Gherkin Basics

Maintenant que vous avez acquis une certaine confiance dans la façon dont fonctionne Cucember. Nous allons regarder Gherkin, le langage que nous utilisons pour écrire les fonctionnalités de Cucumber. À la fin de ce chapitre, vous allez comprendre comment écrire des spécifications pour votre logiciel qui peut être à la fois lu par vos intervenants et testé par  
Cucumber. Vous apprendrez ce que chacun des mots clés de Gherkin fait et comment  
ils s'inscrivent tous ensemble pour être lisibles, comme des spécifications exécutables de Cucumber. Vous ne saurez pas encore comment faire fonctionner les tests, mais vous serez prêt à les écrire.

3.1 Qu'est-ce Gherkin ?  
Nous construisons des logiciels pour les personnes (appelons-les parties prenantes), il est notoirement difficile de savoir exactement ce qu'ils veulent que nous construisons. Dans son célèbre essai, ‘pas de solution miracle, Fred Brooks’ dit: "La partie la plus difficile pour construire un système logiciel est de décider précisément ce qu'il faut construire".  
Nous avons tous travaillés sur des projets où, en raison d'un malentendu, le code  
sur lequel nous avions travaillé dur pendant plusieurs jours à dû être jeté.  
Une meilleure communication entre les développeurs et les parties prenantes est essentielle pour aider à éviter ce genre de perte de temps. Une technique qui aide réellement à faciliter   
cette communication est l'utilisation d'exemples concrets pour illustrer ce que nous voulons  
que le logiciel fasse.

**Exemples concrets :**

Nous voulons construire le comportement désiré du système en utilisant des exemples du monde réel, nous restons les pieds sur terre dans le langage et la terminologie qu’utilise nos partenaires: nous parlons leur langue. Quand nous parlons des termes utilisés dans les exemples, ils doivent vraiment s'imaginer utiliser le système.  
Discuter : signifie qu'ils peuvent commencer à nous donner des commentaires et suggestions utiles, avant qu’une seule ligne de code ne soit écrite.  
Pour illustrer cela, imaginons que vous deviez vous charger de la construction d'un système de paiement par carte de crédit.  
L'une des exigences est de s'assurer que les utilisateurs ne peuvent pas entrer des données erronées.

Voici une façon d'exprimer ce qui suit:

*Les clients devraient être empêchés d'entrer des informations invalides de leur carte de crédit.*

Ceci est un exemple de ce que les équipes agiles font souvent, ce sont des appels à des critères d'acceptation ou des conditions de satisfaction.

Nous utilisons le mot acceptation parce qu'ils nous disent ce que le système doit être capable de faire pour que nos parties prenantes trouvent cette solution acceptable.  
La déclaration des exigences précédentes est utile, mais elle laisse beaucoup trop  
de place à l'ambiguïté et à l'incompréhension. Elle manque de précision.

Qu'est-ce qui fait exactement qu’un ensemble de détails est invalide ? Comment l'utilisateur doit être empêché d’y pénétrer?

Essayons d’illustrant cette exigence avec un exemple concret:  
Si un client entre un numéro de carte de crédit qui n'ai pas exactement 16 chiffres, quEt  
il essayera de soumettre le formulaire, un message d'erreur doit l’avertir du nombre correct de chiffres à saisir.

Vous voyez comment cette deuxième déclaration est beaucoup plus spécifique? En tant que développeur nous devons être en mesure de commencer à travailler sur le code. En tant qu'acteur, nous avons une idée bien plus claire de ce que le développeur va construire.  
En fait, un acteur lisant cela pourrait souligner qu'il y a certains types de carte qui sont valides à moins de 16 chiffres et nous donnerons un autre exemple.  
C'est le réel pouvoir des exemples:

Ils stimulent notre imagination, ce qui nous permet d’explorer et découvrir des cas plus particulier, que nous n’aurions peut-être trouvé beaucoup plus tard.  
En donnant un exemple pour illustrer notre exigence, nous avons tourné une **acceptation  
critère** dans un test d'acceptation. Maintenant, nous avons quelque chose sans ambiguïté que  
l’on peut utiliser pour tester le comportement du système, que ce soit manuellement ou à l'aide d'un script de test automatisé.  
  
**Spécifications exécutables**  
Un autre avantage d'utiliser des exemples concrets, c’est qu'ils sont beaucoup plus faciles à  
valider pour le système de fonctionnement que les énoncés avec des exigences vagues.

De fait, si nous sommes claires et bien organisés sur la façon dont nous les exprimons, nous pouvons obtenir que l'ordinateur les vérifier pour nous.   
  
Gherkin nous donne une structure légère pour documenter des exemples des comportements que désirent nos parties prenantes.

De manière à ce qu'ils puissent être facilement compris à la fois par les parties prenantes et par Cucumber.

Bien que Gherkin s’apparente à un langage de programmation, son but essentiel est la lisibilité humaine, ce qui signifie que vous pouvez écrire des tests automatisés qui se lisent comme une documentation.

**Voici un exemple:**  
Feature : Commentaires lors de la saisie non valides des détails d’une carte de crédit

Dans des tests utilisateurs, nous avons vu beaucoup de gens qui ont fait des erreurs en entrant leur carte de crédit. Nous éviter de perdre des utilisateurs à ce stade crucial de la transaction.

Contexte :

Given J'ai choisi quelques objets à acheter  
Et Je suis sur le point d'entrer les détails de ma carte de crédit

Scenario: Numéro de carte de crédit trop court  
 Quand J'entre un numéro de carte qui est à seulement 15 chiffres  
 Et Tous les autres détails sont corrects  
 Et Je soumets le formulaire  
 Ensuite le formulaire doit être réaffichée

Et Je dois voir un message m'avisant du nombre correct de chiffres à respecter

Scenario: Date d'expiration invalide

Quand J'entre une date d'expiration de carte qui est dépassée   
 Et Tous les autres détails sont corrects  
 Et Je soumets le formulaire  
 Ensuite Le formulaire doit être réaffichée  
 Et Je verrais un message me disant la date d'expiration doit être mal

3.2 Format et syntaxe  
Gherkin utilisent une extension de fichier simple. Ils sont enregistrés en tant que texte brut, ce qui signifie qu'ils peuvent être lus et modifiés avec des outils basique.   
Un fichier Gherkin est donné par sa structure et sa signification en utilisant un ensemble de mots-clés spécifiques.  
Il existe un ensemble équivalent de ces mots clés dans chacun des supportée  
langues parlées, mais pour l'instant, nous allons jeter un oeil à ceux français:  
• Fonction  
• Contexte  
• Scenario  
• Compte tenu de  
• When  
• Ensuite  
• Et  
• Mais  
• Exemples  
Nous passerons le reste de ce chapitre à explorer comment utiliser les plus communs  
de ces mots-clés, ce qui sera suffisant pour vous aider à commencer à écrire votre propre  
  
**Être à sec**  
Tous les exemples de ce chapitre sont valables Gherkin et peuvent être analysés par  
Cucumber. Si vous souhaitait les essayer pendant que nous avançons dans le chapitre, il suffit de créer un fichier test. vedette.  
 Ensuite, exécutez-le avec le texte suivant:  
 $ cucumber test.feature --dry-run  
Le commutateur - dry-run communique avec Cucumber afin d’analyser le fichier sans l'exécuter. Il vous dira si votre Gherkin n'est pas valide.

3.3 Feature  
Chaque fichier Gherkin commence avec le mot-clé Feature. Ce mot-clé n’influe pas sur le comportement de vos tests. Il vous donne juste une bonne pratique  
à mettre à appliquer pour avoir une documentation de synthèse sur le groupe de tests  
suivre.  
Vous pouvez inclure n'importe quel texte dans la description, sauf une ligne commençant par un les mots Scenario, Fond, ou Scenario aperçu. La description peut s'étendre  
sur plusieurs lignes.   
  
Dans un fichier Gherkin valide, une entité doit être suivie par l'un des suivants:  
• Scenario  
• Fond  
• Scenario aperçu  
  
3.4 Scenario  
Chaque Feature contient plusieurs *scenarios*.  
Chaque *Scenario* est un exemple concret de la façon dont le système doit se comporter dans une situation particulière. Si vous additionnez le comportement défini par tous les scenarios, vous aurez le comportement attendu de la fonction.  
When Cucumber exécute un scenario, si le système se comporte comme décrit dans le  
scenario, le scenario va passer sinon, ce sera un échec.   
Chaque Feature a généralement entre cinq et vingt Scenario, chacune décrivant différents exemples de la façon dont cette Feature doit se comporter dans différentes circonstances.   
  
**Given, Quand, Then**  
En Gherkin, nous utilisons les mots clés Given, Quand et Then pour identifier les trois parties du scenario:  
  
Scenario: retrait réussi d'un compte dans le crédit  
 Given que j'ai 100 $ de mon compte # le contexte  
 When je demande $ 20 # l'événement (s)  
 Then $ 20 devraient être distribuer # résultat   
  
Ainsi, nous utilisons *Given* pour mettre en place le contexte où le scenario se produit, When  
interagir avec le système, et *Then* permet de vérifier que le résultat de cette interaction était ce que nous attendions.  
  
**And, But**  
Chacune des lignes dans un scenario est considérer comme une étape. Nous pouvons ajouter des étapes pour chaque section du scenario Given, Quand, ou Then en utilisant les mots-clés And et But  
  
 Scenario: Tentative de retrait en utilisant une carte volée  
 Given que j'ai 100 $ sur mon compte  
 But ma carte n'est pas valide  
 When je demande $ 50  
 Then, ma carte ne doit pas être retournée  
 And Il me sera demandé de contacter ma banque

Cucumber ne se soucie des mots-clés que vous utilisez; le choix est simplement là pour vous aider à créer le scenario le plus lisible possible. Si vous ne le faites pas et souhaitez utiliser And et But, vous pourriez écrire le scenario précédent de cette manière, il fonctionnera toujours de la même façon:

Scenario: Tentative de retrait en utilisant carte volée  
 Given que j'ai 100 $ de mon compte  
 Given ma carte n'est pas valide  
 When je demande $ 50  
 Then, ma carte ne doit pas être retournée  
 Then Il me sera demandé de contacter ma banque  
  
**Indépendance**   
Un concept très important que vous devez saisir lors de l'écriture de scenarios est que   
chaque scenario doit avoir un sens et pouvoir être exécutées indépendamment de tout autre scenario.  
Cela signifie que vous ne pouvez pas mettre de l'argent dans un compte dans un scenario et  
s'attendre à ce que l'argent soit présent dans un scenario différent.  
 Cucumber ne vous empêchera pas de le faire, mais c’est une très mauvaise pratique: vous vous retrouverez avec des scenarios avec des résultats inattendue et difficiles à comprendre.  
  
**3.5 Commentaires**  
Ainsi que les champs de description qui suivent Feature et Scenario,  
Cucumber vous permet de faire précéder vos mots-clés avec des commentaires.  
Comme dans Ruby, les commentaires commencent par le caractère #.

Voici un exemple:  
# Cette fonction couvre la transaction de compte et des modules conducteur matériel   
Feature: retirer de l'argent pour acheter de la bière  
En tant que titulaire de compte  
Je veux retirer de l'argent de l'ATM  
# Vous ne pouvez l’intégrer à l’interface  
Scenario: retirer trop de crédit d'un compte   
Given que j'ai 50 $ dans mon compte  
# When je lève ma baguette magique  
And je retire $ 100  
Then, je devrais recevoir $ 100  
  
**3.7 Qu'est-ce que nous venons d'apprendre**  
Passons en revue ce dont nous avons parlé dans ce chapitre:  
• Nous avons vu la description des principaux mots-clés de Gherkin Feature, Scenario, When, et Then ainsi que leurs comportements.  
• Il y a un model bien définit dans un scenario Gherkin, un contexte  
(Given), un événement (When), et un résultat (Then).  
  
• Chaque scenario doit pouvoir être exécuté de manière indépendante et ne doit pas  
dépendre de la mise en place par d'autres scenarios. Cela signifie que chaque scenario  
doit inclure toutes les données dont il a besoin et qu’il ne doit pas dépendre de l’état d’un autre scénario.   
• Vous pouvez ajouter des descriptions et des commentaires à vos fichiers .feature.

À ce stade, vous avez toutes les connaissances pour commencer à écrire votre propre caractéristiques Gherkin. Même si il y a certains mots-clés que nous n'avons pas encore découverts, nous avons découvert les bases fondamentales. Imaginez que vous avez une machine qui peut transformer vos scenarios de Cucumber parfaitement dans votre développement, et jouer le jeu de la collaboration avec votre équipe pour créer les meilleures descriptions de ce que votre système doit effectuer.

Dans le chapitre suivant, nous allons commencer à explorer les *step definition*, la couche sous les *Features Gherkin* où vous interagirez avec votre application, et rendant vos scenarios vivant.