LOG2440 – Méthod. de dévelop. et conc. d'applic. Web

Travail pratique 3

Chargés de laboratoire:

Minh-Tri Do

Gourdigou Junior Patrice Kolani

Hiver 2022

Département de génie informatique et génie logiciel

1 Objectifs

Le but de ce travail pratique est de vous familiariser avec les pratiques de tests logiciels, notamment les tests unitaires de code JavaScript ainsi que les tests d'intégrations dans un projet logiciel. Vous allez vous familiariser avec des outils de tests tels que la librairie de tests Jest ainsi que les concepts de *Mock*, *Stub* et *Spy*.

Plus particulièrement, vous aurez à mettre en place les tests qui permettent de vérifier et valider le code de la logique du site web fait dans le TP2. Vous aurez également à implémenter une nouvelle fonctionnalité et la tester.

2 Introduction

Lors du deuxième travail pratique, vous avez mis en place la logique nécessaire pour rendre votre site web dynamique. Votre site web est maintenant capable de charger dynamiquement l'information nécessaire à afficher ainsi qu'à gérer l'ajout et la sauvegarde de nouvelles recettes à travers un formulaire modulaire.

Cependant, outre que les tests *Nightwatch* fournis, il n'y a aucune validation ou vérification de votre travail. Plus spécifiquement, le code JS que vous avez écrit n'est pas vérifié de manière formelle. Un problème potentiel ne sera pas détecté que lorsque vous utilisez l'application.

3 Travail à réaliser

À partir du code nécessaire pour répondre aux requis du travail pratique précédent, vous aurez à implémenter les tests unitaires nécessaires pour avoir une couverture de code et branches complète du code source. Une version modifiée du TP2 vous sera fournie avec le dépôt de Git de départ afin de partir du bon pied pour ce travail. Notez que le comportement de cette version correspond au travail demandé au TP2, mais la structure interne n'est pas pareille : l'implémentation comporte plusieurs classes et fonctions avec des fonctionnalités précises. Les tests de bout en bout de *Nightwatch* sont également placés dans un projet différent du site web afin de bien diviser les 2 parties du travail.

Avertissement

Afin de promouvoir la compréhension du langage JavaScript et la librairie, seulement la librairie de tests **Jest** est permise pour ce travail pratique. Vous utiliserez également la syntaxe ES2015 et les modules JS qui sont supportés par Jest.

3.1 Modules ECMAScript (ESM) et serveur HTTP local

Au cours de ce travail, vous aurez à utiliser les modules ECMAScript. La particularité de ces modules est qu'il n'est pas possible de les charger en utilisant le schéma URI File, c'est-à-dire, vous ne pouvez pas simplement ouvrir le fichier **index.html** à partir de votre ordinateur. Les ESM doivent être servis par un serveur HTTP.

Dans votre cas, vous utiliserez *lite-server*, un serveur statique minimaliste qui vous permettra d'avoir un déploiement local d'un serveur local qui fournit les fichiers sources de votre travail. Pour lancer le serveur, vous pouvez utiliser la commande npm **start** dans votre terminal avec : **npm start**. Assurez-vous d'avoir installé les dépendances du projet avec la commande **npm ci** au préalable.

Par défaut, ce serveur est accessible à l'adresse : "http ://localhost :5000".

Note : le nom **localhost** est un raccourci pour l'adresse IP **127.0.0.1**. Le lancement du serveur *lite-server* affichera les adresses auxquelles le serveur est accessible.

3.2 Tests automatisés

Vous remarquerez que les tests de bout en bout pour chacune de vos pages vous sont fournis dans le répertoire `tests/e2e` du projet `nightwatch`. Ces tests vous permettront de valider votre travail et le contenu de vos pages HTML.

Pour lancer la suite de tests, vous n'avez qu'à aller à la racine du répertoire `nightwatch` avec un terminal et de lancer la commande **npm run e2e** afin de lancer les tests fournis.

Note : comme le site web utilise ESM il faut valider des pages HTML fournies par un vrai serveur. Il faut donc lancer le serveur avec la commande **npm start** avant de lancer les tests dans la répertoire *siteWeb*.

Les tests à compléter se trouvent dans le répertoire `tests/jest` du projet `siteWeb`. Notez que certains tests sont déjà faits pour vous afin de vous aider à démarrer le projet.

Pour lancer la suite des tests Jests, vous n'avez qu'à lancer la commande **npm test**. Vous pouvez également utiliser la commande **npm run coverage** qui lancera les tests unitaires et vous donnera un rapport de la couverture du code par la suite.

3.3 Éléments à réaliser

Maintenant que le site web est fonctionnel, la prochaine étape est d'implémenter les éléments nécessaires pour bien tester la logique d'implémentation. Avant de débuter, assurez-vous que le projet de départ est fonctionnel et vous êtes capables de naviguer à travers le site. Le code de départ implémente les fonctionnalités demandées au TP2.

Vous aurez également à implémenter une nouvelle fonctionnalité : Recherche par ingrédient. Vous mettrez en pratique les concepts de TDD *(Test Driven Development)* : les tests unitaires vous sont fournis et vous devez implémenter la logique correspondante.

3.4 Recherche par ingrédients

Cette fonctionnalité permet de trier les recettes et en obtenir seulement les recettes ayant un ingrédient en particulier. Il est également possible de faire une recherche avec un *match* exact d'ingrédient : seulement les recettes qui contiennent un ingrédient avec les text exact de la recherche seront retournées.

Par exemple : une recherche de l'ingrédient *oignon* affichera les 3 recettes suivantes puisqu'elles nécessitent un ingrédient qui contient la chaîne *oignon* : *Bol protéiné à la mexicaine (ingrédient : **oignon**)*, *Boeuf croustillant coréen (ingrédient : **oignon vert**)*, *Tofu croustillant Tonkatsu (ingrédient : **oignon(s) rouge(s)**)*. La même recherche, mais avec l'option **Match exact** d'activée retournera seulement la recette *Bol protéiné à la mexicaine*.

Une recherche avec un ingrédient qui ne fait pas partie de la liste d'aucune recette affichera une liste vide et un compteur qui indique "0 recettes" dans la page.

Notez que la recherche par ingrédient est indépendante du filtrage par catégorie. L'utilisation d'une des manières de recherche ignore la configuration de l'autre. Par défaut, une recherche par ingrédient enlève la sélection des boutons de filtrage par catégorie. Consultez la méthode `configureSearch()` dans le fichier `recipes.js` pour plus d'information.

Vous devez compléter la fonction `filterByIngredient` dans `recipe_manager.js` afin de répondre aux requis de la fonctionnalité. Une suite de tests unitaires dans le fichier `recipe_manager.test.js` vous est fournie pour vous permettre de bien implémenter votre fonctionnalité. Assurez-vous de bien lire les tests unitaires pour vous aider à compléter la fonction.

3.5 Vérification du code source

Le but principal de ce travail pratique est de mettre en place des tests unitaires pour le code qui vous est fourni. Vous trouverez les fichiers de test dans le répertoire `test/jest` déjà créés pour vous. Notez que certains tests unitaires sont déjà faits pour vous afin de vous aider à en écrire les autres. Vous avez également le titre des tests à compléter et qui vous donnera une idée du code à écrire pour compléter le test.

Assurez-vous de gérer les dépendances entre les différentes classes et/ou fonctions impliquées dans un test.Vous aurez à modifier le code fourni pour construire des *Stub*, *Mock* ou *Spy* lorsque nécessaire.

Avertissement

Les tests à compléter sont vides, mais sont quand même considérés comme réussis lors de l'exécution de tous les tests. Ne vous fiez pas seulement à la sortie dans la console pour savoir si vous avez terminé votre travail. On vous recommande de placer la ligne suivante dans vos tests avan leur complétion pour s'assurer qu'un test non-terminé échoue : `expect(false).toBe(true);`

Les sections suivantes décrivent les différents fichiers de test à compléter. Notez que dans certains fichiers, vous n'avez qu'à compléter des tests et dans d'autres, vous devez implémenter quelques fonctions qui vous permettront de bien configurer votre échafaudage de tests.

Vous pouvez ajouter d'autres tests unitaires si vous le jugez nécessaire. Vous devez viser une couverture complète (100%) du code source.

3.5.1 recipes_displayer.test.js

Vous devez implémenter la fonction `setUpHTML()` qui vous permet de configurer le DOM pour vos tests. Inspirez-vous de `recipes.test.js` qui contient un exemple d'échafaudage.

Vous devez également compléter les 3 tests unitaires vides dans ce fichier.

3.5.2 recipe_displayer.test.js

Vous devez implémenter la fonction `setUpHTML()` qui vous permet de configurer un DOM minimal pour vos tests. Vous pouvez vous inspirer du fichier `recipes.test.js` qui contient un exemple d'échafaudage. Vous devez compléter le *Stub* dans la variable `recipe`.

Vous devez également compléter les 3 tests unitaires vides dans ce fichier.

3.5.3 add_recipe.test.js

La configuration des tests dans ce fichier vous est fournie à titre d'exemple. Vous pouvez vous en inspirer pour vous aider à compléter les autres tests.

Notez que l'attribut `location` de l'objet `window` est modifé ici pour s'assurer qu'il n'y pas un chargement de page accidentel lors des tests.

Vous devez compléter les 2 tests unitaires vides dans ce fichier.

3.5.4 form.test.js

Vous devez compléter le *Stub* dans la variable `formStub` pour pouvoir adéquatement tester la soumission du formulaire. Lisez attentivement le reste du code du test unitaire fourni pour construire correctement votre *Stub*.

Vous devez également compléter le test unitaire vide au début de ce fichier.

3.5.5 recipe.test.js

Vous devez compléter le test unitaire vide au début de ce fichier. Vous pouvez ajouter d'autres tests unitaires si vous le considérez nécessaire.

3.5.6 recipes.test.js

La configuration dans ce fichier vous est fournie à titre d'exemple. Vous pouvez vous en inspirer pour les autres tests. Vous devez compléter les 2 tests unitaires vides dans ce fichier.

3.5.7 recipe_manager.test.js

Notez que la suite de tests **"filterByIngredient test"** est déjà remplie pour vous afin de pouvoir implémenter cette fonctionnalité. Vous n'avez pas besoin d'ajouter d'autres tests, à moins que les tests donnés ne couvrent pas le code que vous avez écrit.

Vous devez compléter les 8 tests unitaires vides dans ce fichier.

3.5.8 storage_manager.test.js

Notez que lorsque vous travaillez avec l'objet `LocalStorage`, la méthode `setItem` existe sur son *Prototype* et non sur l'objet directement. Ceci s'applique pour les autres méthodes de *Storage*. Vous avez un exemple fourni dans 2 des tests unitaires à compléter.

Vous devez compléter les 5 tests unitaires vides dans ce fichier.

Conseils pour la réalisation du travail pratique

1. Séparez bien les classes testées de leur dépendances. Faites un usage des concepts de *Stub*, *Mock* et *Spy* de Jest.
2. Lisez bien les tests fournis pour vous aider à implémenter la fonctionnalité manquante et les tests unitaires à compléter.
3. Consultez l'exemple de test unitaires avec Jest vu en classe.
4. Exécutez les tests fournis souvent afin de valider votre code.
5. Ajoutez des tests si vous jugez nécessaire afin de pouvoir couvrir complétement le code du projet.
6. Respectez la convention de codage établie par *ESLint*. Utilisez la commande `npm run lint` pour valider cet aspect.

4 Remise

Voici les consignes à suivre pour la remise de ce travail pratique :

1. Le nom de votre entrepôt Git doit avoir le nom suivant : **tp3_matricule1_matricule2** avec les matricules des 2 membres de l'équipe.
2. Vous devez remettre votre code (*push*) sur la branche **master** de votre dépot git. (pénalité de 5% si non respecté)
3. Le travail pratique doit être remis avant **23h55**, le **10 mars**.

Aucun retard ne sera accepté pour la remise. En cas de retard, la note sera de **0**.

Le navigateur web **Google Chrome** sera utilisé pour tester votre site web.

5 Évaluation

Globalement, vous serez évalués sur le respect des exigences fonctionnelles de l'énoncé, ainsi que sur la qualité de votre code JS et sa structure. Plus précisément, le barème de correction est le suivant :

Exigences	Points
Nouvelle fonctionnalité : Recherche par ingrédient	
Respect des exigences fonctionnelles de l'énoncé	2
Tests unitaires	
Tests pour `recipes_displayer.js`	2
Tests pour `recipe_displayer.js`	2
Tests pour `add_recipe.js`	2
Tests pour `form.js`	2
Tests pour `recipe.js`	1
Tests pour `recipes.js`	2
Tests pour `recipe_manager.js`	2
Tests pour `storage_manager.js`	2
Qualité du code	
Structure du code	1
Qualité et clarté du code	2
Total	20

L'évaluation se fera à partir de la page d'accueil du site, soit `index.html`. À partir de cette page, le correcteur devrait être capable de consulter toutes les autres pages de votre site web et interagir avec les différents éléments du site.

L'évaluations se fera à travers le projet `nightwatch` ainsi que les tests dans le projet `siteWeb`.Tous les tests fournis doivent obligatoirement passer. Tous les tests unitaires écrits par vous doivent obligatoirement passer. Vous devez avoir une couverture complète du code.

Le code doit respecter les règles établies par *ESLint*. La commande *lint* ne doit pas indiquer des erreurs dans la console après sont exécution.

Ce travail pratique a une pondération de **6%** sur la note du cours.

6 Questions

Si vous avez des interrogations concernant ce travail pratique, vous pouvez poser vos questions sur MS Teams ou contacter votre chargé de laboratoire.

Annexe

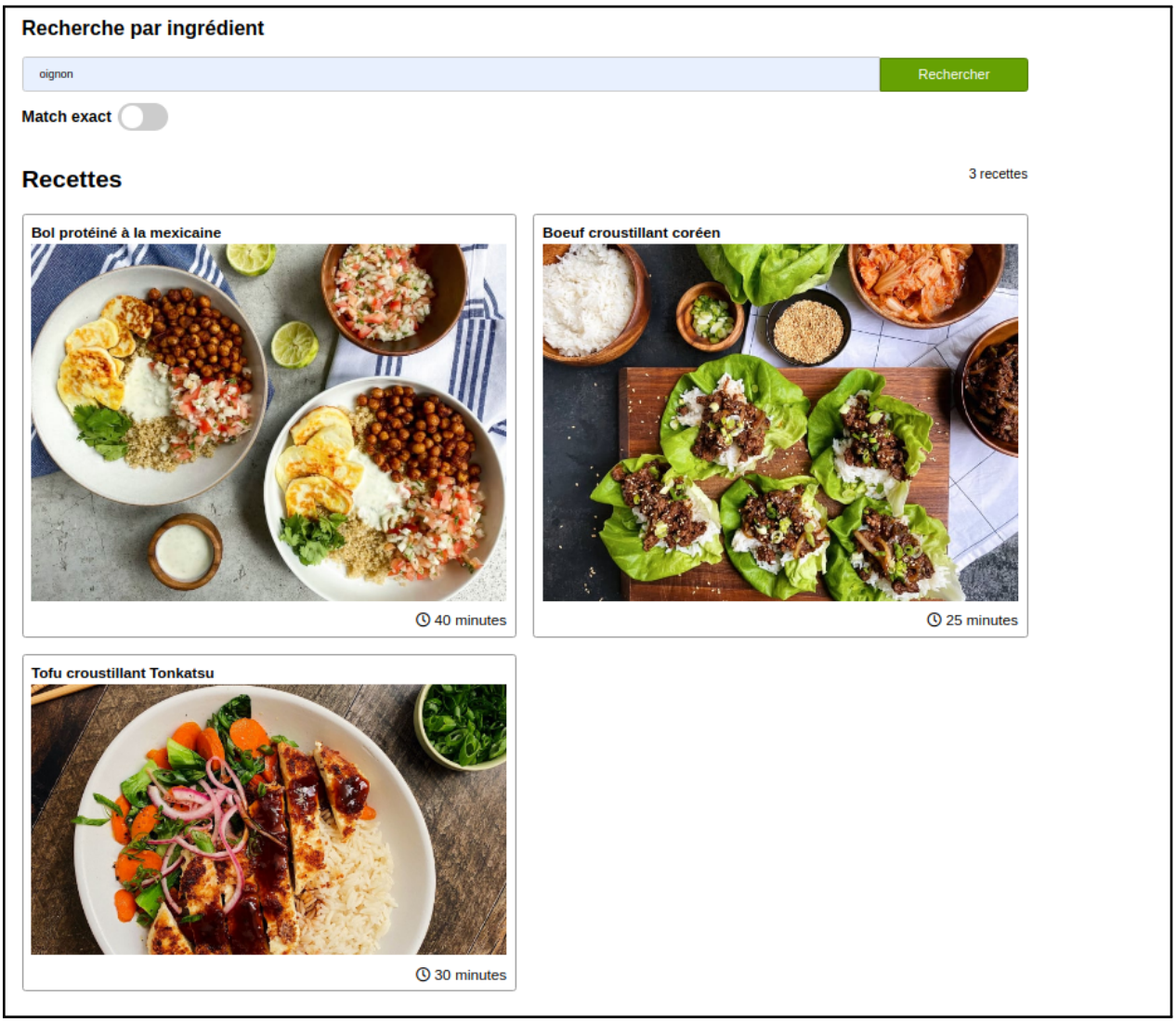

FIGURE 1 – Recherche par ingrédient

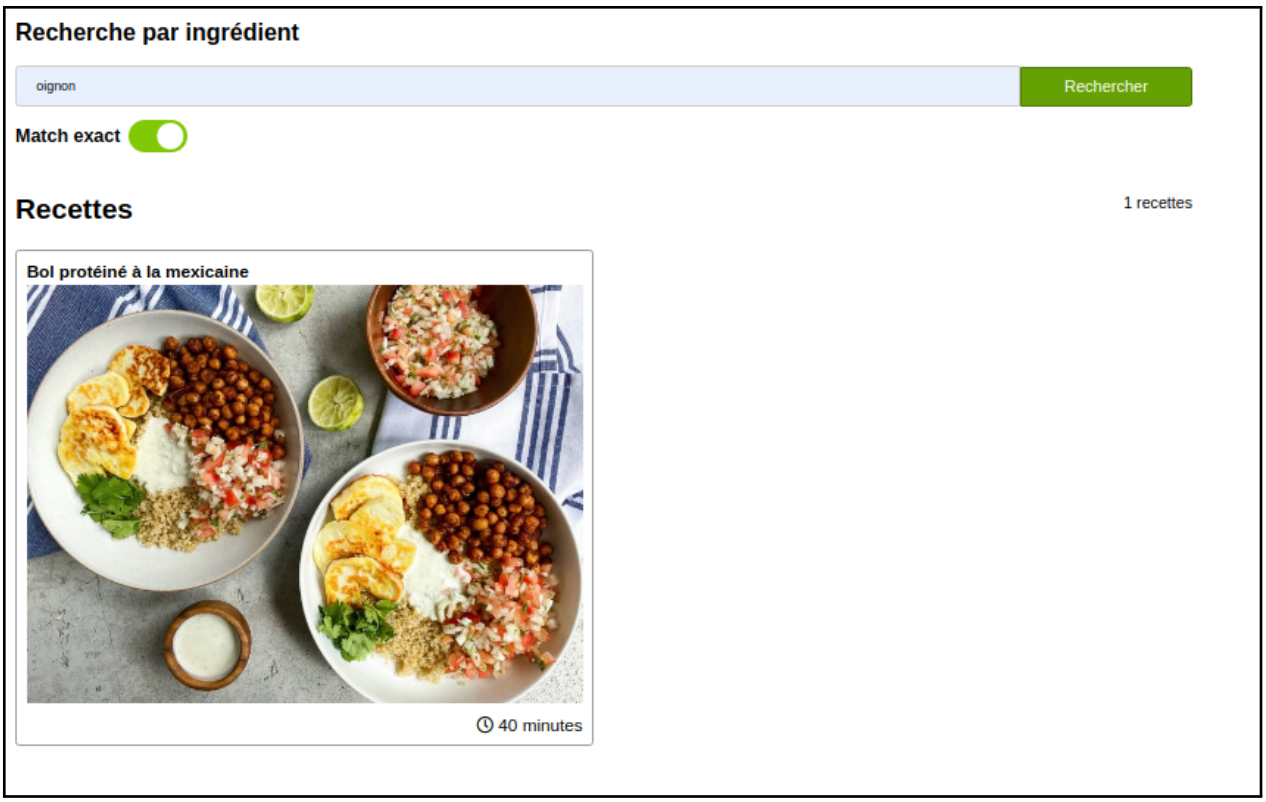

FIGURE 2 – Recherche par ingrédient avec un match exact