Elliot GAULIN
Technologies émergentes
420-5A3-VI gr. 01

APPLICATION DE MESSAGERIE INSTANTANÉE AVEC WEBSOCKET

Document d’analyse

Travail présenté à
M. Étienne Rivard

Département d’informatique
Cégep de Victoriaville
Le 15 décembre 2022

Buts du projet

Avec ce projet, mon but était principalement d'apprendre à utiliser WebSocket autant côté serveur que côté client. Pour ce faire, j'ai décidé de créer une application de messagerie instantanée. Les buts de cette application sera les suivants :

- Instaurer un système d'authentification afin de connaître l'émetteur du message.
- Permettre à un utilisateur d'envoyer des messages à un autre utilisateur.
- Seulement le destinataire du message devra recevoir le message.
- Le projet devra être prêt pour la remise le 15 décembre 2022.

Liste de concepts

- WebSocket :
  L'**API WebSocket** est une technologie évoluée qui permet d'ouvrir un canal de communication bidirectionnelle entre un navigateur (côté client) et un serveur. Avec cette API vous pouvez envoyer des messages à un serveur et recevoir ses réponses de manière événementielle sans avoir à aller consulter le serveur pour obtenir une réponse.[1]

- Express :
  Express est un cadriciel minimal et flexible pour node.js permettant le développement d'une application web. Il peut être à la fois utilisé pour faire des apis et pour générer des pages web avec Jade, son engin de modèle.[2]

- Node.js :
  Node.js est un logiciel permettant de rouler du code javascript côté serveur, à la manière de Python.

- Authentification :
  L'**authentification** est un processus permettant à un système informatique de s'assurer de la légitimité de la demande d'accès faite par une entité (être humain ou un autre système) afin d'autoriser son accès à des ressources du système (système d'exploitation, réseaux, applications…)[1] conformément au paramétrage du contrôle d'accès. Le système attribue à cette entité les données d'identité pour une session d'accès (ces attributs sont détenus par le système ou peuvent être fournis par l'entité lors du processus d'authentification). À partir des éléments issus de ces deux processus, l'accès aux ressources du système pourra être paramétré (contrôle d'accès).[3]

  Dans mon cas, elle sera utilisée pour me permettre de connaître l'identité de la personne se connectant à mon application et/ou qui à envoyer le message.

- React:
  React (aussi appelé **React.js** ou **ReactJS**) est une bibliothèque JavaScript libre développée par Facebook depuis 2013. Le but principal de cette bibliothèque est de faciliter la création d'application web monopage, via la création de composants dépendant d'un état et générant une page (ou portion) HTML à chaque changement d'état. [4]

---

[1] https://developer.mozilla.org/fr/docs/Web/API/WebSockets_API
[2] https://expressjs.com/
[3] https://fr.wikipedia.org/wiki/Authentification
[4] https://fr.wikipedia.org/wiki/React

Comparatifs de différentes technologies

Cadriciel côté serveur[5]

Cadriciel	Language	Pour	Contre	Choisi
Django	Python	● Flexible ● Structure bien définie ● Beaucoup de fonctionnalité déjà incluses	● Ne convient pas bien au petits projets ● Une seule requête à la fois ● Mauvaise connaissance du langage de ma pars	
Express	Javascript	● Même langage pour côté client et serveur (JS) ● Aucune règle strict ● Très grande quantité de package supplémentaire (NPM) ● Configuration simple et rapide	● Manque de standardisation ce qui peut nuire lorsque le projet prends de l'ampleur	X

J'ai choisi Express pour quelques raisons même si au départ j'avais commencé avec Django afin d'apprendre un nouveau framework. La raison principale est la rapidité de mise en route. En effet, avoir une base d'application fonctionnelle en Django m'a pris beaucoup plus de temps qu'en Express. De plus, mon aisance en Express m'a permis de faire la base de l'api plus rapidement pour pouvoir me concentrer sur la portion qui était réellement intéressante c'est-à-dire la portion sur WebSocket.

Pour le cadriciel côté serveur, j'avais aussi deux choix réalistes : Angular et React. Par contre, je n'ai pas pris le temps de faire une matrice de décision puisque je me suis rendu compte assez rapidement que je ne me sentais pas assez à l'aise avec Angular au moment où j'ai entamé le projet pour pouvoir être aussi efficace. Alors, j'ai choisi de faire l'interface utilisateur de mon projet en React en utilisant la librairie Material UI pour avoir à faire le moins de CSS possible et donc pouvoir me concentrer sur la logique des appels réseaux autant HTTP que WebSocket.

[5] https://mobiskill.fr/blog/conseils-emploi-tech/django-vs-express-quel-framework-back-end-choisir/

Enjeux et solutions

- Au départ, j'avais choisi Django avec pour principale bonne raison l'idée d'apprendre un nouveau langage et framework. Je me suis vite rendu compte de l'ampleur de charge de travail que celà allait m'apporter lorsqu'après un cours de 3h, je n'avais pas encore fini le processus de mise en route à 100%.
  La solution : Pour moi, la solution a été assez simple : j'ai choisi un autre cadriciel pour mon "back-end". Le cadriciel choisi comme mentionné plus haut à été Express en raison de sa simplicité, de sa rapidité de mise en route et puisqu'il s'agit d'un framework que je connais déjà assez bien pour être efficace dans celui-ci, ce qui m'a permis de passer plus de temps à bien comprendre WebSocket

- En développant mon API, j'avais instauré un système d'authentification Basic très rudimentaire puisque je ne souciais pas réellement de la sécurité de mon application pour ce projet ci, le but étant simplement d'utiliser le protocole WebSocket. Ainsi donc, je passais un jeton encodé en base64 dans l'entête de la requête de cette façon :
  ```
  {
          Authorization : Basic "jeton_en_base64"
  }
  ```
  Le problème est survenu lorsque j'essayais de faire la requête de connexion à mon serveur WebSocket, l'api Javascript ne me permettait pas de modifier les headers à ma guise. De plus, l'ancienne méthode d'authentification Basic de WebSocket était maintenant rendue obsolète.
  La solution : Afin de ne pas avoir à réécrire une grande partie de mon api, j'ai décidé de m'établir un standard : la première requête effectuée après la connexion doit obligatoirement contenir le jeton d'authentification sinon la connexion sera rompue. J'avais trouvé une autre méthode[6], plus robuste, pour faire cette authentification mais j'aurais dû réécrire une bonne portion de mon api en plus d'ajouter une table à ma base de données. Le temps m'étant compté, j'ai opté pour une méthode plus simple.

- Un autre problème auquel j'ai dû faire face a été la gestion de l'objet WebSocket dans l'application React. Il m'arrivait souvent que l'application essayait d'utiliser l'objet WebSocket après que la connexion ait été fermée pour une raison quelconque (mauvaise authentification, serveur web qui à planté, etc.).
  Solution : Pour corriger ce problème, j'ai placé l'objet dans l'état de mon composant React. Par la suite, j'ai écrit une méthode qui lançait un essai de connexion au serveur WebSocket. À chaque mise à jour de la page, je vérifiais l'état de mon objet. Si celui-ci était nul ou bien que la connexion était fermée, j'exécutais la méthode décrite précédemment. Cette méthode a bien entendu des défauts. Par exemple, mon application va faire constamment des essais de connexion au serveur WS même si celui-ci n'existe pas ou si la connexion est refusée.

[6] https://devcenter.heroku.com/articles/websocket-security

Conclusion

Pour finir, je suis très content de l’application produite, surtout avec les contraintes de temps imposées par la situation particulière du cours. J’ai grandement appris sur le protocole WebSocket alors mon but principal a été atteint. Aussi, j’ai beaucoup appris sur la gestion des appels à l’api dans une application web. En effet, j’ai appris à séparer mes requêtes afin d’améliorer le temps de chargement de la page ce qui rends l’application beaucoup plus agréable à utiliser. Par contre, il y a quelques fonctionnalités que j’aurais aimé avoir le temps d’implémenter dans l’application React comme le formulaire d’inscription et la possibilité d’envoyer un message à un usagé avec lequel aucune conversation n’a précédemment été créée. Pour ce qui est de mon avis à propos de WebSocket, je trouve qu’il s’agit d’une technologie très pratique et qui répond très bien à un besoin réel du monde du développement web. En effet, avant l’arrivée de cette technologie, il n’y avait pas de moyen d’avoir accès à des données en temps réel. La solution était auparavant de faire un appel à l’api après un intervalle de temps ce qui donnait lieu à une foule d' appels inutiles. De plus, il n’y avait pas de canal de communication à partir du serveur vers le client sans que le client n'ait spécifiquement fait de requête.