

Gestion des utilisateurs avec une session et mise en forme avec bootstrap

Benoit Favre, modifié par Thomas Schatz

généré le 15 mars 2022

1 Introduction

L'objectif de ce TP est de compléter un site web de gestion de recettes afin de n'autoriser la modification des données qu'à des utilisateurs authentifiés. Vous allez gérer la création de comptes, le login, et la restriction des pages nécessitant une authentification. On améliorera aussi la présentation visuelle du site à l'aide du framework CSS bootstrap. On rendra notamment le site autoajustable (responsive) en fonction de la largeur de l'écran.

2 Présentation du site

Données. Les données initiales sont dans `data.json`. Elles ont été extraites du site <https://cuisine-facile.com/> et restent propriété de ce site. On va utiliser une base de données SQLite pour stocker et manipuler ces données. Le script `create-db.js` permet de remplir la base à partir du contenu de `data.json`.

Modèle. Une recette est constituée de métadonnées comme son titre, sa description ou sa durée, ainsi que d'une liste d'ingrédients et une liste d'étapes (*stages*). Dans la suite, une recette est représentée en javascript sous la forme de l'objet suivant :

```
let recipe = {
  id: integer, // identifiant; seulement lorsque l'on lit une recette depuis la base
  title: 'text',
  description: 'text',
  duration: 'text',
  ingredients: [
    {name: 'text'},
    {name: 'text'},
    ...
  ],
  stages: [
    {description: 'text'},
    {description: 'text'},
    ...
  ]
}
```

La base de donnée contient trois tables créées avec les requêtes suivantes :

```
CREATE TABLE recipe (id INTEGER PRIMARY KEY AUTOINCREMENT, title TEXT, img TEXT,
  description TEXT, duration TEXT)
CREATE TABLE ingredient (recipe INT, rank INT, name TEXT)
CREATE TABLE stage (recipe INT, rank INT, description TEXT)
```

La table `recipe` contient le titre, l'url de l'image, la description et la durée des recettes ; la table `ingredient` contient la liste ordonnée des ingrédients, et `stage` contient la liste ordonnée des étapes. Chaque ingrédient est associé à un numéro de recette, son indice dans la liste des ingrédients (`rank`) et le texte qui le décrit (`name`). Chaque étape (`stage`) de la recette est associée à son indice (`rank`) dans la liste des étapes et sa description.

Le modèle est défini dans le fichier `model.js` et offre les fonctions suivantes :

- `read(id)` : obtenir le contenu d'une recette à partir de son identifiant
- `create(recipe)` : sauvegarder une recette dans la base à partir de sa représentation sous forme d'objet javascript
- `update(id, recipe)` : modifier le contenu d'une recette identifiée par `id` dans la base, à partir de sa représentation sous forme d'objet javascript
- `delete(id)` : supprimer une recette à partir de son identifiant
- `search(query, page)` : rechercher des recettes à partir d'une requête textuelle, et obtenir une page de résultats (d'indice `page`, pour des pages de 32 éléments)

Le fichier `model.js` contient plus de détails sur ces fonctions.

Vues. Les vues sont fondées sur des templates `mustache` qui sont conservés dans le répertoire `views/`. Le site contient les templates suivants de vues :

- `views/header.html` : entêtes en commun entre toutes les pages
- `views/footer.html` : pied de page de toutes les pages
- `views/index.html` : page principale
- `views/search.html` : affichage des résultats de recherche
- `views/read.html` : affichage du contenu d'une recette
- `views/create.html` : formulaire de création d'une recette
- `views/update.html` : formulaire de mise à jour des informations d'une recette
- `views/delete.html` : formulaire de confirmation d'effacement d'une recette

Afin d'éviter les répétitions, deux vues partielles (`header.html` et `footer.html`) contiennent le début et la fin du html partagé par les trois vues. Le début contient en particulier un formulaire de recherche affiché sur toutes les pages.

Routes. Le serveur express est dans `server.js`. Les routes auxquelles répond le serveur sont les suivantes :

- GET `/` : page principale du site
- GET `/search?query=text&page=num` : page `num` de résultats de recherche pour la requête `text`
- GET `/read/:id` : affichage du contenu de la recette d'identifiant `id`
- GET `/create` : formulaire de création d'une recette
- GET `/update/:id` : formulaire de mise à jour des informations de la recette d'identifiant `id`
- GET `/delete/:id` : formulaire de confirmation d'effacement de la recette d'identifiant `id`
- POST `/create` : route de création effective d'une recette
- POST `/update/:id` : route de modification effective de la recette d'identifiant `id`
- POST `/delete/:id` : route de suppression effective de la recette d'identifiant `id`

Le fichier `server.js` contient plus de détails sur le fonctionnement de ces fonctions. Pour faire fonctionner le serveur vous devrez installer les modules manquants : `express`, `mustache-express`, `better-sqlite3`.

3 Partie 1 : Sessions

L'objectif est d'ajouter la gestion des utilisateurs et d'interdire aux utilisateurs non connectés d'accéder aux pages qui permettent de modifier la base. Pour cela, vous allez devoir implémenter l'authentification des utilisateurs, la création de nouveaux utilisateurs ainsi que le contrôler les accès aux différentes routes. Tout d'abord, assurez-vous que vous comprenez bien tous les aspects du fonctionnement du site, puis :

1. Créez une table `user` dans la base SQLite avec trois champs : `id` un identifiant unique auto-incrémenté, `name` le nom de l'utilisateur et `password` son mot de passe.
2. Ajoutez manuellement un utilisateur à la base pour pouvoir effectuer des tests (vous pouvez par exemple inclure la création de la table et de l'utilisateur dans le script `create-db.js`).

3.1 Préparation

Tout d'abord, pour faciliter votre tâche, installez le paquet `cookie-session` avec npm. Vous pouvez y faire appel dans le serveur comme cela :

```
const cookieSession = require('cookie-session');
app.use(cookieSession({
  secret: 'mot-de-passe-du-cookie',
}));
```

Ce module¹ permet de garder des informations de session dans un cookie crypté passé au client à chaque requête. Il faut y associer un mot de passe unique à l'application (typiquement généré aléatoirement) pour que ce cryptage soit effectif. (Notez que si l'accès au site ne se fait pas par le protocole https, l'encryption du cookie de session n'empêche pas un utilisateur malveillant ayant accès au réseau de se faire passer pour un autre utilisateur en récupérant son cookie encrypté et en le présentant au serveur).

3.2 Contrôle d'accès

Le contrôle d'accès sera implémenté sous la forme d'un middleware nommé `is_authenticated` qui vérifiera que la session contient un identifiant d'utilisateur et renverra une erreur 401 (accès non autorisé) si ce n'est pas le cas. Ce middleware sera utilisé uniquement sur les routes à accès restreint. Pour rappel, on peut appliquer un middleware à une route spécifique de cette façon :

```
function middleware(req, res, next) {
  // ...
  next();
}
app.get('/route', middleware, (req, res) => ...);
```

3.3 Vues conditionnelles

Le barre du haut définie dans la vue `header.html` et la vue `read.html` contient des boutons qui permettent de créer et modifier les recettes. On souhaite maintenant ne pas afficher ces boutons si l'utilisateur n'est pas authentifié.

Pour que les templates `mustache` aient accès à l'information d'authentification, il faut créer une valeur dans le dictionnaire `res.locals`, ce qui permet de conditionner le rendu à la valeur de cette variable. Il faut créer un nouveau middleware qui va mettre `res.locals.authenticated` à `true` si la session est valide, et copie le nom de l'utilisateur depuis la session dans `res.locals.name`. Ce middleware devra être appelé à toutes les routes, ce qui peut être fait avec `app.use(middleware)`.

On pourra alors écrire un template `mustache` de ce type :

```
{{#authenticated}}
  <p> contenu uniquement visible si l'utilisateur est authentifié </p>
{{/authenticated}}
```

De la même manière, on peut cacher du contenu lorsque l'utilisateur est authentifié :

```
{{^authenticated}}
  <p> contenu affiché lorsque authenticated est non défini ou contient une valeur fausse
{{/authenticated}}
```

1. Documentation ici <https://expressjs.com/en/resources/middleware/cookie-session.html>.

3.4 Authentification

1. Ajoutez une fonction `login(name, password)` au modèle qui renvoie le numéro d'utilisateur s'il est bien authentifié, ou -1 sinon.
2. Créez une vue `views/login.html` avec un formulaire d'authentification avec deux champs `name` et `password` et un bouton de soumission. L'action du formulaire sera d'appeler la route POST `/login`.
3. Ajoutez une route GET `/login` qui renvoie le formulaire d'authentification
4. Ajoutez une route POST `/login` qui vérifie les identifiants passés dans `req.body.name` et `req.body.password` (nom des champs dans le formulaire), et ajoute l'identifiant et le nom de l'utilisateur à la session s'ils sont valides. En cas de succès, rediriger vers `/`, sinon vers `/login`.
5. Ajouter une route POST `/logout` qui désauthentifie l'utilisateur (c'est à dire met la session à `null`) et redirige le client vers `/`.
6. Ajoutez le nom de l'utilisateur et un bouton "Se déconnecter" dans la barre en haut de la page lorsque l'utilisateur est authentifié.

3.5 Création d'utilisateurs

1. Ajoutez une fonction `new_user(name, password)` au modèle qui crée un nouvel utilisateur dans la base et renvoie son identifiant.
2. Créez une vue `views/new_user.html` avec un formulaire d'ajout d'utilisateur avec deux champs `name` et `password` et un bouton de soumission. L'action du formulaire sera d'appeler la route POST `/new_user`.
3. Ajoutez une route GET `/new_user` qui renvoie le formulaire d'ajout d'utilisateur
4. Ajoutez une route POST `/new_user` qui ajoute effectivement l'utilisateur, l'enregistre dans la sessions et redirige le client vers `/`
5. Ajoutez deux boutons "Se connecter" et "Nouvel utilisateur" à la vue `views/index.html` si l'utilisateur n'est pas connecté.

4 Partie 2 : mise en forme avec Bootstrap

Vous devez modifier les templates des vues pour obtenir une mise en forme du site avec Bootstrap. Le site de Bootstrap (<https://getbootstrap.com/>) contient une documentation et des exemples (n'hésitez pas à regarder le code source de la page de chaque exemple).

Utiliser Bootstrap Pour ce TP, vous ne devez qu'inclure le lien vers le CSS Bootstrap dans les entêtes de votre HTML (pas besoin des script javascript, ni d'utiliser npm). Ce fichier CSS vous donne accès à toutes les classes CSS de Bootstrap.

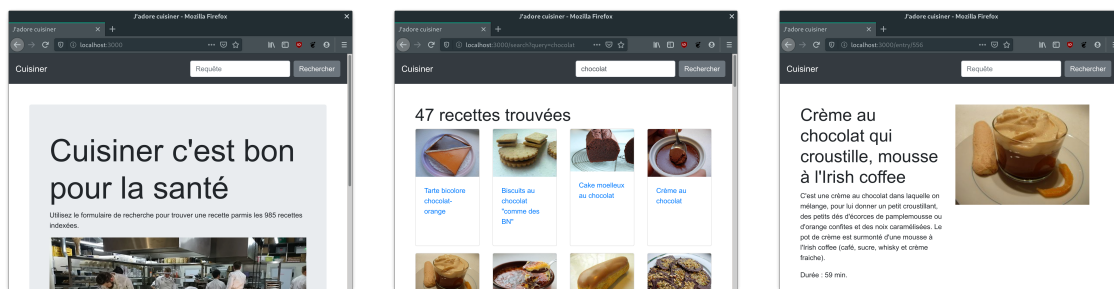


FIGURE 1 – Capture d'écran de la forme attendue pour un sous-ensemble des vues

4.1 Barre en haut de page

Pour la barre en haut de la page, vous vous inspirerez de l'exemple Bootstrap "starter template". La barre ne contiendra qu'un texte sur la gauche (lien vers /) et le formulaire de recherche sur la droite. Ce contenu doit être inséré dans les sous-templates `header.html` et `footer.html`. Vous devrez lire la documentation sur la barre de navigation (`navbar`).

4.2 Vue de la page principale

Cette vue s'inspire de l'exemple "jumbotron". Notez que l'image doit être de classe `img-fluid`.

4.3 Vue d'une recette

La vue d'une recette utilisera le système de grille Bootstrap (grid). Elle sera constituée de trois lignes (row), la première pour le titre, la description, la durée et l'image de la recette ; la seconde pour les ingrédients et la troisième pour les étapes de la recette.

Le système de grille permet de d'afficher un nombre de colonnes (col) variable en fonction de la largeur de l'écran, de manière à adapter le site, par exemple, à un écran de téléphone portable. Vous utiliserez cette capacité pour la première ligne qui contient les informations générale de la recette. Cette ligne devra avoir deux colonne, une avec les informations textuelles (titre, description...) et une autre avec l'image de la recette. Utilisez les classes `row-cols` de manière à ce que l'affichage prennent deux colonnes lorsque l'écran est large et une seule lorsque l'écran est petit (voir la documentation de grid).

4.4 Vue des résultats de recherche

Les résultats de recherche seront présentés sous la forme de cartes (card). L'exemple "album" et la documentation des cartes (card) vous y aideront. Notez que vous pouvez utiliser la classe `h-100` pour que les cartes soient de la même hauteur. N'hésitez pas à modifier la marge entre les cartes avec des classes du type `mb-3`. Le nombre de cartes affichées en largeur devra être variable en fonction de la largeur de l'écran (par exemple de 4 à 1 cartes lorsque la largeur diminue).

4.5 Autres vues

Utilisez bootstrap à votre convenance pour mettre en forme les autres vues.

5 Pour aller plus loin (optionnel)

1. Cryptez le mot de passe de l'utilisateur avec `bcrypt` avant de le sauvegarder dans la base de données
2. Affichez des messages en cas de mauvais identifiants, ou pour avertir l'utilisateur qu'il s'est bien connecté ou déconnecté
3. Rendre le site conforme avec la RGPD