

Labo 3 : Bot-tender Chatroom

Créé par Christopher Meier.
Inspiré d'un exercice écrit par Haoyi Lee.

Objectifs

L'objectif de ce labo est de créer un espace de discussion Web (*chatroom*) en utilisant Scala et de découvrir les bibliothèques [Cask](#) et [ScalaTags](#). Dans cet espace de discussion, plusieurs utilisateurs pourront communiquer ensemble. Les messages de chacun seront visibles par tous sur une page Web.

- *Cask* est un framework HTTP simple en Scala inspiré par le projet Flask de Python.
- *ScalaTags* est un moyen simple et rapide de générer du code HTML/XML/CSS en utilisant Scala.
- Les avantages de ces deux bibliothèques sont qu'elles sont légères, elles intègrent bien les constructions Scala et qu'elles évitent l'utilisation de frameworks complexes pour des cas simples.

Ce laboratoire intègre le bot implémenté lors du Labo 1 et du Labo 2. Un utilisateur pourra ainsi communiquer avec le bot-tender depuis l'espace de discussion.

Indications

Le code source du laboratoire vous est fourni sur Cyberlearn.

- Ce laboratoire est à effectuer par groupe de 2, en gardant les mêmes formations que pour le labo 2. Tout plagiat sera sanctionné par la note de 1.
- Votre implémentation du labo 2 doit être intégrée à ce laboratoire. Si vous le souhaitez vous pouvez utiliser l'implémentation d'un de vos collègues (n'oubliez pas d'indiquer la source en commentaire)
- Ce laboratoire est à rendre en plusieurs parties (voir section Rendus). Pour simplifier la récupération, nous vous demandons de créer un repo Github privée (gratuit) avec les membres de votre groupe et d'inviter l'assistant et le professeur comme collaborateurs. Les commits représentant les étapes à rendre seront taggés.
- Il n'est pas nécessaire de rendre un rapport, un code propre et correctement commenté suffit. Faites cependant attention à bien expliquer votre implémentation.
- Faites en sorte d'éviter la duplication de code.
- Préférez un style de programmation fonctionnel.

Pour faire ce laboratoire, il faut dans un premier temps parcourir les fonctionnalités principales fournies par ces deux bibliothèques via les deux liens suivants : [Cask](#) et [ScalaTags](#). Vous allez ensuite utiliser certaines de ces fonctionnalités lors des étapes d'implémentation.

Description

Le site Web de chatroom est composé de deux pages principales :

1. La page d'accueil, qui contiendra les messages de l'espace de discussion ainsi qu'un formulaire pour envoyer un nouveau message. (voir figure 1)
2. La page de login et d'inscription, qui contiendra respectivement les formulaires pour connecter un utilisateur existant et pour créer un nouvel utilisateur. (voir figure 2)

Afin d'explorer les capacités de la bibliothèque *Cask*, nous allons implémenter plusieurs manières de communiquer entre le navigateur et le serveur, telles que l'envoi de formulaires POST ou encore la souscription à un websocket.

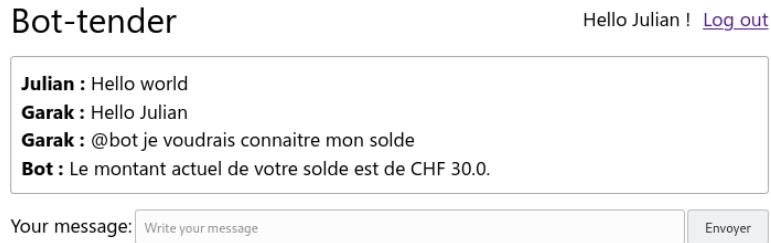


FIGURE 1 – Une capture de la page d'accueil



FIGURE 2 – Une capture de la page de connexion et d'inscription

Implémentation

Pour vous guider dans votre implémentation, nous avons établis des étapes. Le code source qui vous est fourni fait référence à ces étapes dans des commentaires (`// TODO - Part 3 Step N`).

Étape 0 : Creation du repo git

1. Créer un repo privé Github avec les membres du groupes.
2. Ajouter l'assistant (`c-meier`) et la professeure (`nfatemi`) comme collaborateurs.
3. Donner le lien de votre repo sur Cyberlearn.

Étape 1 : Rendre disponible les fichiers statiques

Complétez la classe `StaticRoutes` de sorte que les fichiers statiques `js/main.js` et `css/main.css` (situés dans le dossier `src/main/resources`) soient disponibles depuis le navigateur.

Étape 2 : Mettre en place la page d'accueil

— Dans la classe `MessagesRoutes`.

- En utilisant ScalaTags, créer la structure HTML qui correspond à la figure 3. La page doit être disponible depuis l'url /
- Ajouter les méthodes (de préférence factorisé) qui génère l'HTML dans l'objet **Layout**
- La figure 3 indique la structure HTML des captures d'écran. Les éléments en gras indique les attributs nécessaires au code Javascript fourni.
- Les messages seront chargés et affichés à l'étape 4. Par défaut, afficher un texte centré indiquant que les messages sont en cours de chargement.
- Lier les fichiers CSS et Javascript de l'étape 1 au code HTML

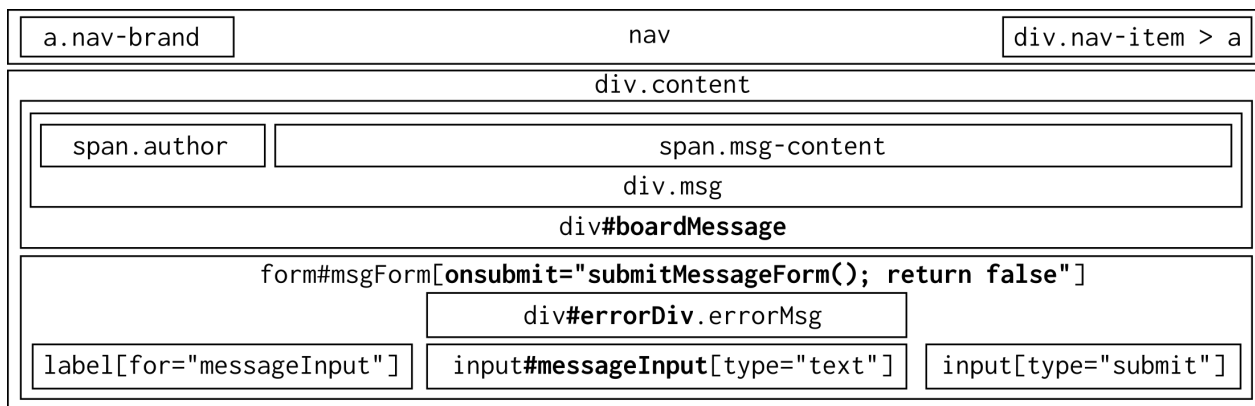


FIGURE 3 – Un model de la structure de la page d'accueil

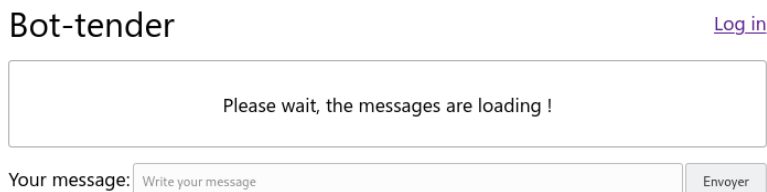


FIGURE 4 – Une capture de la page d'accueil à l'étape 2

Étape 3 : Mettre en place les fonctionnalités de login

- Dans la classe **UsersRoutes**.
- En utilisant ScalaTags, créer une page permettant de se connecter et de s'inscrire. La page doit être disponible depuis l'url /login
- Si le nom d'utilisateur de la tentative de connexion n'existe pas sur le serveur, il faut afficher un message d'erreur.
- Le formulaire de connexion doit envoyer à l'url /login une requête POST avec le nom d'utilisateur en utilisant le format "form".
- Le formulaire d'inscription doit envoyer à l'url /register une requête POST avec le nom d'utilisateur en utilisant le format "form".
- Une connexion ou inscription réussie doit afficher une page Web indiquant le succès.
- L'accès à l'url /logout doit déconnecter l'utilisateur actuel et afficher une page Web indiquant le succès.
- L'identité de l'utilisateur connecté est stockée dans une session. Le code permettant de récupérer la session actuelle est fourni dans la classe **Decorators**.

Bot-tender

[Go to the message board](#)

Login

The specified user does not exists

Username:

Register

Username:

FIGURE 5 – Une capture de la page de connexion avec une erreur

Étape 4 : Traitement des messages

- Dans la classe `MessageImpl` implémenter le stockage des messages de l'espace de discussion.
- En plus du nom de l'expéditeur et du contenu du message, `MessageService` stocke également les infos suivantes, si elles existent :
 - Le nom de l'utilisateur mentionné dans le message. Un message mentionne un utilisateur si le message commence par "@". Par exemple, le message "@Julian Hello" mentionne l'utilisateur "Julian".
 - Le type de la requête (`ExprTree`) envoyé au Bot. Par exemple, le message "@bot Je suis affamé" doit avoir le type `Hungry`.
 - L'id du message auquel le message actuel réponds. Par exemple, le message du bot qui répond à un message de requête a une référence vers l'id du message de requête.
- Dans la classe `MessagesRoutes`, implémenter la reception des messages envoyés par le formulaire à l'url `/send` au format JSON (code javascript fourni).
- Le code Javascript fourni se connecte à un websocket depuis l'url `/subscribe`. À chaque nouvelle notification, il met à jour (remplace) les messages affichés avec les 20 dernier messages. Vous devez implémenter les endpoints pour que lorsqu'un utilisateur envoie un message tous les utilisateurs le reçoivent immédiatement.
- Un GET sur l'endpoint `/clearHistory` doit supprimer tout l'historique des messages.

Bot-tender

[Log in](#)

No messages have been sent yet

You must be logged to send a message

Your message:

FIGURE 6 – Une capture de la page de d'accueil avec une erreur

Étape 5 : Intégrer le bot-tender à l'espace de discussion

- Modifier la classe `MessagesRoutes` pour que les messages qui commencent par `@bot` soient traités par le Bot-tender implémenter au Labo 2.
- La réponse du bot est envoyée sur l'espace de discussion et donc visible par tout le monde.
- Le bot doit utiliser l'utilisateur connecté depuis la page de login

— Les messages d'identification ("Je suis _Michel") ne doivent avoir aucun effet sur l'authentification Web.

Rendus

Pour assurer un suivi continu du laboratoire et pour nous permettre de vous donner des feedbacks au fur et à mesure, nous vous demandons de rendre votre code plusieurs fois.

Lorsque vous êtes prêt à faire un rendu, veuillez suivre les instructions suivantes (adapter au besoin) :

1. Commiter le travail que vous voulez rendre (ex. `git commit`).
2. Tagger le commit actuel avec un tag annoté (ex. `git tag -a part3-stepN`). Comme commentaire, indiquez nous si vous avez eu un éventuel problème.
3. Pusher le tag qui vient d'être créé (ex. `git push origin part3-stepN`).

Vous devez effectuer les rendus suivants :

- Pour Mer. 11.05.2022 à 23h59, il faut faire les étapes 1 et 2. Utiliser le tag `part3-step2`.
- Pour Mer. 18.05.2022 à 23h59, il faut faire l'étape 3. Utiliser le tag `part3-step3`.
- Pour Mer. 25.05.2022 à 23h59, il faut faire les étapes 4 et 5 et rendre toutes les sources. Utiliser le tag `part3-step5`.