TP3 - Logique d'un quiz avec JavaScript
6 octobre 2016

1 Objectifs

Les buts de ce travail pratique, réalisé en équipe de deux personnes exactement, sont :

- De vous familiariser avec JavaScript et la librairie jQuery.
- De préparer des services web sur le serveur.
- D'utiliser le stockage local chez le client.

À la fin de ce TP, le site sera utilisable, considérant un contexte d'auto-évaluation informelle, sans mesures de sécurité.

2 Introduction

Tout au long du trimestre, vous implémenterez un site web qui génère des questionnaires (quiz) pour les personnes qui désirent tester leurs connaissances sur les notions du Web. Nous nous attarderons principalement sur des questions sur le domaine du HTML5, CSS3 et JavaScript, mais nous pouvons étendre ces questions à d'autres technologies du Web telles que HTTP, NodeJS, AngularJS, etc.

3 Principes de fonctionnement

Nous présentons dans cette section les principes de fonctionnement du questionnaire, afin de vous fournir une vision globale de ce que vous devrez implémenter tout au long du trimestre. Mais toute la logique et les calculs derrière ces principes ne seront pas implémentés avant le TP3.

La page centrale du site correspond au tableau de bord de l'utilisateur. Le tableau de bord donne la possibilité à l'utilisateur d'effectuer des tests rapides ou un examen officiel. Il contient également diverses informations sur les résultats des tests de l'utilisateur tels que le nombre de tests rapides réussis sur le nombre de tests rapides effectués, et la moyenne des examens.

Les tests rapides permettent à l'utilisateur d'améliorer ses connaissances sur le Web. Le choix des questions du test se fera de façon aléatoire selon une base de données. Cette base de données va contenir l'ensemble des questions possibles sur chaque domaine.

Les examens officiels permettent de tester officiellement les connaissances de l'utilisateur. À côté du bouton qui débute l'examen, l'utilisateur devra rentrer les paramètres de l'examen. Ces

paramètres peuvent correspondre au domaine dans lequel il désire se faire évaluer (HTML, CSS ou JavaScript) et le nombre de questions que devra contenir l'examen.
Les pages de tests rapides et d'examens sont très similaires.

La page de test rapide contient la question, ainsi qu'un choix multiple de réponses à la question (des boutons radio ou des cases à cocher). L'utilisateur a le choix de retourner au menu (sans avoir besoin de répondre à la question), ou de répondre à la question et de cliquer sur le bouton **suivant**. S'il clique sur le bouton **suivant**, une nouvelle question à laquelle l'utilisateur doit répondre s'affiche, et ainsi de suite.

La page d'examen ressemble énormément à la page de test rapide. Nous retrouvons une question avec les réponses possibles. Ensuite, l'utilisateur a le choix d'abandonner l'examen (en conséquence, il aurait une note de 0 pour cet examen), ou de répondre à la question et d'aller à la question suivante.

4 Travail à réaliser

Dans le TP1 et le TP2, vous avez élaboré la structure et la mise en forme de votre site web. Vous n'avez pas encore ajouté la couche logique au site. Au cours du TP3, vous allez implémenter la logique du site web à l'aide de JavaScript.

Afin de promouvoir la compréhension du fonctionnement des composantes, les seules librairies externes permises pour ce TP sont **jQuery** et **Modernizr**.

Selon certaines directives, vous serez obligés de modifier légèrement votre structure initiale afin de répondre aux exigences.

Dans ce travail pratique, vous aurez besoin de sauvegarder les données du joueur afin de les utiliser de pages en pages. Pour ce faire, vous devez utiliser le *localStorage* de HTML5 avec JavaScript.
Plusieurs sites offrent des tutoriels simples, dont http://diveintohtml5.info/storage.html . Vous pouvez visualiser l'état de votre *localStorage* avec l'outil de débogage de votre fureteur (raccourci F12).

Également, vous serez obligés de manipuler le DOM de vos pages HTML et d'effectuer des traitements sur des listes JavaScript. Pour ce faire, je vous conseille d'utiliser les fonctions disponibles dans *jQuery*.

Les questions et les réponses seront obtenues par requêtes AJAX. Je vous recommande d'utiliser des fonctions de *jQuery* pour gérer cette requête.

Je recommande aussi de transmettre ces informations à l'aide d'objets JSON. *jQuery* offre des fonctions conçues pour obtenir et parser ce type d'objet.

Voici les directives de ce travail pratique :

1. Ajouter un nouveau fichier de routage sur votre serveur pour les requêtes AJAX. Ceci aura la même fonctionnalité que *index.js* utilisé au TP2. La raison de la séparation n'est que pour le regroupement logique des fonctionnalités.
   Pour que votre serveur commence à utiliser ce fichier, vous allez éditer *app.js* dans la racine de votre projet.
   Premièrement, ajoutez une ligne *require* pointant vers votre ficher et assignez une variable à ce dernier (comme celles qui sont déjà là).
   Par la suite, vous pourrez ajouter une autre entrée *app.use*, comme celle qui est déjà là, pour référer au chemin que vous voulez pour votre service (par exemple /ajax ou /api), suivi de la variable de votre fichier de routage.
2. Ajouter une/des route(s) pour pouvoir effectuer des requêtes pour une nouvelle question et obtenir l'objet réponse. (JSON suggéré, mais vous êtes libre de choisir tout autre format).
   Pour retourner directement un JSON avec Express, vous pouvez utiliser retourner res.json dans la fonction de routage.
3. Création d'une pseudo base de données sur votre serveur qui sera utilisée dans les tests rapides ET les examens.
   La pseudo BD peut simplement être un tableau d'objets JavaScript côté serveur. Vous pouvez le mettre dans fichier .js, dans un dossier (que vous appellerez Data, par exemple).
   Par la suite, vous pouvez faire référence à ces objets à partir de votre fichier de routage, en utilisant require(./cheminVersFichierExterne).
   Un minimum de 10 questions devrait se retrouver dans cette BD. Chaque entrée devrait contenir l'ID de la question, son domaine, la question à poser, les choix de réponses et sa bonne réponse.
4. Validation des données du formulaire d'initialisation d'un examen.
   Pour le mode examen, l'utilisateur devrait être capable de choisir un domaine de questions possibles.
   Il devrait également pouvoir indiquer le nombre de questions à faire pour l'examen.
   Vous pouvez garder les préférences courantes de l'usager dans *sessionStorage* (comme *localStorage*, mais expire à la fin de la session) afin de pouvoir faire la bonne requête.
5. Lors d'un test rapide, une question sera choisie aléatoirement de la pseudo-BD. La logique de sélection aléatoire sera du côté serveur. Vous pouvez garder cette logique relativement simple, comme choisir une valeur entre 0 et le nombre total de questions, puis l'utiliser pour référer à l'index de la question.

Nous devrons voir le domaine de la question, la question elle-même et ses réponses possibles. Un test rapide n'a jamais de fin. Lorsque l'utilisateur clique sur « Question suivante », alors une nouvelle question sera choisie aléatoirement de la BD. Un test rapide s'arrête uniquement lorsque l'utilisateur clique sur « Retourner au menu ».

Pour créer un examen, il faut extraire des questions de la pseudo-BD selon les paramètres du formulaire d'examen. Vous pouvez avoir des routes spécifiques à la catégorie demandée, ou encore utiliser des paramètres de GET.

Pour ce TP, les questions dupliquées à cause des requêtes aléatoires sont permises.

6. Les statistiques courantes du test rapide ou de l'examen seront mis à jour selon les questions répondues par l'utilisateur.
7. Les statistiques permanentes de l'usager, comme le nombre d'examens réussis et échoués par catégorie, et le nombre de questions rapides réussis ou échoués, etc. doivent être stockés de manière à ce qu'ils restent disponibles si l'usager revient plus tard et affiche ses statistiques (sauf si localStorage a été effacé ou si l'usager réinitialise ses statistiques).
8. L'abandon d'un test rapide retourne sans conséquence le joueur au tableau de bord. L'abandon d'un examen apporte l'utilisateur à la page terminale (fin de l'examen) avec une note de 0 / X.
   Ce n'est que lorsque l'usager atteint la page finale que l'examen est considéré comme réussi et qu'il incrémente les statistiques de l'usager.
9. Lors de la fin d'un examen, son domaine et son résultat (le nombre de questions réussies sur le nombre de questions effectuées) seront sauvegardés.

   Chaque résultat est affiché dans la boîte de détails d'examens.
   Par exemple, « Examen 1 - (HTML CSS) : 9 / 10 »

   La moyenne de tous les examens est affichée dans la boîte de statistiques du tableau de bord en forme de pourcentage.
10. Pour sélectionner la bonne réponse, l'usager doit glisser la bonne réponse dans une zone « réponse » avec la souris (*drag and drop*). Utiliser les fonctionnalités de HTML5 avec JavaScript pour cette opération. Vous pouvez vous référer à : https://www.html5rocks.com/en/tutorials/dnd/basics/ .
11. Quand une réponse est sélectionnée, l'affichage offre une rétroaction visuelle (ex. encadrer de rouge ou de vert). De plus, l'usager n'a plus le choix de changer sa réponse. Il ne peut que poursuivre ou quitter.
12. Lors des examens ou des questions rapides, le contenu de la page est modifié de manière asynchrone avec AJAX. Ce n'est que le contenu et les réponses qui changent.
13. Dans le tableau de bord, nous devrons retrouver un bouton nommé « Remise à zéro » qui réinitialise les données sauvegardées (tel que sa note des tests rapides, la liste

d'examens). Vous pouvez placer ce bouton en dessous de la boîte de statistiques du tableau de bord.

14. La page de résultat d'un examen doit indiquer la note obtenue dans l'examen. Le message devrait dépendre de la note obtenue par l'utilisateur. Il y aura 4 types de messages selon 4 paliers de notes différentes : (0% à 25%), (25% à 50%), (50% à 75%) et (75% à 100%).
15. Fonctionnelle sur Firefox
16. Suivre les bonnes pratiques de codages JavaScript.

5 Évaluation

Globalement, vous serez évalués sur la logique de l'application qui sera implémentée à l'aide de JavaScript. Plus précisément, le barème d'évaluation est le suivant :

Exigences	Points
Fonctionnalité et qualité de l'API sur serveur	15
Statistiques cohérentes, fonctionnelles et affichées correctement	25
Appels AJAX et mises à jour dynamiques fonctionnels	25
« Drag & drop » des réponses, validation, et progression entre questions	25
Autres directives et appréciation globale	10

L'évaluation se fera à partir de la page d'accueil. À partir de cette page, je devrais être capable de consulter toutes les pages de votre site web.

6 Direction de la remise

1. Pour la remise, on demande de placer votre code source dans un dossier compressé nommé : TP3_matricule1_matricule2. Vous avez le choix du type de compression. Le dossier node_modules doit être exclu.

2. Vous devrez également créer un fichier nommé temps.txt à l'intérieur du dossier de votre projet. Vous indiquerez le temps passé **au total** pour ce TP.

3. Le travail pratique doit être remis **avant** 23h55 le dimanche 23 octobre 2016 sur Moodle.

7 Pénalités

1. La politique pour les travaux pratiques remis en retard : -20% pour chaque jour de retard.

2. Les directives 1 et 2 de la remise : -5% si ces directives ne sont pas respectées.

Bonne chance !