Rapport du TP AppliWeb

Adresse du dépôt Github : https://github.com/KerryanOPMace/appliweb

Adresse du serveur : https://serveur-8cdy.onrender.com

Adresse du site web : Discussion

Le projet est accessible en ligne via l'url ci-dessus, mais aussi en local si besoin. Il suffit de cloner le repo github, d'installer les dépendances avec *npm install* et de lancer le backend avec la commande *node index.js,* l'API sera alors à l'écoute via le port 8080. Puis en servant le front, vous pourrez accéder au service de messagerie en local.

Dépôt Github :

Le dépôt Github est composé de :

- Serveur -> l'api backend qui reçoit les requêtes et envoie les réponses
  - Index.js
  - Package.json
  - Package-lock.json
- Client -> le front qui permet à l'utilisateur de visualiser les messages
  - Index.html
  - Logiques.js
  - Styles.css
- Rapport -> le présent rapport

Ce dépôt Github est public. Il est lié à Render qui met à jour les versions déployées de l'api et du front dès qu'une nouvelle version du repo arrive (à chaque git push)

Render :

Sur Render, j'ai déployé le backend et le front sur 2 services différents. Le serveur en tant que *Web Service* et le front en tant que *Static Site*. J'ai utilisé la version gratuite du service qui alloue 0.1 CPU (ce qui est plus que suffisant pour faire fonctionner notre messaging service). Je ne sais pas si l'offre gratuite de Render est à durée limitée ou non, c'est pour cette raison que j'ai également implémenter la possibilité de faire tourner le service en local.

Choix d’implémentation :

Côté API… :

L'API est construite avec **Express.js** et offre plusieurs endpoints pour gérer les messages (ainsi qu’un compteur pour les premières questions du sujet). Elle utilise un middleware CORS pour accepter les requêtes de n'importe quel domaine. Les données sont envoyées et reçues au format **JSON**, avec des codes de statut pour gérer les erreurs.

Endpoints de l'API :

- /msg/nber : retourne le nombre de messages.
- /msg/getAll : retourne tous les messages.
- /msg/get/:id : retourne un message par son ID.
- /msg/del/:id : supprime un message par son ID.
- /cpt/query : retourne la valeur du compteur.
- /cpt/inc : incrémente le compteur (avec ou sans paramètre).
- /msg/post : ajoute un nouveau message avec un pseudo.

L'API écoute sur le port 8080 en développement local. Tous les endpoints, à l'exception de l'ajout de message, utilisent la méthode **GET** pour récupérer des données comme le nombre de messages, un message spécifique, ou la valeur du compteur. L'ajout de messages se fait via un endpoint **POST** afin d'envoyer des données (message et pseudo) dans le corps de la requête. Les données de l'application sont stockées de manière temporaire en mémoire, dans des tableaux JavaScript. Les messages sont conservés dans un tableau allMsgs qui contient des objets, chacun représentant un message avec son texte, le pseudo de l'auteur et la date de création.

...Et côté client :

Le tableau *msgs* est utilisé pour stocker les messages récupérés de l'API et afficher ces derniers dans l'interface utilisateur. La fonction *update()* permet de mettre à jour la liste des messages affichée sur la page, en créant un élément de liste (<li>) pour chaque message et en y insérant son contenu, son auteur et la date.

La fonction ajoutMessage() permet d'envoyer un nouveau message à l'API via une requête POST. Elle récupère le message et le pseudo de l'utilisateur à partir des champs correspondants, puis les envoie au serveur sous forme de JSON. Une fois le message envoyé, les champs sont réinitialisés.

La fonction *miseAJour()* est utilisée pour récupérer tous les messages de l'API avec une requête GET et mettre à jour le tableau msgs avec les nouveaux messages. Elle appelle ensuite la fonction update() pour rafraîchir l'affichage des messages. Cela implique donc qu'après avoir cliqué sur « Envoyer », il faut cliquer sur « Mettre à jour » pour voir les nouveaux messages apparaître.

La fonction *changerStyle()* permet de basculer entre le mode clair et le mode sombre en ajoutant ou en retirant des classes CSS du body.

Enfin, des écouteurs d'événements sont attachés aux boutons pour déclencher les fonctions appropriées lorsqu'un utilisateur interagit avec l'interface. L'application charge également les messages au démarrage en appelant *miseAJour()*