Agilité : End to End test process

TP6 Agilité

Toward an end-to-end testing process during project development

Jérôme Buisine
jerome.buisine@univ-littoral.fr

6 novembre 2021

Durée : 6 heures

L'objectif de ce TP est de vérifier le bon fonctionnement d'une application web. Cette interface graphique doit permettre à la fois de régler des paramètres de simulation de notre réseau routier mais également d'afficher les résultats obtenus de la simulation de ce réseau. Nous allons vérifier, étape par étape, que l'interface graphique fonctionne comme attendue en utilisant le framework Cypress.

1 Travail

Vous allez partir d'un projet initial, dans lequel se trouve un **jar** exécutable comprenant l'API REST d'intéraction avec le réseau routier et l'interface web proposée. Nous allons tester l'ensemble des fonctionnalités de cette interface au fur et à mesure de l'avancement du TP.

Voici l'ensemble des outils / langages qui seront utilisés pour ce TP :

— 1. IntelliJ/Idea (version Ultimate) comme environnement de développement ;
— 2. git comme système de versionning du projet ;
— 3. Gitlab comme serveur d'hébergement de votre projet git (permettant l'accès à des outils d'intégration, livraison, deploiement continues) ;
— 4. Java pour exécuter le serveur d'API REST ;
— 5. Node.js pour lancer l'application web et ses dépendances. Il sera donc nécessaire de l'**installer** (version LTS de préférence) si ce n'est pas déjà le cas ;
— 5. Le framework Cypress permettant de simuler des interactions avec un site web directement et vérifiant le bon fonctionnement ;

2 Initialisation du projet

Un projet git comprenant l'ensemble des éléments (le simulateur et son interface web) pour le sujet, est disponible à l'adresse suivante : TP6-Agility. Vous pouvez forker et ajouter ce projet comme dépendance (sous-module) de votre projet actuel :

```
git submodule add https://gitlab.com/XXXX/XXXX web
```

2.1 Installation du projet

En suivant la procédure du fichier **README.md** du projet sous-module, procéder à l'installation de l'application (sans lancer Cypress pour le moment) puis visualiser l'interface web.

2.2 L'interface web

L'interface web est composée de plusieurs onglets :
— **Simulation** : permettant de lancer une simulation et d'avoir des informations temps réelles de celle-ci.
— **Configuration** : qui permet de modifier les paramètres du réseaux routiers et ajouter des véhicules.
— **Overview** : qui offre un visuel des éléments du réseau routier, et notamment la localisation des différents segments.

Les fonctionnalités et spécificités de cette interface web sont les suivantes :
— Les données modifiées relatives aux segments, feux et rond-points, ne sont pas réinitialisées tant que le serveur d'API Java n'est pas redémarré.
— Un véhicule est ajouté par un segment d'entrée et de sortie du réseau. Comme par exemple le segment 5 et le segment 26 (voir schéma onglet **Overview**, S5 et S26).
— Une fois une simulation lancée, il est nécessaire de recharger la page pour en lancer une nouvelle.
— Le fait de rafraîchir la page entraîne la suppression des véhicules existants (nouveau contexte de simulation). Comme indiqué précédemment, les données relatives à la structure du réseau, quant à elles, ne changent pas.

Note : il est important de souligner qu'une fois le serveur de l'API REST lancé, il est accessible à l'adresse suivante http://127.0.0.1:4567. La route '`/elements`' de l'API vous permettra de récupérer la structure du réseau routier courant et ses paramètres. Vous pouvez tester de contacter cette route depuis votre navigateur pour accéder au contenu une fois le système initialisé depuis l'application web.

3 Utilisation de Cypress

3.1 Introduction

Cypress est un framework permettant de vérifier que la structure d'une page web (le DOM pour Document Object Model), est bien comme attendue après un scénario d'actions réalisées. On cherche ici à simuler les intéractions utilisateurs sur une plateforme web. Il est également possible d'exécuter des requêtes classiques via les méthodes du protocole HTTP.

Le dossier '`web-app/cypress`' du projet sous-module est propre à l'utilisation de Cypress. On y trouve notamment le fichier '`web-app/cypress/integration/sample_spec.js`' dans lequel l'ensemble des tests Cypress seront codés. On trouve également dans ce dossier des vidéos des tests exécutés par Cypress ou encore, des screenshots d'erreurs rencontrées.

Il est possible d'exécuter Cypress depuis le dossier '`web-app`' :

```
./node_modules/.bin/cypress run
```

Ou d'ouvrir une interface permettant le visuel des différents tests effectués par Cypress :

```
./node_modules/.bin/cypress open
```

Les tests Cypress sont déjà intégrés au projet Gitlab. Vous pouvez visualiser le fichier de configuration '`.gitlab-ci.yml`' du sous-module pour comprendre son fonctionnement. De cette manière, vous pouvez éditer votre fichier de configuration CI/CD pour y intégrer les tests Cypress. Il faut bien faire attention ici à la cohérence des accès de chaque élément (certains stages déjà développés dans les TPs précédents ne seront plus forcément nécessaires).

Note importante : Pour chaque test, vous allez pouvoir vérifier le fonctionement de celui-ci à partir de la CI/CD du projet web. Lorsque vos tests seront terminés, il vous faudra mettre à jour le sous-module dans le projet principal et y ajouter des modifications pour actionner la CI/CD pour prise en compte des tests Cypress.

Pour cela, il vous faudra ajouter au début du stage `cypress` dans le projet principal, l'instruction ci-dessous permettra d'initialiser votre sous-module côté CI/CD.

```
git submodule update --init --recursive
```

3.2 Exemples

Vous trouverez ci-dessous, quelques exemples pratiques de commandes que Cypress peut permettre lors de l'exécution de tests :

```
// 1. Effectue une action de clic sur un élément sélectionné dans le DOM
cy.get('a[href="#configuration"]').click()

// 2. Récupération puis saisie d'une valeur dans un champs de formulaire
cy.get('input[name="speed"][form="segment-5"]').clear().type('30')

// 3. Effectue une requête sur un élément extérieur au site (ici API) puis
    vérification du résultat
cy.request('GET', 'http://127.0.0.1:4567/elements').as('elements')

// utilisation de l'alias 'elements'
cy.get('@elements').should((response) => {
        expect(response.body['segments'][4]).to.have.property('speed', 10)
})

// 4. Récupère un formulaire puis réalise des actions sur des éléments enfants
cy.get('form[id="form-1"]').within(() => {
        cy.get('input[type="text"]').type('test')
        cy.get('button').click()
})
```

Chaque test devra être composé de sa description et de ses actions :

```
// Test 1 d'exemple par défaut :
describe('The Traffic web site home page', () => {
    it('successfully loads', () => {
        cy.visit('/')
    })
})
```

3.3 Première partie : développment de tests

En vous basant sur la documentation et les précisions apportées, implémenter les tests énumérés ci-dessous dans le fichier '`web-app/cypress/integration/sample_spec.js`' à la suite du premier test. Après avoir vérifié que votre test fonctionne sur votre propre machine, et ce pour chacun des tests, réaliser un commit et mettre à jour le projet Gitlab. Vérifier que la pipeline et le job 'cypress' fonctionne correctement.

3.3.1 Scénario 1

En cliquant sur l'onglet ‘`Configuration`‘ du menu, vérifier que l'on arrive bien sur la page de configuration.

3.3.2 Scénario 2

Vérifier sur la page configuration que notre réseau routier courant comprend bien 28 segments.

3.3.3 Scénario 3

Vérifier que le nombre de véhicules est pour le moment de 0.

3.3.4 Scénario 4

Modifier la valeur de vitesse du segment 5 à 30 puis vérifier que la mise à jour a bien été réalisée sur le serveur de l'API en exploitant la route ‘`elements`‘.

Vérifier également qu'une fenêtre s'affiche pour informer l'utilisateur après modification et qu'elle se ferme correctement après avoir cliqué sur le bouton ‘`Close`‘.

Note : la route ‘`elements`‘ vous fournit des informations sur les paramètres du réseau routier. Vous pouvez visualiser sont contenu afin de vous en familiariser.

3.3.5 Scénario 5

Modifier le rond point pour qu'il puisse avoir une capacité de 4 et une durée de 15 secondes. Rafraîchir la page pour vérifier que les modifications sont bien répercutées côté client. Vérifier également la persistance de ces données depuis la route ‘`elements`‘.

3.3.6 Scénario 6

Modifier le feu d'identifiant 29 avec les valeurs suivantes :
— Orange duration : 4
— Green duration : 40
— Next passage duration : 8

Vérifier également la persistance de ces données depuis la route ‘`elements`‘.

3.3.7 Scénario 7

Ajouter 3 véhicules avec les chemins d'origine / destination, et temps de départ suivants :
— Origine : S5 / Destination : S26 / Départ : 50
— Origine : S19 / Destination : S8 / Départ : 200
— Origine : S27 / Destination : S2 / Départ : 150

Vérifier la cohérence des données véhicules côté serveur depuis la route ‘`vehicles`‘.

Vérifier que l'affichage des véhicules sont bien présents depuis l'interface de configuration et au nombre de 3.

3.3.8 Scénario 8

Vérifier sur l'interface de simulation que tous les véhicules sont bien à l'arrêts.

Réaliser ensuite une simulation de 120 secondes. Vérifier que la barre de progression est bien à 100% en fin de simulation.

Vérifier également que seul le premier véhicule est en mouvement en fin de simulation.

3.3.9 Scénario 9

Rafraîchir la page pour commencer une nouvelle simulation ou appeler la route du serveur API '`http://localhost:4567/init`' pour réinitialiser la simulation. Vérifier que la simulation a été réinitialisée et que les 3 véhicules ne sont plus présents.

Ajouter les 3 mêmes véhicules que pour le **scénario 7**.

Réaliser ensuite une simulation de 500 secondes.

Vérifier que chaque véhicule n'est plus en mouvement en fin de simulation

3.3.10 Scénario 10

Rafraîchir la page pour commencer une nouvelle simulation. Vérifier que la simulation a été réinitialisée et que les 3 véhicules ne sont plus présents.

Ajouter les 3 véhicules suivants :
— Origine : S5 / Destination : S26 / Départ : 50
— Origine : S5 / Destination : S26 / Départ : 80
— Origine : S5 / Destination : S26 / Départ : 80

Réaliser ensuite une simulation de 200 secondes.

Vérifier la position de chacun des véhicules en fin de simulation :
— Véhicule 1 : doit être sur le segment 17
— Véhicule 2 : doit être sur le segment 29
— Véhicule 3 : doit être sur le segment 29

3.3.11 Seconde partie : simplification des tests

En vous basant sur l'application React développée ('`web-app/src/index.js`'). Ajouter des idenfiants sur certains blocs pour faciliter la ré-écriture de vos différents scénarios.

Pour cela, vous allez créer un nouveau fichier de test à cet effet. L'idée est ici de montrer qu'il est important en amont de posséder une architecture DOM correcte et détaillée pour l'écriture de ce genre de tests. À noter qu'il faut que les tests précédents soient toujours fonctionnels.