загрузка начинается сразу:

Рис. 3 - после установки файла в input идет загрузка

Через некоторое время отобразится документ с панелью кнопок просмотра:

Рис. 4 - загруженный документ отображается в браузере для просмотра

Далее происходит переход к следующему шагу т.к. элемент панели с классом top-toolbar отобразился.

3.1.1.3 Шаг Then document viewer buttons panel appears

В данном шаге проверяется что элемент с классом top-toolbar видим на экране (см. Рис. 4)

3.1.1.4 Шаг ThenRenderOutputShouldAppear

На данном шаге проверяется что появился элемент с классом gd-page-image - с отрендеренным документом

Рис. 5 - Элемент с прорисованным документом имеется на экране

3.1.1.5 Шаг ThenPagesIndicatorShouldShowOfTotalPagesOnButtonsPanel

В данном шаге проверяется значение селектора страниц, а за одно и общее количество страниц документа. Как только документ отрисован, должна быть открыта первая страница и он должен содержать 6 страниц. Для этого берется HTML значение (внутренний innerHTML) элемента с классом **current-page-number** и через разделитель получается два значения:

1 часть - текущая выбранная страница

2 часть - общее количество страница

Далее выбрасывается ошибка если:

* Текст вообще пустой
* если не удалось распарсить значение полученные через черту
* если значения полученные не совпадают

3.2 Контексты группы тестов (Feature) и теста (Scenario)

В Specflow есть два контекста:

FeatureContext - контекст для того чтобы разделить какие-то общие свойства/значения общие для ВСЕХ ТЕСТОВ находящиеся в пределах Feature файла.

ScenarioContext - контекст для того чтобы сохранить и оперировать с свойствами/значениями в пределах ТОЛЬКО ОДНО теста.

3.3 Создание и удаление контекстов (creation and dispose)

FeatureContext - создается НОВЫЙ КАЖДЫЙ РАЗ ПРИ ВЫПОЛНЕНИИ ГРУППЫ ТЕСТОВ (feature), во время запуска первого теста. Удаляется когда все тесты (Scenarios) завершены.

ScenarioContext - создается КАЖДЫЙ НОВЫЙ РАЗ ПРИ ВЫПОЛНЕНИИ КОНКРЕТНОГО ТЕСТА (Scenario). Удается как-то только тест (Scenario) завершен.

4 Разработка собственных тестов, шагов и контекстов

4.1 Обзор начального набора для разбработки GroupDocs.Apps.Common.Tests

Общий начальный набор предусмотрен в проекте GroupDocs.Apps.Common.Tests:

| **Папка/Файл** | **Для чего предназначен** |
| --- | --- |
| Папка Configuration | Для получения определяемых параметров из app.config |
| Configuration\AppConfig.cs | Автосоздание файла конфигурации, получение значений из файла app.config |
| Configuration\TestsConfiguration.cs | Общий класс для получения конфигурации тестов (для всех не только UI тестов) |
| Configuration\UiTestsConfigration.cs | Общий класс для конфигурации UI тестов (наследние от TestsConfiguration) |
| Core\UiTestsBaseSteps | Набор основных предопределенных шагов UI тестов для последующего использования в тестах и их написания |
| Enums\TestsRunLocation.cs | Enum TestsRunLocation - со значениями Local, Ci  Определяет где выполняются тесты:  Local - локально на компьютере разработчика  CI - На сервере CI |
| Exceptions\UiTestsException.cs | Основное исключение для UI тестов UiTestsException |
| UI | Содержит подготовленные для разработки основные объекты для разработки UI тестов |
| UI\Context\UiTestsFeatureContext | Базовый контекст UI тестов группы тестов (UiTestsFeatureContext) |
| UI\Browser.cs | Объект Browser для:   * взаимодействия с браузером (подождать когда появится элемент, открыть новую страницу с заданным URL) * для получения данных о открытой в браузере странице (URL) * для получения информации о элементах на странице (поиск элементов, поиск их значений) |
| UI\ FeatureContextKeys.cs | Ключи для хранения в словаре FeatureContext |

4.1.1 Обзор предопределенных шагов UI тестов

Предопределенные шаги для написания тестов находятся в классе UiTestsBaseSteps, в который инжектируется FeatureContext UiTestsFeatureContext. Вместо '(.\*)' нужно вписывать необходимое значение, например так:

Take screenshot with name 'test.png'

| **Директива шага на Gherkin** | **Что делает** |
| --- | --- |
| Take screenshot with name '(.\*)' | Делает скриншот текущей открытой странице в браузере и сохраняет в указанную папку, которая указана в ключе ScreenshotsDirectory файла конфигурации - формат только PNG |
| Navigate to URL '(.\*)' | Перейти к заданному URL, для удобства добавлено ожидание в 2 секунды |
| Click on element with class '(.\*)' | Сделать щелчок мышью на элементе с указанным CSS классом |
| Element '(.\*)' is not visible | Проверить что на момент выполнения шага заданный CSS классом не доступен |
| Given Upload file with name '(.\*)' to input with class '(.\*) | Выбрать указанный файл (полный путь) в загрузчик с указанным CSS классом.  Для инициации загрузки нужно произвести щелчок на кнопке (примечание в образцах тестах GroupDocs.Viewer web app загрузчик иницировал загрузку после выбора файла самостоятельно) |
| Then Upload file with name '(.\*)' to input with class '(.\*) | Тоже самое, только для возможности использования со словом Then |
| Then Wait (.\*) secs | Подождать заданное число секунд |
| Given Wait (.\*) secs | Тоже самое только для возможности использования со словом Given |

4.2 Конфигурация UiTestsConfigration

Образец файла конфигурации

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<appSettings>

<add key="TestsRunLocation" value="Ci" />

<add key="ScreenshotsDirectory" value="UiTestsResults\Screenshots" />

<add key="DownloadDirectory" value="UiTestsResults\Downloads" />

<add key="WebApplicationRootUrl" value="https://products.groupdocs.app/viewer/" />

</appSettings>

</configuration>

**TestsRunLocation** - указывает где выполняются тесты локально (Local) или на Ci. Если указан Ci - окно браузера будет скрыто, но тесты будут выполняться.

**DownloadDirectory** - относительный путь, папка в которую будут загружаться файлы, если будет щелчок на кнопке загрузки файла. Если заданного каталога на диске нет - создается автоматически.

**ScreenshotsDirectory** - относительный путь, папка в которую будут складываться сделанные скриншоты. Если заданного каталога на диске нет - создается автоматически.

**WebApplicationRootUrl** - Корневой URL Web приложения, в данном случае GroupDocs.Viewer web application

Получить значение в конфигурации можно обращением к статическому свойству, например

UiTestsConfigration.DownloadDirectory вернет каталог загрузки файлов

4.3 Создание собственного Feature контекста на примере UiTestsFeatureContext

Для того чтобы создать свой собственный контекст нужно создать новый класс с конструктором принимающим в качестве параметра FeatureContext:

Мы не должны порождать новый класс от FeatureContext, т.к. в SpecFlow контекст передается через встроенный механизм BoDi Dependency injection - то есть контекст *инжектируется*. В интернете встречается такой способ получения контекста FeatureContext.Current - данный способ является **устаревшим.** Требуется получать контекст через инжектирование.

Далее нам потребуется добавить внутреннюю переменную хранящую текущий FeatureContext:

Т.к. У нас UI тесты на потребуется взаимодействовать с браузером который мы должны создать только ОДИН раз, поэтому добавим соответствующее свойство:

Теперь нам нужно запомнить значение объекта ВНУТРИ FeatureContext, для этого Specflow нам предлагает следующие методы Get и Set:

Теперь прописываем установку и получение объекта Browser в геттерах и сеттерах проперти:

Инициализированный браузер останется внутри контекста, о том как прописать инициализацию браузера перед запуском группы тестов в следующем разделе.

4.4 События (hooks) Specflow и создание своих событий

Для того чтобы выполнить какую то логику до/после тестов/группы тестов(feature)/конкретного теста (scenario) в specflow предусмотрены события (hooks):

| **Атрибут** | **Использование тэгов** | **Описание** |
| --- | --- | --- |
| [BeforeTestRun]/[AfterTestRun] | **-** | Логика выполняется до/перед всеми тестами |
| [BeforeFeature]/[AfterFeature] | **+** | Логика выполняется до каждой группы тестов или после  **Метод должен быть статичным и публичным** |
| [BeforeScenario] или [Before]  [AfterScenario] или [After] | **+** | Логика выполняется до каждого или после каждого теста |
| [BeforeStep]  [AfterStep] | **+** | Выполняется перед/после каждым шагом теста |

Для создания собственной логики для использования в данных событиях создайте класс и поставьте ему атрибут [Binding]

А теперь добавим статический метод с параметром FeatureContextа (который будет получаться путем инжектирования) и назовем его например BeforeFeature, и поставим перед ней атрибут [BeforeFeature]

Т.к. как нам нужно чтобы браузер **запускался только при UI тестах**, в скобках проставляем в кавычках webApp - то есть браузер будет запускаться когда в самом начале есть тэг webApp (см. Рисунок 1).

Код указанный в данном метод будет выполняться перед каждой группой тестов с указанным тэгом.

В данном методе пропишем получение временного идентификатора папки и других параметров, а также создания (открытия) веб-браузера.

После окончания группы тестов нам требуется выгружать браузер, добавим теперь аналогично метод AfterFeature с передачей Feature контекста в метод, с атрибутом AfterFeature:

Добавим туда логику выгрузки (закрытия) браузера, объект браузера будет получен из инжектированного контекста:

4.5 Подготовка Visual Studio для разработки тестов и добавления шагов

Чтобы было легче вести разработку собственных шагов для Visual Studio поставьте расширение

“Specflow for visual studio”, зайдите в Extensions-> Manage Extensions и сделайте поиск по слову “Specflow”

Рис. 6 - установка расширения Specflow для студии

Далее убедитесь что расширение установлено и включено. Для этого зайдите в Extensions->Manage Extensions и убедитесь что расширение присутствует и кнопка “Enable” отсутствует, присутствует только только кнопка “Disable”.

Рис. 7 - проверка что расширение установлено и включено

4.5 Создание собственного проекта с тестами

4.5.1 Добавление тест проекта

Сначала создайте проект тестов для собственного проекта вида GroupDocs.Apps.[ВАШ ПРОЕКТ].Tests.Specflow относительно папки проекта проект с тестами должен быть в папке

Groupdocs.app.web\src\

Рис.8 - Добавление проекта с тестами

Рис. 9 - Добавление проекта с тестами - указание имени проекта и правильного местоположение внутри основного проекта

Для получения доступа к предопределенному набору для разработки тестов добавьте ссылку (add reference) на проект GroupDocs.App.Common.Tests

Рис. 10 - Добавление проекта с тестами - указание на сборку GroupDocs.Apps.Common.Tests

4.5.2 Добавление необходимых NuGet пакетов

Добавьте к пакету тестов следующие NuGet пакеты:

* Selenium.Support
* Selenium.WebDriver.ChromeDriver
* SpecFlow.NUnit.Runners4.5.3 Создание собственной группы тестов Feature

Создайте в проекте GroupDocs.Apps.[ВАШ ПРОЕКТ].Tests.Specflow папку Features, а далее добавьте файл типа SpecFlow Feature File, для примера назовем нашу группу тестов TestYouTube.feature

Напишем текст с шагами теста и обязательно В САМОМ НАЧАЛЕ тэг webApp.

Текст теста написан теперь нам потребуется к каждому шагу написать “обвязку” (backend) реализацию. Для этого наведем курсор мыши на первый шаг, щелкнем правой кнопкой и выберем

“Go to Definition”, либо сразу нажимаем F12

Т.к. Реализации данного шага нет будет SpecFlow плагин предложит сгенерировать и скопировать в буфер обмена.

Теперь создадим в папке Core класс в котором у нас будут все реализации наших шагов:

Чтобы мы смогли использовать предопределенный набор шагов и объект Browser наследуемся от базового класса UiTestsBaseSteps и обязательно классу добавим атрибут Binding.

А теперь добавим ранее скегерированную заготовку кода в наш класс.

В случае с первым шагом все просто открыть на заданный URL и все, со следующим шагов уже нужно использовать css селекторы:

Получаем элемент по ид элемент input поисковой строки search, а далее посылаем нажатия клавиш. Можно заметить что мы именно посылаем нажатия клавиш, а не устанавливаем значение через атрибут.

После того как мы набрали в поиск что нам нужно нажимаем кнопку поиска. Все естественно и так же как если бы это делал пользователь.

Рис. 11 - поиск по запросу в YouTube

Далее добавлена задержка в 10 секунд на открытие страницы (этот шаг уже реализован в базовом классе шагов), а после нам нужно проверить и щелкнуть на нужном нам плейлисте.

Реализуем шаг проверки наличия нужного плейлиста в списке:

Тут мы получаем элементы типа плейлист из списка по классу ytd-playlist-renderer, и смотрим что у них внутри есть заданное имя нужного в тесте плейлиста

Рис. 12 - поиск по запросу в YouTube

Когда мы проверили что нужный плейлист есть, щелкнем на нем чтобы посмотреть.

Реализуем шаг поиска ссылки плейлиста и щелчка на ней:

Мы находим все элементы по css селектору (обратите внимание в коде css Selector задается как By.CssSelector), находим соответствующую ссылку внутри которой есть имя нужного плейлиста и уже щелкаем на ней.

После щелчка YouTube должен открыть первое видео в плейлисте и начать его воспроизводить.

Рис. 13 - Видео открылось и воспроизводится

Добавим код шага “Video is Playing”, нам нужно будет удостовериться что плеер видим на экране и что он собственно сейчас именно воспроизводит видео.

Проверим наличие плеера на экране по классу html5-video-player, и проверим что он воспроизводит проверяя наличие класса playing-mode.

4.5.4 Запуск тестов

Тесты появятся в Test Explorer в Visual Studio как и остальные обычные тесты, но перед тем как они появятся нужно СОБРАТЬ ПРОЕКТ с тестами. После сборки мы увидим новый подготовленный тест.

Рис. 14 - Новый тест появился в Test Explorer

Пробуем запустить наш тест и он даст нам ошибку. Почему? Дело в том что SpecFlow как было ранее в этом материале описано SpecFlow может тестировать не только UI, поэтому браузер автоматом не запускается.

Рис. 15 - Новый тест упал с ошибкой из-за того что браузер не был запущен

Ошибка The given key 'MainBrowser' was not present in the dictionary произошла потому-что в словаре FeatureContext SpecFlow элемент этого словаря с ключом MainBrowser не был задан, т.к. Браузер не был запущен.

Для того чтобы браузер запускался, нужно реализовать его запуск перед группой тестов (Feature) и выгрузку после окончания тестов в Feature, используя хуки (события hooks). Добавьте код из раздела 4.4 и браузер будет запускаться.

После снова щелкнув мышкой на данном тесте запустим его, произойдет запуск браузера и можно будет видеть все действия выполняемые тестом в браузере.

Рис. 16 - Новый тест прошел после добавления хуков/событий

Если мы щелкнем мышью на “Open additional output for this result” то увидим детализацию по шагам и времени их выполнения, это же окно можно и нужно использовать для отладки теста если он упадет.

Рис. 17 - Детализация выполнения теста по шагам

4.6 Запуск теста из командной строки и на сервере CI GitLab

Тест запускается из командной строки через команду

**dotnet test --logger "console;verbosity=detailed" src/GroupDocs.Apps.Viewer.Tests.Specflow/GroupDocs.Apps.Viewer.Tests.Specflow.csproj**

Где src/GroupDocs.Apps.Viewer.Tests.Specflow/GroupDocs.Apps.Viewer.Tests.Specflow.csproj - это полный путь к файлу проекта, где находятся тесты, вместо проекта можно еще давать путь скомпилированной DLL с тестами.

Для настройки запуска тестов на сервере CI необходимо отредактировать файл .gitlab-ci.yml

1. (опционально) добавить новый этап ui\_tests\_prod
2. Добавить новую job viewer\_app\_ui\_tests (имя можно поставить свое) stage: ui\_tests\_prod добавить только если добавлялся новый этап
3. В данной job прописать команду выше

Пример настройки .gitlab-ci.yml

stages: - build - deploy\_staging - deploy\_prod **- ui\_tests\_prod**

viewer\_app\_ui\_tests: **stage: ui\_tests\_prod**  script: - dotnet test --logger "console;verbosity=detailed" src/GroupDocs.Apps.Viewer.Tests.Specflow/GroupDocs.Apps.Viewer.Tests.Specflow.csproj only: - staging when: manual

Рис. 18 - Новое задание (job) после редактирования файла YML.

Чтобы тесты запускались в режиме CI в проект надо будет добавить файл app.config (выставить обязательное копирование если файл новее) в котором выставить режим выполнения CI, иначе SpecFlow будет пытаться открыть окно браузера, что нужно только в режиме Local (по-умолчанию тесты запускаются в локальном режиме с показом окна браузера).

Рис. 19 - Пример файла app.config с выставленным режимом CI.

О конфигурации и ее ключах см. раздел 4.2