Antoine CASTEL

Rapport d'Analyse - Application de Chat

Architecture Générale

Cette application de chat repose sur une architecture client-serveur comme demandé dans le projet :

- **Frontend** : HTML/CSS/JavaScript vanilla (sans framework)
- **Backend** : Node.js avec Express.js
- **Hébergement** : Render (https://render.com/) freeplan
- **Communication** : API REST

Les Liens :

`Attention: après 15 minutes d'inactivité, le Backend est shutdown. En ouvrant le Frontend il faudra donc attendre quelques minutes pour que le Serveur Backend soit à nouveau up.`

- **Site Web** : https://cs-projet-architecture-applicative-rkp9.onrender.com/
- **Backend** : https://cs-projet-architecture-applicative.onrender.com

Structure de Données

Stockage des Messages

Les messages sont stockés dans une structure de données simple :

```
var allMsgs = [
  {
    "msg": "Hello World",
    "pseudo": "System",
    "date": new Date().toISOString()
  },
  // ...autres messages
];
```

Chaque message est un objet JavaScript avec trois propriétés : - `msg` : contenu du message - `pseudo` : nom d'utilisateur (avec "Anonymous" comme valeur par défaut) - `date` : horodatage au format ISO pour faciliter l'affichage relatif

Particularités du projet

1. **Stockage en mémoire** : Les messages sont stockés en mémoire sur le serveur, ce qui signifie qu'ils sont perdus en cas de redémarrage du serveur. Cette approche simplifie l'implémentation mais n'est pas adaptée à une application de production où une base de données serait nécessaire pour avoir une redondance.

2. **API REST simplifiée** : Toutes les opérations utilisent des requêtes GET, même pour l'ajout et la suppression de messages. Bien que cela ne soit pas conforme aux bonnes pratiques REST (qui utiliseraient POST, DELETE), cela simplifie l'implémentation.

Fonctionnalités

Côté Serveur

1. **Endpoints API** :
   - `/msg/getAll` - Récupère tous les messages
   - `/msg/post/:message?pseudo=name` - Ajoute un message
   - `/msg/del/:index` - Supprime un message
   - `/msg/get/:index` - Récupère un message spécifique
   - `/msg/nber` - Compte le nombre de messages
2. **Gestion CORS** : Configuration pour permettre les requêtes cross-origin, essentielle pour une application répartie sur différents domaines.

Côté Client

1. **Formatage des dates relatif** :
   - Transformation des timestamps ISO en formats lisibles comme "il y a 5 minutes"
   - Différents niveaux de granularité selon l'ancienneté ("à l'instant", "il y a X minutes/heures", "hier", etc.)
2. **Rafraîchissement automatique** :
   - Polling toutes les 30 secondes pour mettre à jour les messages
   - Choix de polling plutôt que WebSockets pour simplifier l'implémentation
3. **Thème clair/sombre** :
   - Persistance des préférences via localStorage
   - Transition fluide entre les thèmes
4. **Support des emojis** :
   - Sélecteur d'emojis personnalisé
   - Animation à l'insertion
5. **UX Travaillé** :
   - Animations pour l'ajout/suppression de messages
   - Indicateurs de chargement
   - Confirmation avant suppression
   - Gestion des erreurs avec feedback utilisateur