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

Travail pratique 2

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 introduire au langage de programmation JavaScript et la manipulation du DOM (Document Object Model). Vous allez vous familiariser avec la syntaxe ES2015+ de JavaScript et la gestion des événements. Finalement, vous utiliserez un outil d'analyse statique pour le respect de conventions de programmation (*linter*) : ESLint. Cet outil définit un ensemble de règles à suivre pour un code uniforme et standard.

De manière plus concrète, vous aurez à mettre en place plusieurs interactions sur différentes pages du site web afin que celui-ci devienne dynamique et fonctionnel. À la fin de ce travail, le site web sera utilisable, mais aucune mesure de sécurité ne sera présente puisque l'ensemble des opérations seront réalisées du côté client.

2 Introduction

Lors du premier travail pratique, vous avez mis en place la structure et la mise en forme du site web à réaliser. Cependant, comme vous l'avez sans doute remarqué, celui-ci était complètement statique. En effet, il n'était pas possible de choisir une recette spécifique à afficher où d'interagir avec le contenu du formulaire ou d'afficher une liste de recettes correspondant au critère de tri et à la catégorie sélectionnés, par exemple.

Afin d'être en mesure d'ajouter des interactions du côté client, le langage JavaScript doit être utilisé. En ce sens, ce langage permet entre autres de manipuler les éléments du DOM (*Document Object Model*), d'avoir accès aux API du navigateur (ex : Local Storage). Dans le cadre de ce travail pratique, la syntaxe *moderne* du JavaScript, ES2015+, sera utilisée.

3 Travail à réaliser

En vous basant sur le travail pratique précédent, vous aurez à ajouter la couche de logique du site web grâce au langage JavaScript. Un corrigé du TP1 vous sera fourni avec le dépôt de Git de départ afin de partir du bon pied pour ce travail. Notez que certaines pages HTML ont été modifiées par rapport à l'énoncé du TP1 pour les besoins du TP2.

Avertissement

Afin de promouvoir la compréhension du langage JavaScript, les bibliothèques telles que jQuery ne sont pas permises pour ce travail pratique. Vous devez manipuler le DOM seulement à travers les méthodes natives.

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 plusieurs tests pour chacune de vos pages vous sont fournis dans le répertoire `tests/e2e`. Ces tests vous permettront de valider votre travail et le contenu de vos pages HTML. Il est fortement recommandé de lire les tests et leur description avant d'écrire votre code pour vous aider avec le résultat final attendu.

Dans ce travail, vous aurez un ensemble de tests supplémentaires qui viendra s'ajouter aux tests du premier travail pratique. Ces tests vous permettront de valider les fonctionnalités supplémentaires à ajouter dans ce travail pratique.

Pour lancer la suite de tests, vous n'avez qu'à aller à la racine de votre répertoire git avec un terminal et de lancer la commande **npm run e2e** afin de lancer les tests fournis. Évidement, vous devrez au préalable avoir installé l'environnement d'exécution NodeJS.

Note : comme le site web utilise ESM, on ne peut pas charger les pages HTML avec "file" lors des tests. 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. Voir la section 3.1.

3.3 Éléments à réaliser

Maintenant que le serveur web est fonctionnel, la prochaine étape est d'implémenter les éléments nécessaires pour que le site soit utilisable. Avant de débuter, assurez-vous que le projet de départ est fonctionnel et vous êtes capables de naviguer à travers les différentes pages du document. Le code de départ présente un affichage statique des pages. Vous devez enlever certaines parties du contenu HTML puisqu'il sera généré par le code JS. Ces parties sont clairement indiquées par des commentaires dans les fichiers HTML.

3.4 Chargement et gestion des recettes

Le fichier `data/recipes.js` contient l'ensemble des recettes de départ pour le travail definies avec un format précis. Afin de pouvoir créer du HTML dynamiquement, votre site doit pouvoir charger les recettes et les sauvegarder dans son entrepôt local (Local Storage).

Vous devez compléter la classe `StorageManager` qui gère l'accès au *LocalStorage*. Lors du lancement du site, la classe doit vérifier si le storage contient déjà une liste de recette et sinon, charger le fichier `data/recipes.js` et le sauvegarder. Il doit être possible d'obtenir le contenu du storage à travers la méthode `getData` et sauvegarder une nouvelle recette à travers la méthode `saveData`. Finalement, la méthode `resetData` permet de vider le contenu de l'entrepôt local. Consultez le fichier `storage_manager.js` et les entêtes des fonctions fournies pour plus de détails des paramètres d'entrée et valeurs de retour des méthodes.

Vous devez compléter la classe `RecipeManager` qui gère l'accès aux recettes obtenues par `StorageManager`. On peut ajouter une recette à travers la méthode `addRecipe`, récupérer une recette en fonction de son identifiant à travers la méthode `getRecipe` ainsi que de filtrer les recettes selon leur catégorie à travers la méthode `filterRecipes`. Consultez le fichier `recipe_manager.js` et les entêtes des fonctions fournies pour plus de détails des paramètres d'entrée et valeurs de retour des méthodes.

3.5 Affichage de la liste des recettes et filtrage

3.5.1 Liste des recettes

Votre site doit être capable d'afficher dynamiquement toutes les recettes de départ ainsi que les recettes qui ont été ajoutées à travers la page "Ajouter une recette".

Vous devez gérer un nombre arbitraire de recettes et générer dynamiquement le code HTML de l'affichage de chaque recette. Le code de départ fourni présente l'affichage statique

voulu dans l'élément `<div>` ayant l'id `recipes-list`. Votre travail sera de générer le contenu HTML de cet élément avec du code JavaScript.

Note : cette balise doit être vide (aucun contenu) à la remise. Tout le contenu doit être généré par le JavaScript.

Vous devez compléter la fonction `displayRecipes` dans le fichier `recipes.js`. Cette fonction s'occupe de générer l'HTML d'affichage pour la liste des recettes. L'HTML généré doit être le même que le code de départ et être validé par les tests fournis. La fonction peut prendre en paramètre une catégorie de filtrage. La page doit alors seulement afficher les recettes de cette catégorie. Voir la section 3.5.2 pour la description du filtrage des recettes. La fonction met à jour le nombre de recettes affichées à travers la fonction `updateRecipesNumber` qui doit également être complétée par vous.

Consultez le fichier `recipes.js` et les entêtes des fonctions fournies pour plus de détails des paramètres d'entrée et valeurs de retour des méthodes.

3.5.2 Filtrage des recettes

Par défaut, la page "Trouver une recette" affiche toutes les recettes (initialement 5 recettes). Le bouton "Toutes les recettes" possède la classe CSS `selected`.

Il doit être possible d'appliquer un filtre de catégorie et afficher seulement les recettes de cette catégorie. Par défaut, il y a 1 recette végétarienne, 2 recettes méditerranéennes et 2 recettes ketos. Le filtre doit être appliqué lorsqu'un utilisateur clique sur un des 4 boutons de la section "Catégories". Le bouton cliqué doit désormais être le seul ayant la classe CSS `selected` et l'affichage doit automatiquement se mettre à jour pour afficher les recettes filtrées ainsi que nombre total de recettes affichées.

Le bouton "Toutes les recettes" affiche toutes les recettes disponibles, peu importe leur catégorie. Ce bouton et cette option de filtrage sont toujours ceux sélectionnés par défaut lorsqu'on accède à la page "recipes.html".

Vous devez compléter la fonction `configureFilters` dans le fichier `recipes.js`. Cette fonction s'occupe d'attacher un gestionnaire de l'événement `click` à chaque bouton qui doit gérer l'application du filtre et la mise à jour de l'affichage correspondant. La méthode doit être capable de gérer un nombre arbitraire de boutons de filtrage et non seulement les 4 boutons initiaux.

3.6 Affichage des information d'une recette spécifique

Votre site doit être maintenant capable de rediriger l'utilisateur vers la page de recette avec le bon affichage et non un contenu statique. Il doit être possible de cliquer sur une recette dans la liste de recettes de la page "recipes.html" et être automatiquement redirigé vers la page "recipe.html" qui contient la description de cette recette.

Pour permettre de savoir quelle recette charger, un paramètre supplémentaire doit être passé dans l'URL de chaque balise `<a>` . Ce paramètre correspond à l'attribut "id" de chaque recette. Par exemple, pour charger la recette avec id=3, l'URL de redirection doit être **"/recipe.html ?id=3"**. On peut accéder à la page d'une recette à travers la liste de recettes ou directement avec *votreSiteWeb/recipe.html ?id=X* où X est l'id de la recette recherchée. L'objet URLSearchParams ainsi que les propriétés search et location de l'objet `document` vous aideront à récupérer la valeur du paramètre id de l'URL une fois sur la page "recipe.html".

Une fois sur la page "recipe.html", vous devez générer dynamiquement l'HTML nécessaire pour afficher les ingrédients, outils et étapes de la recette ciblée. Le code de départ présentent des exemples des conteneurs à remplir : 2 balises `<ul>` ayant les id `list-ingredients` et `tools-list` ainsi qu'une balise `<div>` ayant la classe `recipe-container`. Vous devez également modifier le nom et l'image de la recette.

Note : ces balises doivent être vides (aucun contenu) à la remise. Tout le contenu doit être généré par le JavaScript.

Vous devez compléter les fonctions données dans le fichier `recipe.js` : `displayRecipe`, `displayTools`, `displayIngredients` et `displaySteps` qui permettent de générer l'HTML nécessaire pour le bon affichage des informations de la recette.

Consultez le fichier `recipe.js` et les entêtes des fonctions fournies pour plus de détails des paramètres d'entrée et valeurs de retour des méthodes.

3.7 Ajouter une nouvelle recette

Votre site doit être maintenant capable de permettre à l'utilisateur d'ajouter sa propre recette à travers le formulaire de la page "add_recipe.html". Une recette ajoutée à travers le formulaire est sauvegardée localement et disponible pour consultation dans la liste des recettes de la page "Trouver une recette".

Chaque recette est représentée par une structure uniforme. De plus, les ingrédients, les outils nécessaires et les étapes varient d'une recette à l'autre. Consultez le fichier `data/recipes.js` pour la structure exacte d'une recette.

3.7.1 Ajouter une nouvelle étape

Le formulaire d'ajout doit être adapté pour permettre plusieurs étapes pour une recette. Le bouton **Ajouter une étape** doit permettre de générer et ajouter le code HTML nécessaire pour ajouter une autre étape à la recette. Le code de départ de la page "add_recipes.html" contient la structure HTML que vous devez générer avec du code JS. Un utilisateur doit pouvoir ajouter un nombre d'étapes arbitraires à sa recette, mais il ne peut pas en enlever. Chaque étape est numérotée de manière séquentielle. Cette numération doit être reflétée dans l'interface et les attributs *id* des étapes ajoutées.

Vous devez implémenter la gestion de l'événement `click` du fichier `add_recipe.js`. Un exemple du comportement attendu est disponible en annexe.

3.7.2 Soumettre le formulaire

Une fois le formulaire rempli, l'utilisateur doit être capable de le soumettre et ajouter la recette à la liste des recettes sauvegardées localement. Une fois la recette sauvegardée, elle doit être présente dans la liste des recettes et être consultée comme toute autre recette.

Il y a seulement 4 types de recettes et le type "Autre" ne fait pas partie des options de filtrage. Une recette "Autre" est seulement visible lorsqu'on affiche toutes les recettes.

La soumission du formulaire se fait à travers le bouton "Ajouter la recette". Vous devez implémenter la gestion d'événement de soumission `submit` dans le fichier `add_recipe.js`.

Votre code doit récupérer les donnes du formulaire et construire un objet qui suit la structure d'une recette. La propriété elements d'un formulaire vous sera utile. Lorsque l'objet est construit, vous devez l'ajouter aux autres recettes dans le *LocalStorage*. Une fois l'ajout effectué, la page doit être rechargée et tous les champs du formulaire doivent être vides.

Vous devez faire des manipulations supplémentaires pour extraire certaines données. Les ingrédients sont donnés dans le format suivant : *ingrédient1 :quantité1,ingrédient2 :quantité2* où chaque ingrédient est séparé par une virgule (,) et le nom et la quantité sont séparés par deux points (:). Par exemple : 1 oignon et 1 tomate seront donnés de la manière suivante : "*oignon :1,tomate :1*". Les outils sont séparés par une virgule (,). Le formulaire doit permettre un nombre arbitraire d'ingrédients et outils, mais il doit avoir au moins 1 outil et 1 ingrédient.

La fonction `getImageInput` vous est fournie pour vous aider à extraire une image à partir d'un champ de saisi de fichier `HTMLInputElement`. Notez que la fonction est marquée comme asynchrone et vous devez donc utiliser le mot clé `await` lors de son appel dans votre code. Exemple : `const monImage = await getImageInput(monInput).`

3.7.3 Réinitialiser la liste

Pendant le développement d'un projet, on voudrait parfois être capables de remettre notre application dans un état initial valide. Par exemple, on voudrait supprimer toutes les recettes sauvegardées dans notre *LocalStorage*.

Le bouton "Réinitialiser la liste" doit permettre à l'utilisateur de vider le *LocalSorage*) des recettes sauvegardées localement. Une fois la réinitialisation effectuée, le site doit automatiquement rediriger l'utilisateur à la page "recipes.html". Rendu à cette page, seulement les 5 recettes initiales doivent être présentes et aucun filtre de catégorie ne doit être appliqué.

Vous devez implémenter la logique de réinitialisation et redirection dans la fonction `resetData` du fichier `add_recipe.js`. Cette fonction doit être appelée lorsque l'utilisateur clique sur le bouton "Réinitialiser la liste".

Conseils pour la réalisation du travail pratique

1. Effectuez une validation de vos formulaires grâce aux attributs de HTML5 (voir la section « *Input Restrictions* » de la page *HTML Input Types*).
2. Utilisez les outils de développement de votre navigateur web pour vous aider à déboguer votre code JS (raccourci F12).
3. Respectez la convention de codage établie par *ESLint*. Utilisez la commande `npm run lint` pour valider cet aspect.
4. Utilisez les méthodes des tableaux JS (*map*, *filter*, etc) le plus souvent possible.
5. Exécutez les tests fournis souvent afin de valider votre code.

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 : **tp2_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 **17 février**.

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
Aspect global du site	
Respect des exigences fonctionnelles de l'énoncé	2
Code JavaScript	
Code dans `recipe_manager.js`	2
Code dans `storage_manager.js`	2
Code dans `recipes.js`	3
Code dans `recipe.js`	3
Code dans `add_recipe.js`	4
Qualité du code (Respect des règles ESLint)	
Structure du code	2
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. Tous les tests fournis doivent obligatoirement passer. Le serveur web doit être déployable avec la commande start seulement.

Ce travail pratique a une pondération de **5%** 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

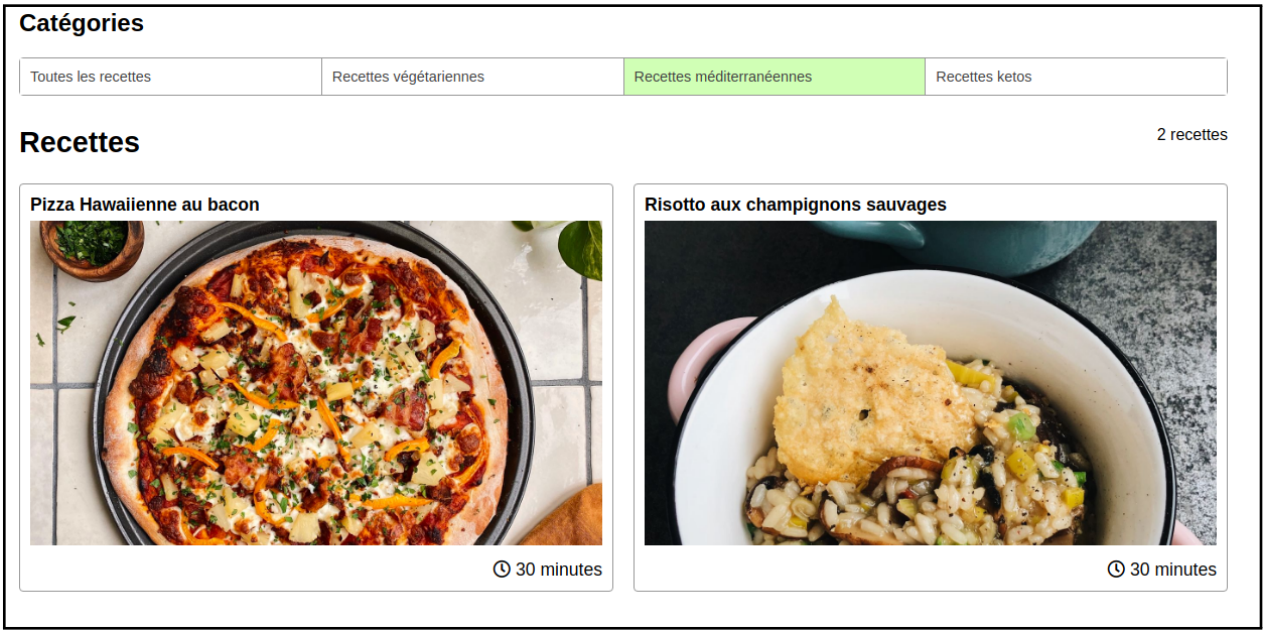

FIGURE 1 – Filtrage de recettes par catégorie

FIGURE 2 – Ajout d'une étape de recette

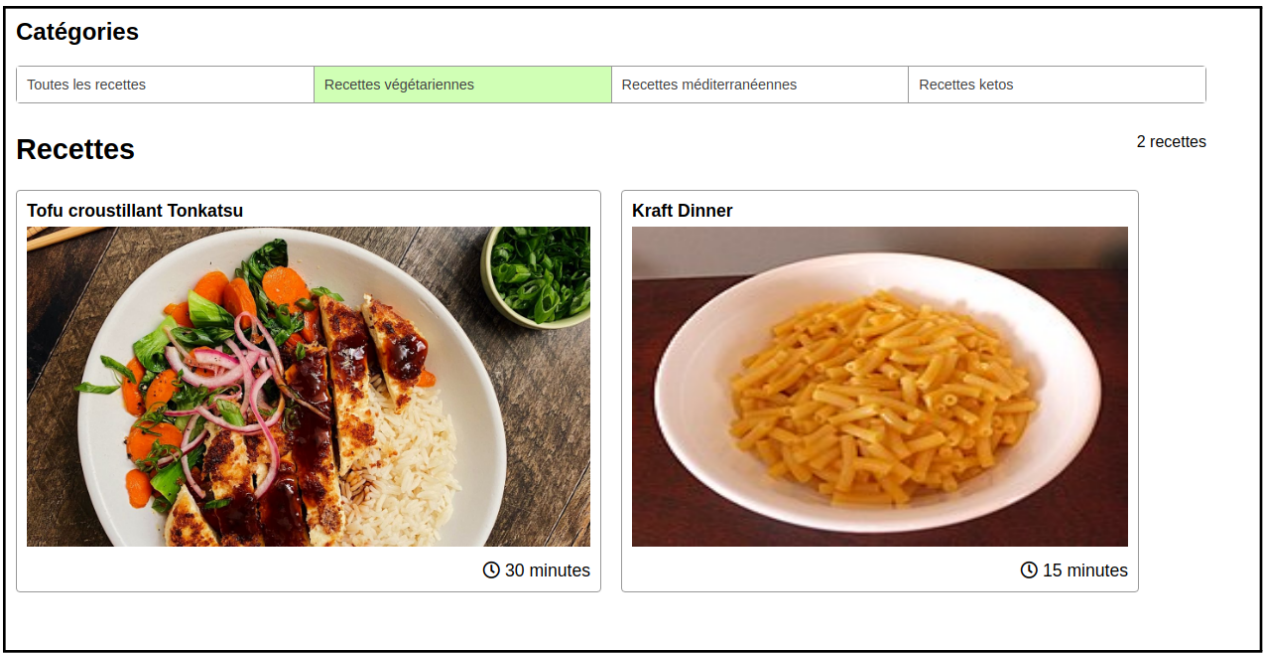

FIGURE 3 – Affichage de nouvelle recette ajoutée