Examen Web App – 2017 – M44

- Tout document papiers autorisé.
- Utilisation de l'ordinateur restreinte à l'utilisation d'un lecteur de PDF, NetBean ou autre éditeur de code, un browser limité au site Internet du cours (support de cours, liens de premier niveau, et exercices compris) et au localhost (127.0.0.7).
- Pas d'autre communication réseau (ou directe) autorisée pendant le travail.
- Toute non-observation des règles ci-dessus entraîne l'échec du module (et de sa rémédiation).

A la fin de votre travail vous rendrez le dossier contenant vos codes et vous donnerez votre nom à ce dossier. Vous êtes responsable de son contenu, et devez vous assurer de la bonne réception du dossier avant de quitter la salle. Le travail sera rendu au plus tard à **11h40** par e-mail à **nicolas.chabloz@heig-vd.ch**

Mise en place (10 minutes)

Téléchargez le fichier de l'examen sur le site du cours. Décompressez le dans votre répertoire htdocs (www, ou autre) de votre serveur Web et créez un nouveau projet « avec sources PHP existantes » sous NetBean (ou un autre éditeur). Vérifiez que tout fonctionne bien en vous rendant à l'url correspondante de votre projet. Vous pouvez à loisir modifier les fichiers HTML, CSS et JavaScript de l'examen.

Cadrage: Une association d'astronomes amateurs désire proposer à ses membres une application Web (Single Page App) pour l'organisation d'événements (ci-après "events") d'observation du ciel. L'application devra permettre aux membres de l'association de consulter, d'ajouter et de mettre en favoris des events. N'importe quel membre pourra proposer un event. Pour des raisons de simplicité, les events ne seront définis que par un titre, une date et un lieu. Comme les membres de l'association n'ont souvent pas accès au Web, il vous est demandé de réaliser une application qui permet la consultation en mode "offline". En mode "offline", seul la consultation des events (listing et favoris) sera possible (l'ajout d'un event ne sera donc pas possible). Toutefois, la mise en cache de l'application ne sera pas à faire dans cet examen.

1) Gestion du menu (20 minutes)

Le menu de la WebApp est simple. Il n'y a que trois fonctionnalités, le listing des events, l'ajout d'un event et "mes events". Vous devez donc coder la gestion des clicks sur les éléments du menu de navigation afin que ceux-ci affichent la "page" appropriée. Au chargement de la WebApp, le listing des events sera affiché. Les éléments du menu doivent changer de couleur pour indiquer à l'utilisateur à quelle "page" il se trouve. De plus l'historique du browser doit rester fonctionnel, c'est à dire que l'utilisateur pourra utiliser les boutons "back" et "forward" de son browser pour naviguer entre les "pages".

2) Status online / offline (20 minutes)

Vous devez indiquer à l'utilisateur le statut offline/online de la WebApp en changeant la couleur du nuage en haut à droite de la WebApp (utiliser les classes CSS existante .online et .offline). De plus lorsque le statut ***online/offline*** est à ***offline***, vous devez désactiver le lien du menu permettant l'ajout d'un event. Il doit être grisé et ne plus rien faire si l'utilisateur click dessus. Lorsque la WebApp est à nouveau en mode ***online***, le lien doit redevenir fonctionnel et ne plus être grisé. **Remarque**: vous pouvez utiliser la classe css ***.disabled*** pour griser le bouton.

3) Synchronisation des événements du backend et du LocalStorage (20m)

Afin que l'application gère la communication des événements entre les membres, ils seront sauvegardés sur le backend. La WebApp devra alors récupérer ces événements pour maintenir sa liste coté client à jour (synchronisation via AJAX). Pour des raisons de simplification, cette synchonisation sera faite **uniquement** au chargement de la page **et si** l'utilisateur n'est pas en mode ***offline***. Le backend fournit la liste complète des

événements sous la forme d'un tableau des événements encodés en JSON. Le service backend est accessible via la méthode HTTP **GET** à cette url: ***backend/get.php*** et devrait normalement déjà vous retourner un tableau contenant deux events de test. Il vous suffit donc de vider le ***JsonStorage*** adéquat, de "boucler" sur ce tableau et d'ajouter les évennements un par un dans le ***JsonStorage***.

4) Mise à jour du DOM des events (20 minutes)

Chaque fois que le ***JsonStorage*** des événements est modifié (ainsi qu'au chargement de la page), vous devez refléter ce changement dans le DOM de la WebApp via un **trigger**. Pour simplifier les choses, la WebApp effacera tous les événements et réaffichera le tout. Vous trouverez dans le code HTML une template pour l'affichage d'un événement, vous pouvez y rajouter des classes CSS. La liste des événements est à ajouter dans ** .**
Remarque: vous ne devez pas vous occuper de la gestion de l'affichage des dates au format français.

5) Ajout d'un rendez-vous (40m)

Lorsque l'utilisateur ajoute un rendez-vous, vous devez transmettre les données de l'événement pour le sauvegarder sur le backend de l'application (via AJAX). Les données sont à reprendre du formulaire. **Vous ne devez pas** vous occuper de la validation des données à ce moment (le backend le fera), ni de la gestion du calendrier pour la date (utiliser le browser Chrome). Le service backend est accessible via la méthode HTTP POST à cette url: ***backend/add.php*** et accepte trois paramètres POST nommés ***title***, ***date*** et ***place***. Si un des ces paramètres est manquant, ou n'a pas le bon format, le backend vous retournera une erreur au format JSON. Exemple de réponses: **{"status":"fail","data":{"event":"A title is required"}}** ou encore: **{"status":"error","data":{"event":"The title is malformatted"}}** ou si l'événement a bien été sauvegardé sur le backend: **{"status":"success","data": null}**

Dans les cas où la réponse aurait un status différent de ***success***, vous devez afficher un message informant qu'une erreur s'est produite. Dans le cas où le status est à ***success***, un autre message l'informant de la réussite de l'opération devra être affiché et les champs du formulaire seront vidés. Les messages doivent disparaître lorsque l'utilisateur quitte la section d'ajout ou retente un ajout (vous pouvez testez, par exemple, que le message d'erreur et le message de succès ne peuvent apparaître tous les deux en même temps). Vous trouverez les textes de ces messages dans le HTML de la WebApp, ces messages ne doivent pas être des "alert" mais bien des éléments DOM comme c'est déjà le cas dans le HTML.

De plus, si l'ajout de l'événement à réussi (status ***success***), vous devez l'ajouter au ***JsonStorage*** (et de rafraichir le DOM via un **trigger** pour déclencher le code du point **4**).

6)Ajout et suppression d'événement à "Mes Events" (40m)

Dans la liste des events, l'utilisateur pourra cliquer sur le bouton "Ajouter à mes events" pour les events qu'il désire mettre en favoris. Lors du click, vous devez récupérer l'event associé du ***JsonStorage*** et l'ajouter dans **un autre *JsonStorage*** *(nommé par exemple "favoris").*
Comme au point **4)**, vous devez mettre à jour le DOM de la "page" des favoris (** **) lorsqu'un changement ce produit dans le ***JsonStorage*** des favoris (ainsi qu'au chargement de la page). Cette fois utilisez la deuxième template disponible (celle avec le bouton de suppression).
Finalement, vous devez gérer les clicks sur les boutons "Supprimer de mes events". Il vous suffira alors d'enlever l'event associé du ***JsonStorage*** *des favoris* (et de rafraichir le DOM via un **trigger**).