Application : forum de discussion

Nicolas Richard

6 avril 2021

Table des matières

1 Introduction et description générale

Le but de ce mini-projet est de mettre en pratique un certain nombre de techniques et technologies vues durant les cours de développement web. Ce travail devrait être réalisable en 6h de travail, pour tout qui a assimilé la matière vue jusqu'alors.

Le travail consiste en la création d'un serveur de discussions, prenant en charge l'historique des messages. Le serveur sera scindé en plusieurs canaux ayant chacun un sujet spécifique.

Votre serveur devra pouvoir accepter des demandes d'utilisateur pour afficher canal de discussion. Le serveur lui présentera alors l'état de la discussion dans ce canal spécifique. L'utilisateur pourra y contribuer en y écrivant de nouveaux messages, qui seront ajoutés à l'historique. Les autres utilisateurs connectés à ce canal devront récupérer automatiquement ces nouveaux messages.

Utilisez Git pour gérer les différentes versions de votre projet. À chaque étape vous devriez faire un commit.

2 Décomposition du projet

2.1 La base de données

Un script SQL est fourni. Prenez-en connaissance pour comprendre sa structure.

2.2 Mise en place d'une API HTTP

Dans cette première phase, le but est d'arriver à afficher le contenu de la base de donnée (les messages qui existent déjà), et d'en rajouter, via une API en HTTP.

Pour cela, définissez les routes suivantes dans le fichier `routes/api.php` :

— `/channels/` en GET, présente une liste de canaux
— `/channels/{chatRoomId}/messages` en GET, affiche les messages issus du canal donné
— `/channels/{chatRoomId}/messages` en POST, poste un message

Les éléments nécessaires à chaque demande sont transmis comme paramètres de la requête :

login le login de l'utilisateur

content le contenu d'un message

N.B. La réponse est retournée en JSON.

2.2.1 Exemple de requêtes

Voici quelques requêtes (en jQuery) que le serveur doit supporter. Notez que le fichier api.php ajoute automatiquement le préfixe `api/`.

```
$.getJSON("/api/channels/", function (data, status) { console.log(data); })
$.getJSON("/api/channels/21/messages", function (data, status) { console.log(data); })
$.post("/api/channels/21/messages",
       { login: "nri",
         content: "J'y crois pas ? Ça marche !" },
       function(data, status) {
           console.log(data)
       })
```

2.3 Interface utilisateur

Vous pouvez maintenant écrire l'interface utilisateur (UI) grâce notamment à AJAX.

Il faut que l'utilisateur puisse :
— rentrer son login pour se "connecter" (sans mot de passe)
— choisir un canal (webg4, dev1, ...)
— Les messages du canal sélectionné s'affichent
— Un formulaire permet d'envoyer un message sur le canal courant
— Rafraichir la liste des messages manuellement (ou automatiquement, par exemple toutes les N secondes)

Nous vous proposons de créer deux pages :
— une page de connexion et de choix du canal, générée côté serveur
— une page avec les différents messages ; celle-ci sera construite sous forme de SPA, grâce à l'API ci-dessus

Utilisez les possibilités offertes par CSS pour obtenir un affichage acceptable.

Utilisez un canevas pour les éléments communs aux deux pages.

Veillez à placer tous les appels à la base de donnée dans le ou les modèles. Il faut également parer à toute forme d'injection SQL.

2.4 Aller plus loin

Bien sûr vous pouvez améliorer votre mini-projet autant que vous voulez. Quelques idées :
— Mettre en évidence les messages "non-lus" lorsque l'utilisateur se connecte à l'application.
— Mettre en oeuvre un modèle de sécurité comprenant l'ajout d'une authentification par mot de passe
— Améliorer l'aspect "temps réel" par l'utilisation des websockets
— Ajouter une fonctionnalité "messagerie privée"