CARLU Ludovic et FAIVRE Maxime
4A CFA - ESIEA 2018-2019

Hellociné

GitHub : https://github.com/Faivrem/HelloCine
Déploiement - Glitch : https://faivrem-hellocine-7.glitch.me

DESCRIPTION ET VUE D'ENSEMBLE DU PROJET

Hellociné est un site web où l'on peut obtenir différentes informations sur nos films et séries préférés. Celui-ci offre plusieurs possibilités à l'utilisateur et réponds aux différentes contraintes demandées.

Interface web

Nous avons pu construire notre interface web grâce à plusieurs outils. Le langage **HTML** pour la structure de notre page. Une feuille de style **CSS** (appelée "**mon_style.css**") qui a pour but de surcharge notre utilisation de la librairie bootstrap.

Par ailleurs, nous avons utilisé le framework **Vue.js** pour ce qui est du Frontend et ainsi fournir une **Single Page Application**. De ce fait, nous avons une unique page html qui est dynamique en fonction des actions de l'utilisateur.

Serveur web

Pour ce qui est du Backend de notre site, nous avons utilisé **Node.js** et **Express** (le fichier JavaScript principal est appelé "**app.js**" et se situe à la racine du projet).

Authentification et inscription

Un visiteur d'Hellociné peut s'inscrire sur le site grâce à section "**Inscription**" de l'application web. Lors de l'inscription, les mots de passe sont **chiffrés** pour être stocké dans le fichier "**users.json**" un utilisateur inscrit est stocké côté serveur. Il peut alors se connecter directement au site et commencer sa navigation. La persistance de l'authentification est obtenu grâce à l'initiation d'une session côté serveur avec la librairie "**express-session**".

CRUD et interaction avec le serveur

Nous avons stocké nos films et nos séries dans une variable en mémoire sous forme de liste.

Une première interaction avec le serveur est la récupération de la liste des items grâce à une **requête GET lors de la création** de la single page. La variable est alors stocké côté client avec Vue.js et côté serveur avec Node.js.

C : La création d'un film est effectuée grâce à une **requête POST** et en envoyant un **objet JSON**. Du côté du serveur, cet objet est récupéré et ajouté à la variable en mémoire. Le serveur renvoie alors une réponse avec un code 200. Le nouvel objet est donc ajouté dans la liste côté client (Vue.js) et côté serveur (Node.js).

R : La visualisation d'un film peut s'effectuer avec le bouton "**Voir**" . Le bouton va alors déclencher une événement pour attribuer l'index du film voulu à la variable "**currentFilmId**" et changer la page courante pour afficher le component "**viewFilm**". Chaque film dispose d'un attribut Index qui permet d'accéder à l'objet directement (sans parcourir toute la liste).

U : L'édition d'un film est disponible uniquement pour les utilisateurs authentifiées. Celle-ci s'obtient en cliquant sur le bouton "**Editer**". Une page d'édition apparaît alors. L'utilisateur peut éditer le film voulu et voir en direct une visualisation de ces changements. Lorsque l'utilisateur veut valider ces changements, il doit appuyer sur le bouton "**Edit**". Dès lors que ce bouton est pressé, une requête POST est envoyée au serveur avec le nouvel objet sous format JSON. La variable est alors mise à jour côté client et côté serveur.

D : La supression d'un film se fait via l'attribut Index de l'objet film. Grâce au bouton "Supprimer". Le bouton déclenche une requête POST avec le film en format JSON. Le serveur récupère alors l'index et supprime l'objet. Par ailleurs, côté client et côté serveur on lance une nouvelle indexation des films afin que l'attribut Index soit toujours égale au réel index dans la liste des films.

Persistance des données

Nous avons un fichier "**films.json**" qui répertorie une quinzaine de films et série qui sont chargés au lancement du serveur. La liste des films est récupérée par le client à la création la page (section **created**).

La **persistance des films** est donc limitée au stockage "**in-memory**" côté serveur. Nous avons privilégié d'autres fonctionnalités au détriment de celle-ci.

Cependant, nous avons bien implémenté une **persistance des données pour la partie authentification**. En effet, un visiteur qui s'inscrit verra son login et mot de passe sauvegarder à la suite des autres utilisateurs **dans le fichier** "**users.json**" (dossier '**db**').

ÉTAPES DE CONCEPTION

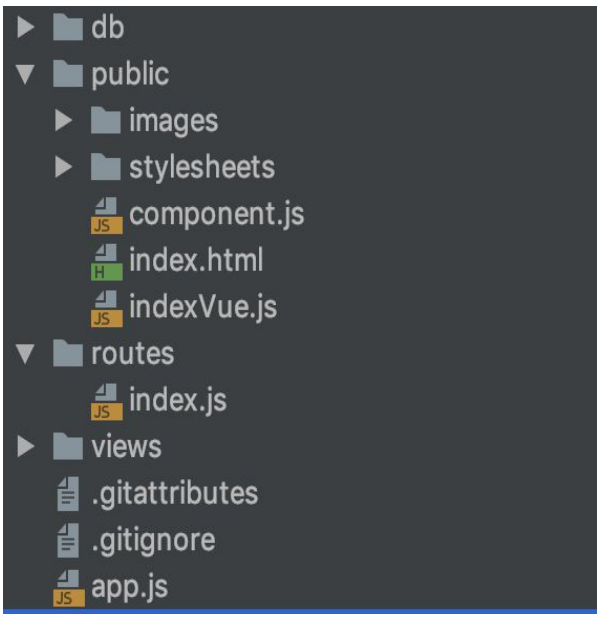
Conception et réalisation

Nous avons commencé par créer nos **pages sous forme HTML avec le CSS** associé. Nous avons pu créer la page d'accueil, l'inscription et la vue d'un film. Pour ajouter de la compatibilité entre les différents écrans (type responsive) nous avons décidé d'utiliser le **framework Bootstrap** afin d'avoir une structure solide pour commencer à implémenter notre propre feuille de style CSS.

Dans un second temps, nous avons étudié la structure et l'implémentation de Vue.js. Nous avons hésité à utiliser **Webpack** mais nous nous sommes résignés à utiliser **une structure simple** et efficace sans Webpack afin de ne pas se noyer dans l'information. Étant une application avec une unique page, nous avons découpé nos anciennes pages HTML en plusieurs "**component**" distinct afin de faciliter le changement de page: des components "statiques" qui sont présents sur chaque vue (*la barre de navigation*) et des components qui se masquent et qui apparaissent en fonction d'une variable (**currentPage**). Pour tester Vue.js avec nos pages nous utilisions des données en dur dans notre single page (initialisé dans la section **data** de notre **indexVue.js**)

En parallèle, nous avons mis en place notre **serveur Node.js**. et une arborescence de fichiers claire :

- **db** : les données sous format de JSON (utilisateurs et films)
- **images** : les images pour notre HTML
- **stylesheets** : nos feuilles de styles CSS
- **components.js** : la définition de nos components et de leurs méthodes.
- **index.html** : la page HTML contenant tous nos components ainsi que leur conditions d'affichage.
- **indexVue.js** : Définition des méthodes et des données pour l'application principale.
- **routes/index.js :** la définition de nos routes pour les requêtes GET et POST ainsi que leurs actions associées.
- **app.js :** fichier principale pour le serveur node et initiation de la session utilisateur.



Déploiement

Nous avons effectué notre déploiement sur le site **Glitch** en indiquant notre adresse de repository **GitHub**.

Vous pouvez retrouver Hellociné à cette adresse : https://faivrem-hellocine-7.glitch.me

DIFFICULTÉS RENCONTRÉES ET OBSERVATIONS

Authentification

Une des premières difficulté a été l'authentification. En effet, nous ne savions pas comment déclarer une nouvelle session et la maintenir durant la navigation de l'utilisateur. Nous avons décidé d'utiliser express-session et d'utiliser une requête GET lors de la création de la page afin de récupérer la session courante de l'utilisateur. De ce fait, si l'utilisateur ouvre une nouvelle page, le serveur renvoie la liste des films et la session de l'utilisateur avec son username.

Déploiement

Nous avions des problèmes avec les différents modules d'installation, en particulier bcrypt. Nous avons donc retirer le dossier node_modules de notre répertoire Git et nous avons utilisé la librairie **bcryptjs** afin d'éviter le gros nombre de dépendances. (https://www.npmjs.com/package/bcryptjs).

Nous avons aussi remarqué que le fichier du serveur Node devait être à la racine du projet Glitch pour que celui-ci soit fonctionnel.

Gestion des erreurs

Pour gérer les erreurs (mot de passe incorrect, utilisateur déjà existant, etc.), nous avons utilisé les codes statut des requêtes HTTP afin de fournir au Frontend des informations sous la forme d'alerte ou de message flash. Cependant, comme nous n'utilisons pas webpack, nous n'avons pas réussi à importer des modules déjà fait de message flash. Nous avons donc utilisé nos propres messages flash dans **indexVue.js** grâce à des variables appelées "**error**" et "**success**".

Observations et conclusion

Ce projet a été enrichissant puisqu'il nous a offert l'opportunité d'utiliser des technologies **récentes** et **puissantes** du développement web telle que Node.js et Vue.js.

Nous avons pu voir une approche différente du **templating** que celle offerte par d'autres moteurs de templating PHP.

Par ailleurs, nous avons pu observé une réactivité de la page web nettement amélioré grâce à une **application single page** via Vue.js qui permet une navigation fluide et **sans rechargement de page**. De plus grâce à **axios**, nous avons pu manipuler nos données avec de requête HTTP et les modifier en direct dans notre template.

L'utilisation de Vue.js, nous a aussi apporté des facilités dans la **gestion des événements** telle que le clic sur un bouton, mais aussi les sections **created,** nous permettant de lancer différentes actions au démarrage de la page, ou **computed** permettant d'obtenir une variable calculée (que l'on a utilisé pour filtrer notre liste de film lors d'une recherche).

Un grand point fort pour Vue.js au niveau du templating a été le découpage en **component** qui nous a permis de minimiser le code et améliorer la lecture de celui-ci, tout en définissant des méthodes propres à nos components.

Pour finir, nous avons pu utiliser les propriétés de **v-model** de Vue.js pour réaliser une visualisation en direct lors de l'édition d'un film.

Une limite de la single page a été le retour arrière. En effet, les flèches 'retour arrière' du navigateur ne permettent de revenir à la page précédente puisque l'application ne dispose que d'une seule page.