OrdiPat All4Test

Automatisation de Test Selenium Webdriver & SpecFlow (BDD)

Client: Ordipat

Zied Hannachi
Team Leader Test Auto

Insaf Kssouri
SDET

29/01/2020

Plan

Test Automatique

L'automatisation des tests est l'utilisation de logiciels

- *Pour définir les conditions préalables de test.*
- *Pour contrôler l'exécution des tests.*
- *Comparer les résultats réels aux résultats prévus.*

Généralement, l'automatisation des tests implique l'automatisation d'un processus manuel déjà en place qui utilise un processus de test formalisé.

L'objectif principal de l'automatisation est de réduire le temps de test des zones qui ont été déjà testées au même niveau qualitatif. Ainsi on évitera la perte de temps, argent et effort humain,

All4Test

Pourquoi et quand automatiser?

- *Tests de régression fréquents*
- *Exemple de test répété*
- *Tests d'acceptation par l'utilisateur*
- *Feedback plus rapide aux développeurs*
- *Réduire l'effort humain*
- *Testez la même application sur plusieurs environnements*

Selenium Web Driver

Selenium est un framework de test logiciel pour le web qui facilite l'automatisation pour les navigateurs.

- WebDriver est conçu pour fournir une interface de programmation plus simple et plus concise, en plus de répondre à certaines limitations de l'API Selenium-RC.
- Il permet d'utiliser un langage de programmation pour écrire des scripts de test dans différents langages de programmation comme html, Java, .net, perl, ruby et qui permet d'utiliser des opérations conditionnelles, des boucles et d'autres concepts de programmation. .

Selenium-WebDriver a été développé pour mieux supporter les pages web dynamiques où les éléments d'une page peuvent changer sans que la page elle-même soit rechargée. L'objectif de WebDriver est de fournir une API orientée objet bien conçue qui offre une meilleure prise en charge des problèmes de test avancés d'applications Web modernes

Selenium est un outil open source
avec le soutien de l'entreprise.

Pourquoi Sélénium

Les tests peuvent être exécutés sur la plupart des navigateurs Web modernes

Selenium se déploie sur les plateformes Windows, Linux et Macintosh.

Behavior Driven Development

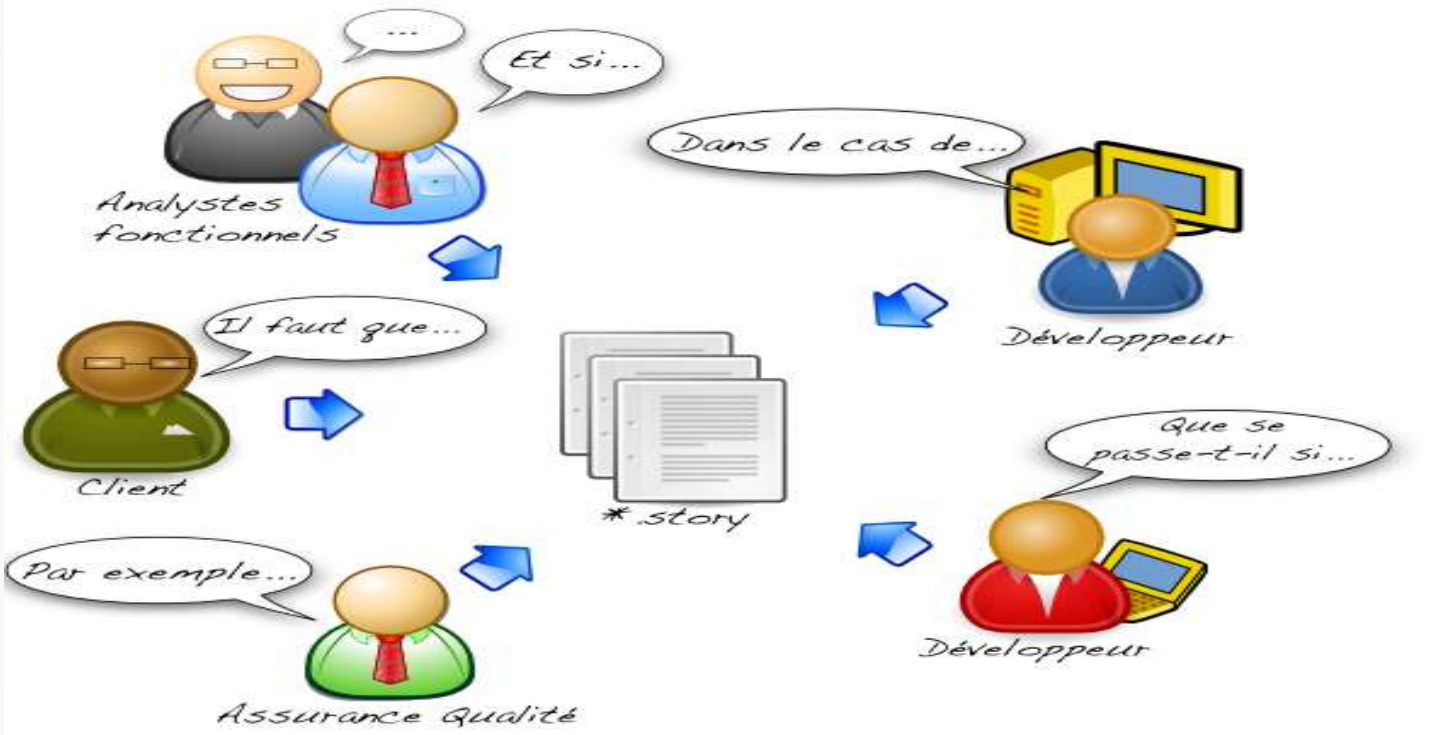
Le **BDD** est une méthode agile qui encourage la collaboration entre les développeurs, les responsables qualités, les intervenants non-techniques et les analystes participant à un projet de logiciel.
Le BDD est présenté comme une évolution du TDD (Test Driven Development).

- le BDD va guider le développement d'une fonctionnalité, tandis que le TDD guidera son implémentation.

Il s'agit d'écrire des tests qui décrivent le comportement attendu du système et que tout le monde peux comprendre

Les Avantage de BDD

1 Un dialogue restauré



L'écriture des scénarios se fait de manière collective: développeurs, clients, équipe support, …: tout le monde peut participer à l'expression du besoin

Les Avantage de BDD

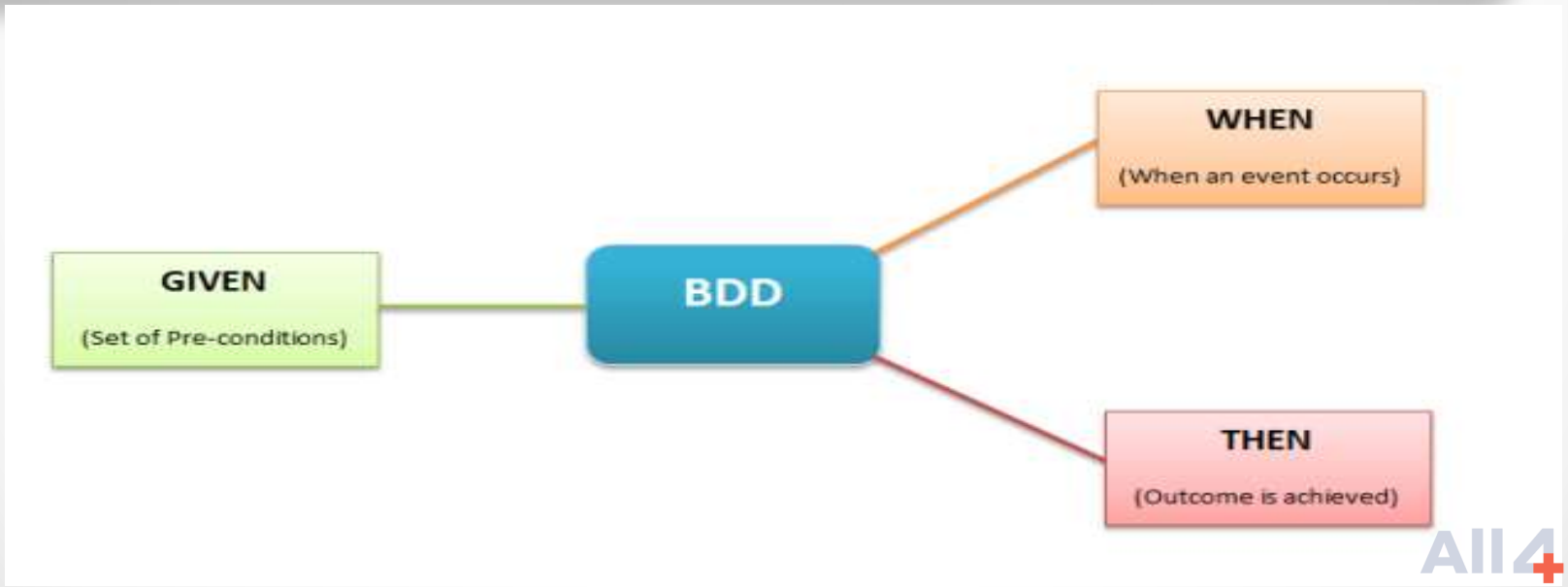
2. **Un développement guidé et documenté**
3. **Une documentation à jour et toujours disponible**
4. **Des tests en continu sur les fonctionnalités**
5. **Passer de la réflexion sur les «tests» à la réflexion sur le «comportement»**

Il s'agit d'une méthodologie de travail, permettant d'écrire des tests compréhensibles à la fois par le client et par le développeur et s'intégrant directement dans la base de code.

All4Test

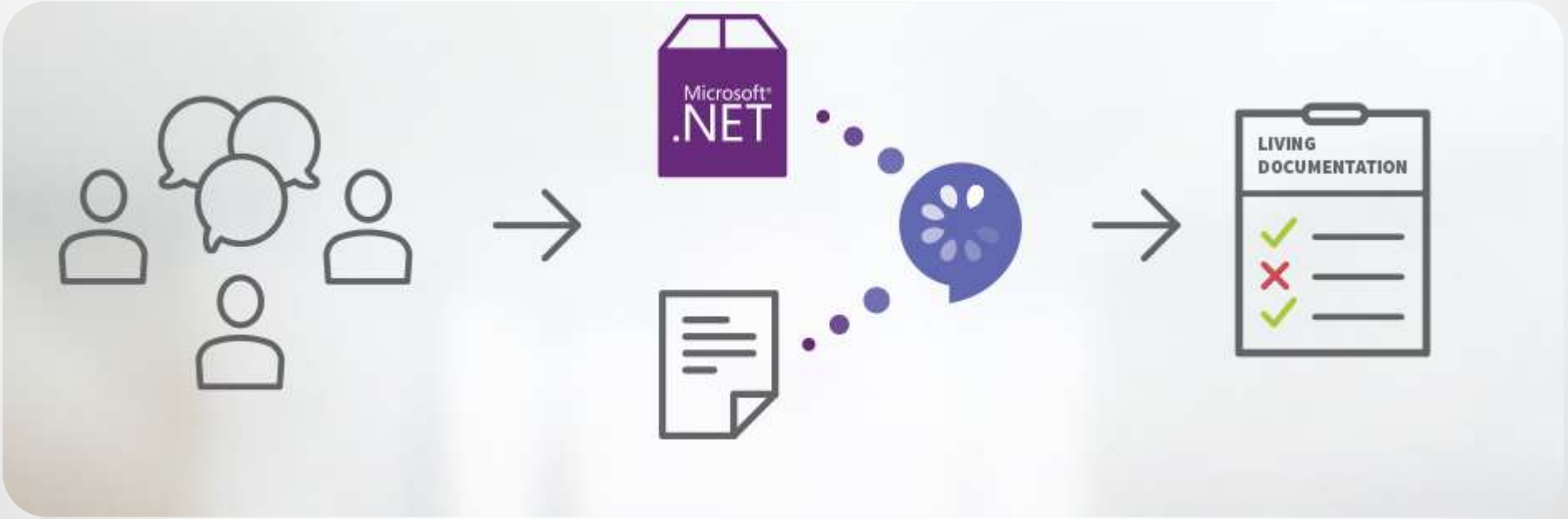
Qu'est-ce que SpecFlow?

SpecFlow est à la fois un composant ou framework de test, et un plug-in pour Visual Studio, qui permet d'écrire des tests d'une manière naturelle, puis de les exécuter avec un dérouleur de tests (test-runner).
SpecFlow est construit sur la base de la syntaxe Gherkin, une grammaire conçue pour écrire des spécifications de comportement.



Qu'est-ce que SpecFlow?

Utilisez SpecFlow pour définir, gérer et exécuter automatiquement des tests d'acceptation lisibles par l'homme dans les projets .NET. La rédaction de tests faciles à comprendre est la pierre angulaire du paradigme BDD et permet également de constituer une documentation vivante de votre système.



SpecFlow en trois étapes faciles

ÉTAPE 1

Spécifier

Décrivez le comportement de votre système en utilisant une syntaxe lisible par l'homme. Définissez les spécifications dans le domaine problématique en utilisant le langage de vos parties prenantes et créez une documentation vivante de votre système.

ÉTAPE 2

Automatiser

Liez vos spécifications de test au code de votre application pour automatiser les tests de votre système. Assurez-vous que tous vos tests réussissent!

SpecFlow en trois étapes faciles

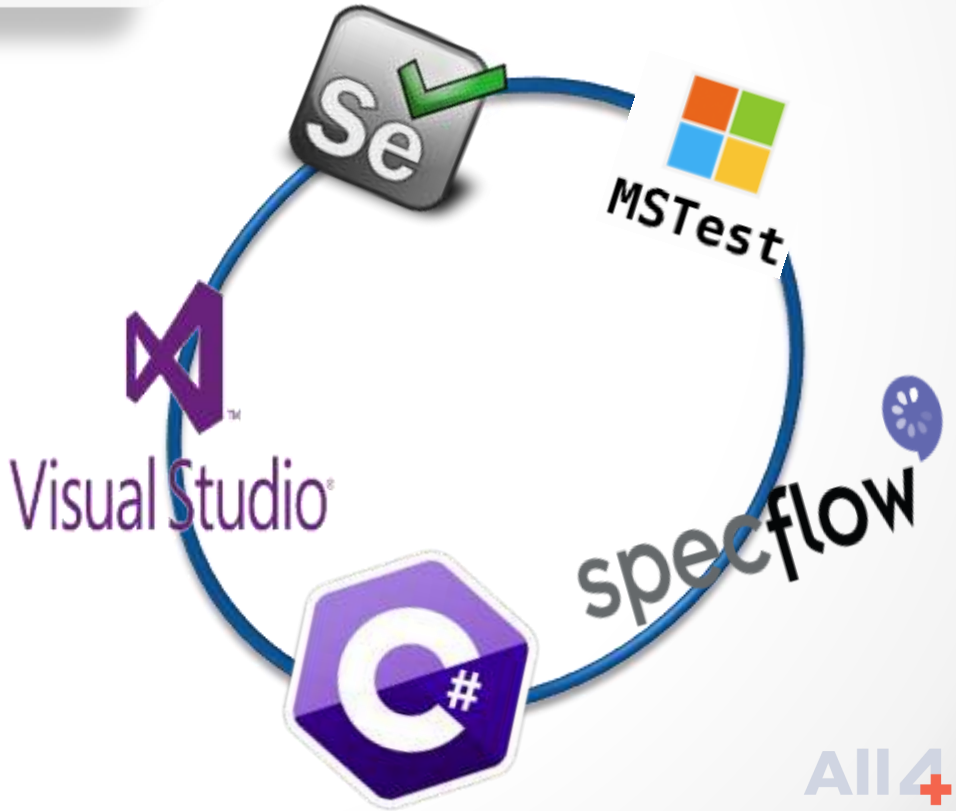
ÉTAPE 3

Prendre plaisir!

Détendez-vous en sachant que SpecFlow identifiera automatiquement les changements de rupture couverts par vos tests. Réduisez le développement médico-légal et profitez de la tranquillité d'esprit de savoir exactement ce que fait et est censé faire votre logiciel, même des mois plus tard.

Environnement

- Selenium Webdriver
- SpecFlow
- MSTest
- C#
- Visual Studio



Technologies

Design Pattern

- POM
- Factory Navigator
- Features

Technos

- Sélénium webdriver
- ATATA
- Specflow (BDD)

Reporting

- ExtentReport

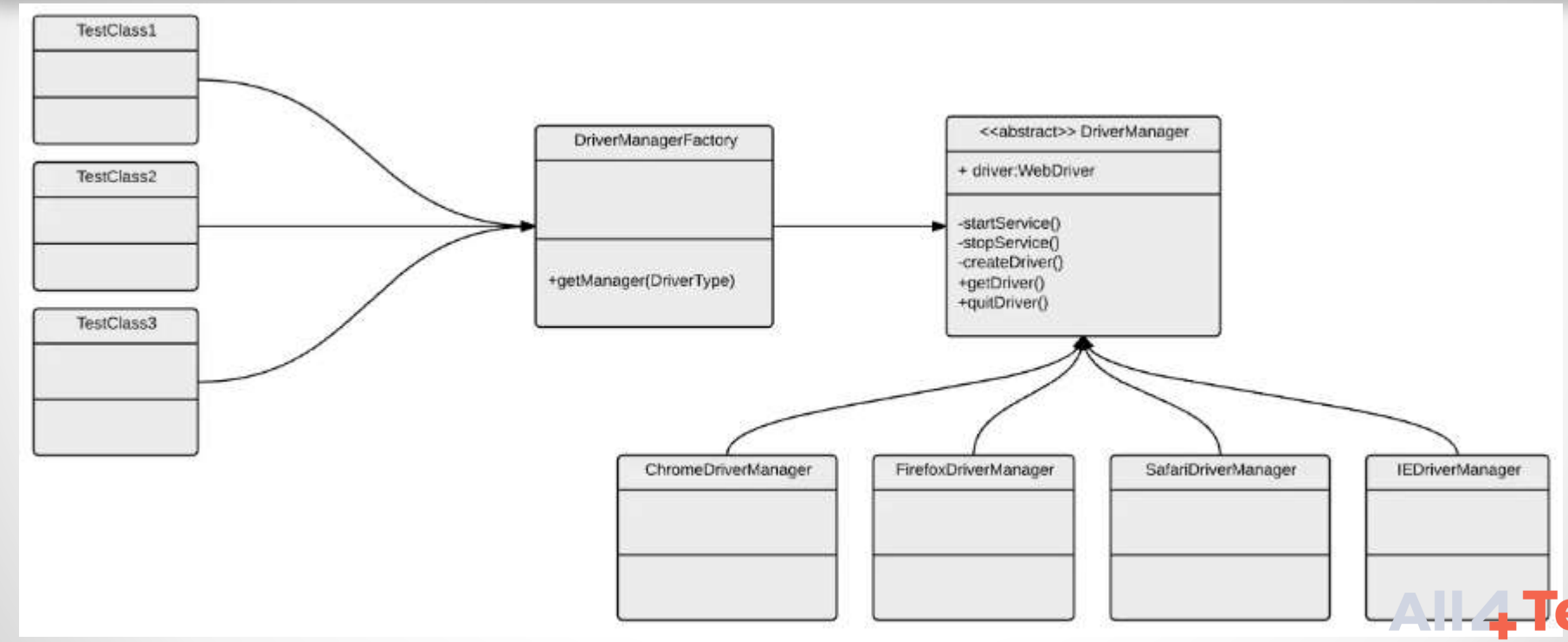
Assertion

- FluentAssertions

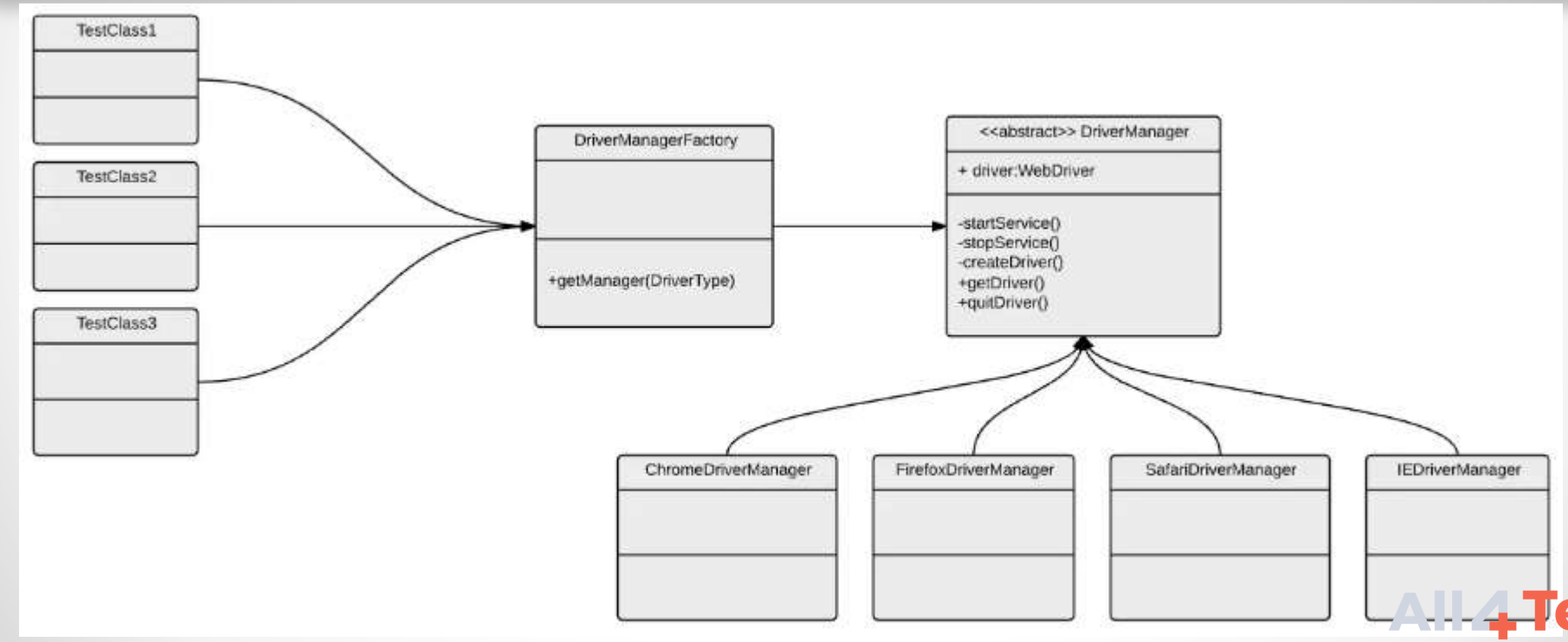
Factory Pattern pour les navigateurs

Vu que la maintenance et la transmission d'un objet ***WebDriver à*** travers différents tests est un processus délicat. En outre, la complexité augmente lorsque nous devons maintenir une seule instance d'un pilote *Web* pendant toute la durée du test.

- Pour surmonter le problème sur l'instanciation de *WebDriver* et maintenir l'instance de navigateur .



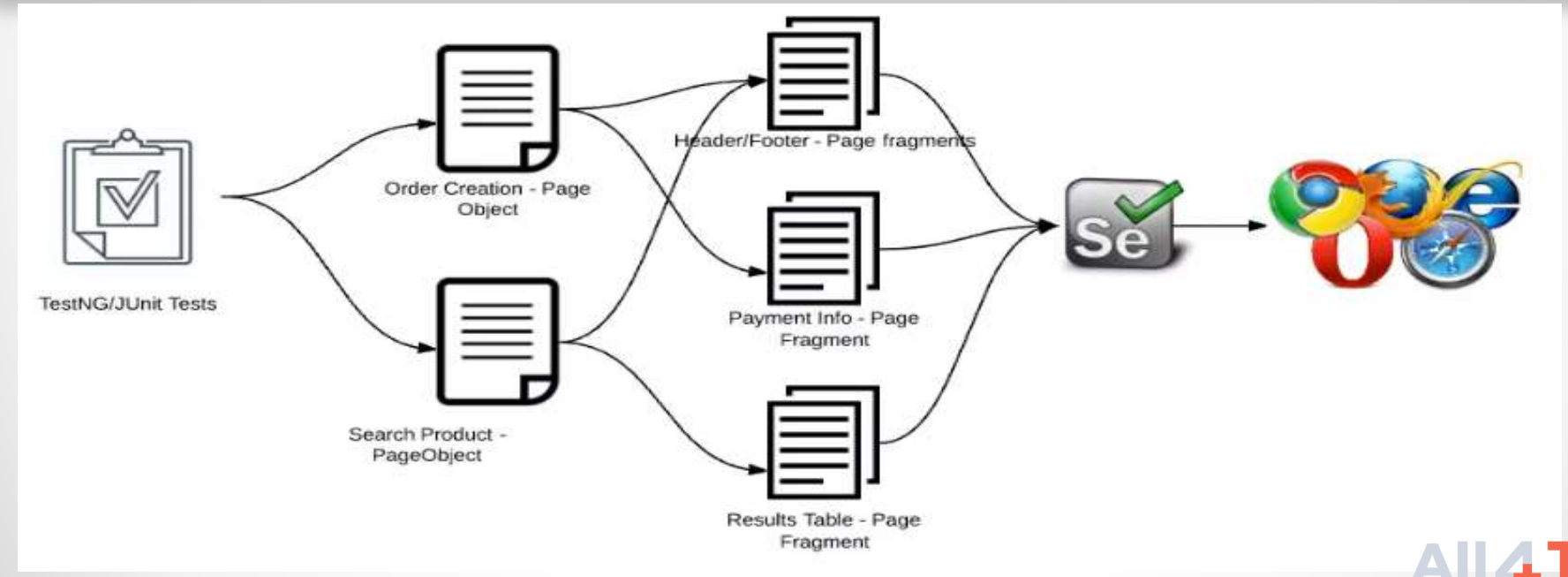
En utilisant Factory Pattern, nous cachons complètement la logique de création des instances de navigateur / service aux classes de test. Si nous obtenons une nouvelle exigence pour ajouter un nouveau navigateur, disons **PhantomJS** , cela ne devrait pas être une grosse affaire. Nous avons juste besoin de créer un **PhantomJSDriverManager** qui étend **DriverManager**



Page Factory

Page Factory est un concept intégré de Page Object Model pour Selenium WebDriver mais il est très optimisé.

Une meilleure approche de la maintenance de script consiste à créer un fichier de classe distinct qui trouverait des éléments Web, les remplirait ou les vérifierait. Cette classe peut être réutilisée dans tous les scripts utilisant cet élément. À l'avenir, s'il y a un changement dans l'élément web, nous devons faire la modification dans seulement 1 fichier de classe .



j'ai suivi le concept de séparation de Page Object Repository et des méthodes de test. De plus, avec l'aide de la classe Page Factory, j'ai utilisé les annotations **@FindBy** pour trouver WebElement et la méthode initElements pour initialiser les éléments web.

@FindBy peut accepter **tagName, partialLinkText, name, linkText, id, css, className, xpath** comme attributs.

```
[FindsBy(How = How.Id, Using = "username")]
private IWebElement UserName { get; set; }
```

Ceci est utilisé pour marquer un champ sur un objet Page pour indiquer un mécanisme alternatif pour localiser un élément. Utilisé conjointement avec PageFactory, cela permet aux utilisateurs de créer rapidement et facilement des objets Page.

SpecFlow Hooks

Vus qu'on rencontre des situations dans lesquelles on doit effectuer les étapes préalables avant de tester un scénario de test. Cette condition préalable peut être quelque chose de:

- *Démarrer un webdriver*
- *Configuration de connexions DB*
- *Configuration des données de test*

De la même manière, il y a toujours des étapes après les tests comme:

- *Tuer le webdriver*
- *Effacer les données de test*
- *Effacer les cookies du navigateur*
- *Déconnexion de l'application*
- *Impression de rapports ou de journaux*
- *Prendre des captures d'écran sur l'erreur*

Pour gérer ce genre de situations, les Hooks de SpecFlow sont le meilleur choix à utiliser en utilisant les méthodes `[BeforeScenario]` et [AfterScenario].

Setup.cs | AuthentificationPOMSteps.cs | AuthentificationPage.cs | AuthentificationPOM.feature

AlissiaE2ETest | AlissiaE2ETest.Utils.Setup | CallBrowser()

```
11  {
12      [Binding]
        1 référence
13      public class Setup: AbstractPage
14      {
15          [BeforeScenario]
16
            0 références
17          public void CallBrowser()
18          {
19              BrowserFactory.InitBrowser("Chrome");
20              var driver = BrowserFactory.Driver;
21              driver.Manage().Window.Maximize();
22          }
23
24          [AfterScenario]
25
            0 références
26          public void TearDown()
27          {
28              if (ScenarioContext.Current.TestError != null)
29              {
30                  String ScreeanshotFolderPath = System.Configuration.ConfigurationManager.AppSettings["FolderScreeanshotPath"];
31
32                  Screenshot ss = ((ITakesScreenshot)driver).GetScreenshot();
33                  ss.SaveAsFile(@ScreeanshotFolderPath + "Alissia_Screenshot.Png", OpenQA.Selenium.ScreenshotImageFormat.Png);
34
35              }
36
37              driver.Dispose();
38          }
39      }
40  }
41
```

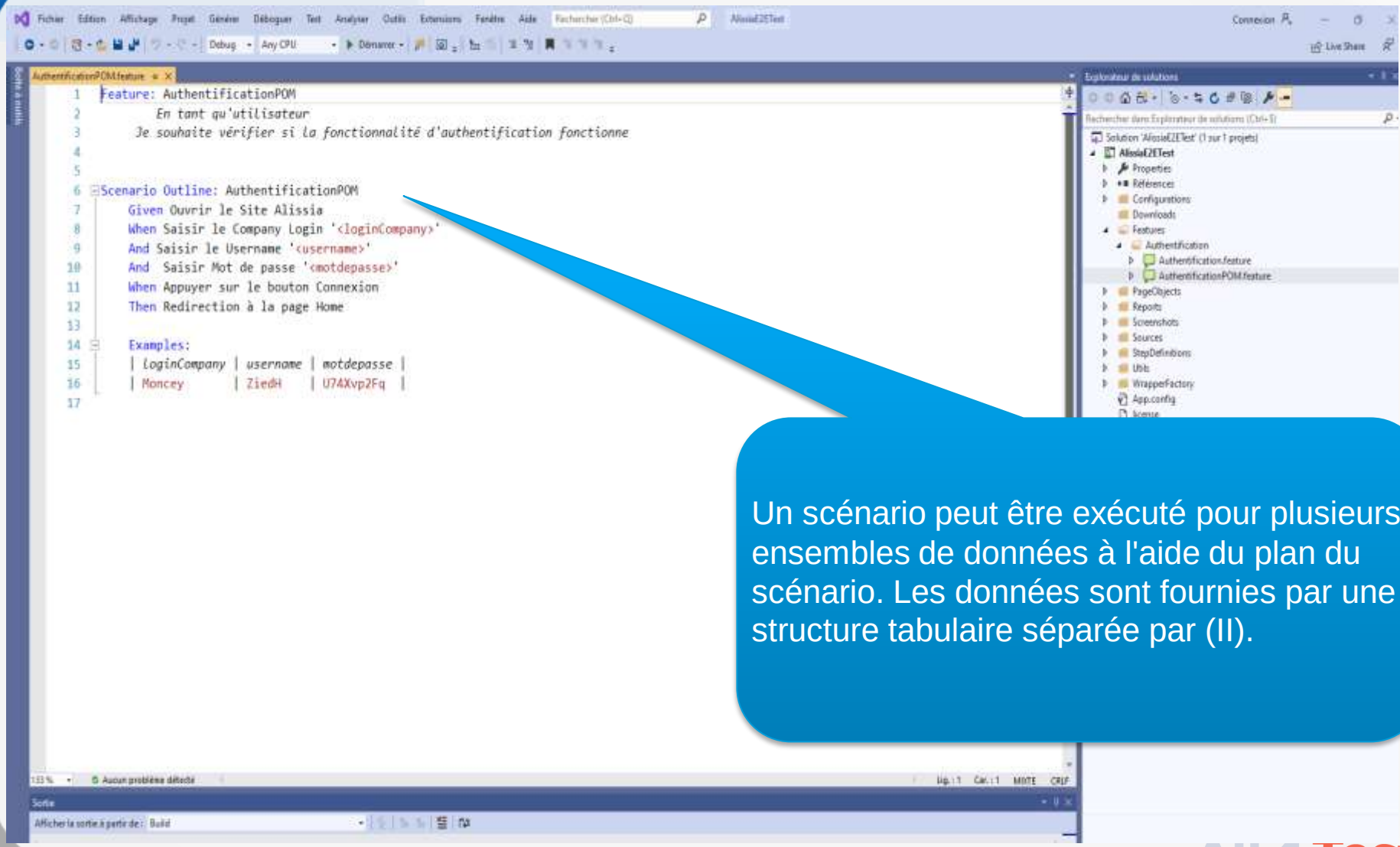
[BeforeScenario]

Ouverture du Navigateur

[AfterScenario]
Captur d'écran
Fermeture du Navigateur

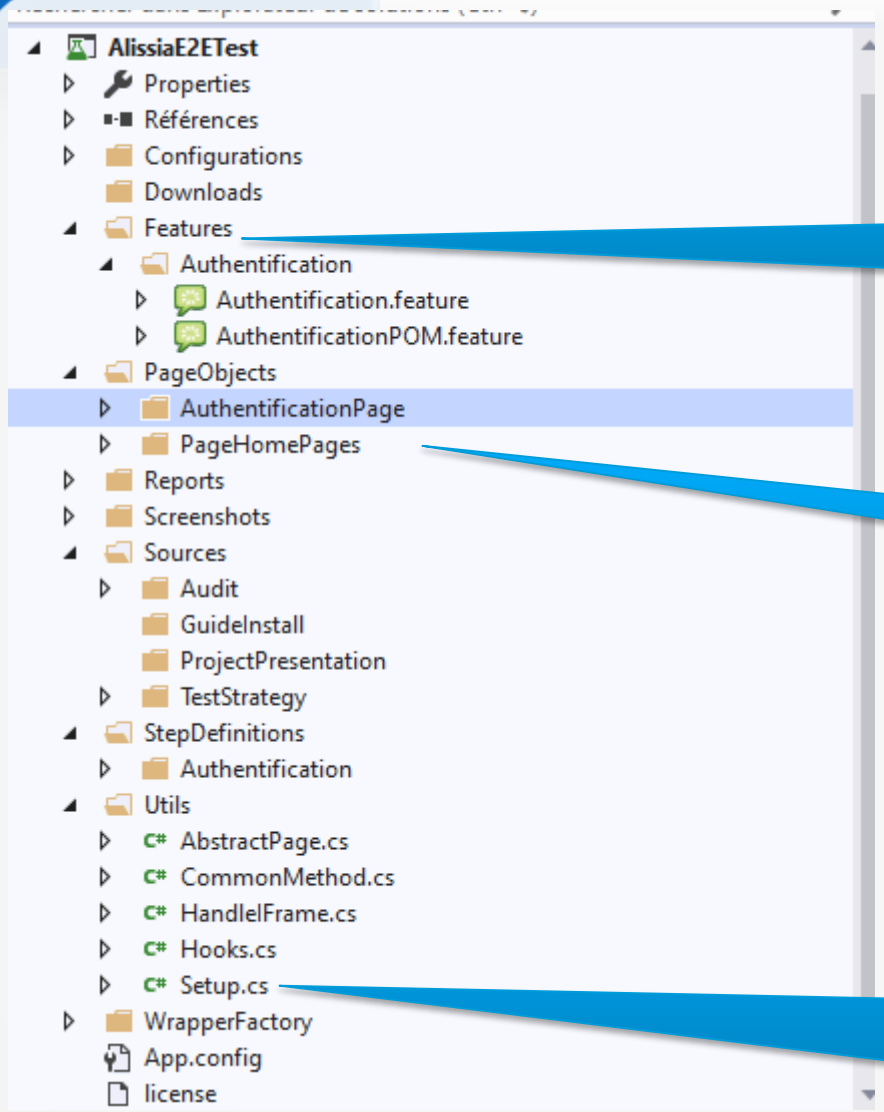
All4Test

Feature (Outline)



Un scénario peut être exécuté pour plusieurs ensembles de données à l'aide du plan du scénario. Les données sont fournies par une structure tabulaire séparée par (II).

Architecture



Dossier pour la création des Features

Page Object Model POM

@Before
Call Browser
@After
Screenshot

All4Test

Merci pour votre attention