Agilité : Behavioral Driven Development

TP7 Agilité

Behavioral Driven Development

Jérôme Buisine
jerome.buisine@univ-littoral.fr

29 novembre 2022

Durée : 5 heures

L'objectif de ce TP est de proposer de nouveau des validations de développements de l'application tel que cela aurait pu être fait par l'approche BDD (validations de comportements attendus de l'application).

1 Travail

Ce support propose d'intégrer l'outil « `behave` » pour amener des validations de comportements de l'application. De ce fait, vous allez dans ce TP vérifier différents scénarios en respectant des consignes spécifiques :

— 1. À partir d'une branche **`feature/UserStoryX`** spécifique à une section de développement, écrire pour chaque fonction ou méthode le contrat qui permet sa vérification.
— 2. Écrire un ou plusieurs scénario(s) attendu en langage Gherkin.
— 3. Ajouter un test comportement qui permet de vérifier un scénario.
— 5. Soumettre en ligne la modification apportée.

Pour vous aider : voici un ensemble de ressources qui peuvent vous être utiles pour ce TP :

— 1. Documentation officielle de l'outil `Git`.
— 2. Gitlab : interface web pour la gestion de projets versionnés sous `Git`.
— 3. behave : une bibliothèque Python pour la prise en compte du BDD.
— 4. selenium : pour exécuter un navigateur depuis Python.

Remarque importante : faites attention au copier/coller à partir du PDF qui peut prendre en compte des caractères inattendus et causer des erreurs.

2 Configuration de l'environnement

Vous allez pour ce TP repartir de votre version de projet `treevolution` où Cypress y avait été intégré.

2.1 Installation des dépendances

Pour la suite du TP, il sera nécessaire d'installer les dépendances suivantes :

```
pip install behave==1.2.6 selenium==4.6.0
```

2.2 Première utilisation de behave

Nous allons à ce stade configurer behave afin qu'il prenne en compte le navigateur Firefox pour chaque test de comportement (scénario). Pour cela, il faut vous ajouter le fichier « `features/environment.py` » avec le contenu suivant :

```
from selenium import webdriver

def before_all(context):
    context.browser = webdriver.Firefox()
    context.browser.set_page_load_timeout(10)
    context.browser.implicitly_wait(10)
    context.browser.maximize_window()

def after_all(context):
    context.browser.quit()
```

Vous pouvez définir un premier exemple de `feature` avec un scénario qui lui est propre. Pour cela, vous pouvez ajouter dans le fichier « `features/example.feature` » :

```
Feature: Search
    Scenario: Search PyPI
        Given I navigate to the PyPi Home page
        When I search for "behave"
        Then I find a link to project "behave"
        And If I click on project "behave", I find its version "1.2.6"
```

Il est maintenant possible d'associer un test automatisé de comportement à ce scénario (dans le fichier « `features/steps/example_steps.py` »), avec le contenu suivant :

```
from behave import given, when, then, step
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys

@given('I navigate to the PyPi Home page')
def step_impl(context):
    context.browser.get("https://pypi.org")
    assert "PyPI" in context.browser.title

@when('I search for "{search_term}"')
def step_impl(context, search_term):
    context.browser.find_element(By.ID, "search").send_keys(search_term)
    context.browser.find_element(By.ID, "search").send_keys(Keys.ENTER)

@then('I find a link to project "{search_result}"')
def step_impl(context, search_result):
    selector = f'a[href="/project/{search_result}/"]'
    assert context.browser.find_element(By.CSS_SELECTOR, selector)
```

```python
@step('If I click on project "{search_result}", I find its version "{version}"')
def step_impl(context, search_result, version):
    selector = f'a[href="/project/{search_result}/"]'
    context.browser.find_element(By.CSS_SELECTOR, selector).click()
    behave_version = context.browser.find_element(By.TAG_NAME, "h1").text.strip()
    assert behave_version == f"behave {version}"
```

3 Développements

Pour l'ajout des tests de comportements automatisés, nous allons donc utiliser `selenium`. Tout comme `Cypress`, il permet d'accéder aux éléments du DOM à partir des méthodes : `find_element` et `find_elements`. Il existe plusieurs manières d'accéder aux éléments depuis le module « `By` » : voir API Selenium.

Pour rappel, l'application web `treevolution` peut-être lancée à partir de la commande :

```
python wsgi.py
```

Note : dans le cadre de ce TP et afin de se focaliser sur l'outil, les développements sont déjà réalisés et leurs comportements sont à vérifier uniquement. Il faut bien garder à l'esprit qu'en temps normal, tout comme la méthode TDD, la méthode BDD implique l'écriture du comportement et du test puis d'effectuer le développement.

À noter que behave s'exécute comme suit :

```
# exécute toutes les features
behave

# exécute une feature spécifique
behave -i features/example.feature
```

3.1 Page d'accueil

Travail : créer un fichier `home.feature` et `home_steps.py` associé.

Scénario 1 : La page principale de l'application doit avoir un titre composé du texte « *Welcome to the Treevolution simulator* ».

3.2 Mis à jour de la configuration

Travail : créer un fichier `config.feature` et `config_steps.py` associé.

Scénario 2 : En accédant à l'onglet `/config` de l'application. Vérifier que les champs nombres de jours et nombres d'arbres indiquent correctement leurs valeurs, si celles-ci sont modifiées.

Indications :

- L'API selenium ne permet pas une interaction directe avec un input de type `range`. Il faudra pour modifier la valeur du champs, exécuter du javascript à l'aide de la méthode `execute_script`.

- Il est possible de forcer l'événement de changement de valeur en Javascript :

```
element = document.getElementById("elementId");
element.dispatchEvent(new Event("change"));
```

- Il peut-être nécessaire de retourner sur la page principale de l'application pour réinitialiser le contexte avant d'accéder à la configuration.

Scénario 3 : En accédant à l'onglet `/config` de l'application. Vérifiez qu'il n'est pas possible de valider une configuration ayant une date de fin de simulation inférieure à celle de début. Un message d'erreur spécifique peut-être vérifié.

Indications :

- Faire attention au format de date attendu par le champs pour que celle-ci soit valide.
- L'API selenium propose la gestion d'un input de type select.

Scénario 4 : En accédant à l'onglet `/config` de l'application. Vérifiez qu'il est possible de valider une configuration avec des dates valides et en ayant sélectionné un type d'arbre. Un message spécifique est attendu.

Indications :

- L'API selenium ne permet pas une interaction directe avec un input de type `date`. Il faudra pour modifier la valeur du champs, exécuter du javascript à l'aide de la méthode `execute_script`.
- Il faudra toujours faire attention au format attendu par le champs de type date.

3.3 Fonctionnement de la simulation

Travail : créer un fichier `simulation.feature` et `simulation_steps.py` associé.

Histoire 5 : À partir d'une configuration valide, il est possible de créer une nouvelle simulation depuis l'onglet `/simulation`. Notamment :

— La date de début correspond bien à celle attendue.
— Le nombre d'arbres vivants également.
— Le nombre d'arbres en humus doit être de 0.
— Le nombre de seeds doit être de 0.

Indications :

- Depuis selenium, il est possible d'accéder à l'élément parent à partir du `XPATH` :

```
element.find_element(By.XPATH, '..')
```

Histoire 6 : À partir d'une configuration valide et d'une nouvelle simulation créée. Le nombre d'espèces d'arbres sélectionné est le même que la configuration.

Histoire 7 : Lorsqu'une simulation est exécutée, la date et le nombre d'arbres ont évolué après un certain délai.

Indications :

- Il simplement possible de faire patienter le programme à l'aide de la librairie `time` de Python.

Histoire 8 : Lorsqu'une simulation est exécutée, il est possible de l'arrêter et de la réinitialiser.

Histoire 9 : Il est possible d'atteindre la fin d'une simulation. La date et le nombre d'arbres sont restés fixes après la fin de cette simulation. Il n'est plus possible de continuer la simulation, mais on peut en créer une nouvelle à partir de la même configuration.

Indications :

- L'API selenium permet d'attendre jusqu'à obtenir un élément souhaité : wait support.

4 Remise des travaux

N'oubliez pas de réaliser un dernier commit de vos travaux. Taguez ensuite votre projet avec le tag « `tp7` » et soumettez-le sur le serveur Gitlab. Cette version de projet fera office de rendu.