Tests application DogHouse

Gestionnaire d’emploi du temps pour un salon de toilettage.

# Présentation du projet

Notre projet consiste en une application de gestion d’emploi du temps pour une entreprise. Nous avons choisi de concevoir l’application pour un salon de toilettage canin. Ce projet a été réalisé entièrement en Javascript et sans framework ni code html.

## Décomposition du projet

Le fichier index.html à la racine du projet est la seule et unique page de l’application. Son contenu est géré par de nombreux fichiers Javascripts.

Nos ressources Javascript sont situées dans le répertoire *js*. Dans ce répertoire se trouvent 2 autres répertoires, ainsi que les sources du modèle et contrôleur. Dans le répertoire *view* se trouvent les fichiers Javascript gérant l’affichage pour l’utilisateur et le répertoire *drivers* contient les sources Javascript de notre gestionnaire de routes, de notre gestionnaire de sessions et de notre gestionnaire de base de données.

## Problème connu avant la réalisation des tests

Le développement en pur Javascript ayant pris plus de temps que prévu, une fonctionnalité n’a pu être intégrée. Il s’agit de la vérification de la superposition ou du chevauchement de créneaux. Aucun test ne portera donc dessus par la suite.

# Déroulement des tests

Pour les tests, nous avons utilisé plusieurs systèmes. Sonar nous a d’abord permis de faire un état des lieux de notre code Javascript. On a ensuite utilisé QUnit pour le côté contrôleur de l’application, puis Selenium pour le côté vue de l’application.

## Analyse des sources avec SonarJS

SonarJS nous a retourné plusieurs types d’erreurs dans nos sources. Nous n’avons pas eu à exclure de fichiers de l’analyse étant donné que nous n’avons pas utilisé de framework pour notre application. Les erreurs les plus fréquentes qui nous ont été retournées sont l’utilisation des « == » au lieu des « === » lors de comparaisons dans les conditions, ou bien encore la virgule « , » finale lors d’une itération d’attributs dans la définition d’un objet. Une dernière erreur récurrente concernait le point-virgule « ; » finale après certaines déclarations de fonctions.

Ces erreurs ont été vites corrigées pour la plupart et dans la mesure du possible. On est passé de 64% à 68% de conformité avec les normes strictes de Javascript pour l’ensemble de nos fichiers.

## Tests du contrôleur avec QUnit

Les fichiers de tests QUnit se situent dans le répertoire *test* de l’application.

* client\_test : Dans le fichier JS sont répertoriés les tests d’ajout/suppression etc… concernant les clients du salon de toilettage, ainsi que d’ajout/suppression etc… pour ce qui concerne les chiens.
* employe\_test : Dans le fichier JS sont répertoriés les tests de connexion à l’application par les employés.
* creneau\_test : Dans le fichier JS sont répertoriés les tests concernant l’ajout et la suppression de créneau dans l’emploi du temps.
* doghouse\_test\_suite : Dans le fichier JS sont répertoriés l’ensemble des tests ci-dessus.

## Tests de la vue sous SeleniumIDE

Les tests sous SeleniumIDE se situent dans le répertoire *tests\_selenium*. N’ayant pas de fichiers de chargements, pour le bon fonctionnement des tests, et pour s’assurer de l’intégrité des données du localStorage, il faut exécuter le test doghouse\_selenium\_tests.html.

* Add\_creneau : crée un client et son chient, ajoute un créneau pour ce dernier, et supprime le créneau
* Create\_client : crée un client
* Create\_dog : Crée un animal pour le dernier client créé
* Create\_client\_fail : test qui vérifie si lors de la création d’un client, s’il manque un champ, déclanche une alerte.
* Delete\_client : Supprime le client créé dans Create\_client
* Login : Teste la connexion
* Login\_fail : Teste une erreur de connexion
* Logout : teste la déconnexion

Ces tests nous ont permis de nous assurer du bon comportement des actions répétitives.

## Couverture des tests grâce à SonarJS

Nous avons tenté d’effectuer la vérification de couverture des tests. Malheureusement, une erreur concernant notre objet Database que nous n’avons pas réussi à résoudre nous a empêché d’avancer, étant donné que nous utilisons notre objet Database à chaque test.

Nous n’avons donc malgré nos vaines tentatives aucune couverture à indiquer.

# Conclusion

Pour conclure, l’analyse de notre code faite par SonarJS nous a permis de mettre en lumière principalement des erreurs dans l’utilisation de Javascript. Comme, durant le développement de l’application, nous avions déjà séparé notre code en de nombreux fichiers, nous avons pu facilement identifier les problèmes et les corriger. Nous avons privilégié QUnit pour ce qui concerne le modèle et le contrôleur et SeleniumIDE pour tout ce qui concerne l’interface utilisateur. Au final, très peu de modifications ont été nécessaires, les fonctionnalités de base fonctionnant sans soucis. Le point noir reste finalement l’impossibilité d’avoir pu effectuer les tests de couverture sous SonarJS.