SOFIP
SE FORMER POUR L'AVENIR

Upload files MERN & Firebase

Le stockage d'images :

Stocker des images et des vidéos sur un serveur Node.js à l'aide de modules comme Multer ou dans une base de données au format blob présente quelques inconvénients par rapport à l'utilisation de services de stockage cloud dédiés tels que Firebase Storage :

- Évolutivité limitée : Si vous stockez des fichiers directement sur le serveur Node.js ou dans une base de données, vous pouvez rencontrer des limitations en termes d'évolutivité. À mesure que le nombre de fichiers et leur taille augmentent, cela peut affecter les performances et la capacité de votre serveur.
- Gestion du stockage : La gestion du stockage des fichiers sur le serveur Node.js ou dans une base de données peut être plus complexe et nécessiter une configuration et une maintenance supplémentaires. Vous devrez vous assurer que votre serveur dispose de suffisamment d'espace de stockage.
- Performance : Stocker des fichiers directement sur le serveur ou dans une base de données peut affecter les performances de votre application, en particulier si vous avez un grand nombre d'utilisateurs qui accèdent simultanément aux fichiers. Les temps de chargement peuvent être plus longs, ce qui peut entraîner une expérience utilisateur médiocre.

Cela peut être une solution si le nombre d'images ne va pas évoluer ou si vous utilisez des liens vers des images.

Meilleures pratiques ?

- Utilisez un service de stockage dédié : Plutôt que de stocker les fichiers directement sur votre serveur web, utilisez des services de stockage cloud. Ils sont conçus pour stocker et distribuer des fichiers volumineux de manière efficace.
- Compression d'images et de vidéos : Pour optimiser les performances de votre application web, envisagez de compresser les images et les vidéos avant de les stocker ou convertir dans un format approprié (webp ou avif). Cela réduit la taille des fichiers et améliore les temps de chargement.
- Stockage sécurisé : Assurez-vous que vos données sont sécurisées en utilisant des connexions HTTPS pour le transfert de données et chiffrez les fichiers stockés, si nécessaire.

Mettre en place architecture côté front avec un composant Upload dans un dossier component. Ce composant est importé dans App.

Firebase :

Avec un compte Gmail, se rendre sur Firebase : https://firebase.google.com/

Cliquez sur Get started ou allez dans la console et créer un projet. Le nommer, puis décocher Google Analytics. Créer le projet et continuer.

Cliquez sur le fragment pour ajouter notre application.

Donner un pseudo à l'appli et enregistrer. Cliquez sur 'aller à la console'. Dans l'interface du projet sur firebase cliquer sur créer puis storage. Choisir le mode test. Puis choisir la localisation la plus proche puis valider.

Lien pour la documentation pour uploader des fichiers avec Firebase : https://firebase.google.com/docs/storage/web/upload-files?hl=fr#full_example

Installer firebase avec React et react-router-dom. Copier ensuite le code avec votre configuration et copier le dans un fichier firebase.js à la source de src. Ajouter l'export d'app.

Nous importons les méthodes nécessaires pour notre code ainsi que l'application firebase

```
// import des méthodes nécessaires pour l'upload sur firebase
import {
  getStorage,
  ref,
  uploadBytesResumable,
  getDownloadURL,
} from "firebase/storage";

// import de notre application firebase
import app from "../firebase";
```

Nous créons 3 variables d'état pour stocker l'image, pour stocker le lien de cette image et pour la progression de l'envoi sur le serveur.

```
const [img, setImg] = useState(null);
const [imgLink, setImgLink] = useState("");
const [imgProgres, setImgProgress] = useState(0);
```

Dans un useEffect nous récupérons l'image dès qu'elle est renseignée et appelons la méthode uploadFile pour l'envoyer sur le serveur Firebase. En dépendance de ce hook, nous passons l'image au cas où il y aurait modification.

```
useEffect(() => {
  img && uploadFile(img);
}, [img]);
```

Nous allons décomposer la fonction uploadFile :

```
const storage = getStorage(app);
const fileName = new Date().getTime() + file.name;
const storageRef = ref(storage, "images/" + fileName);
const uploadTask = uploadBytesResumable(storageRef, file);
```

Nous initialisons le stockage sur Firebase puis créons un fichier unique en concaténant la date lors de la création et le nom du fichier.

Nous créons un référence dans le dossier images dans l'espace de stockage sur Firebase.

Et enfin nous commençons le téléchargement en précisant la référence et le fichier.

```
uploadTask.on(
  "state_changed",
  (snapshot) => {
    const progress =
      (snapshot.bytesTransferred / snapshot.totalBytes) * 100;
    console.log(progress);
    setImgProgress(Math.round(progress));
    switch (snapshot.state) {
      case "paused":
        console.log("Upload is paused");
        break;
      case "running":
        console.log("Upload is running");
        break;
      default:
        break;
    }
  },
```

Dans cette première partie, nous vérifions les changements d'état du téléchargement. La fonction de rappel prend un instantané (snapshot) de l'envoi en cours.

Puis nous calculons la progression en pourcentage et mettons à jour cette variable.

Le switch est optionnel et propose des log dans la console.

```
(error) => {
  console.log(error);
  switch (error.code) {
    case "storage/unauthorized":
      // User doesn't have permission to access the object
      console.log(error);
      break;
    case "storage/canceled":
      // User canceled the upload
      break;
    case "storage/unknown":
      // Unknown error occurred, inspect error.serverResponse
      break;
    default:
      break;
  }
},
```

Ci-dessus, nous proposons une gestion d'erreur. Nous pouvons catcher l'erreur et l'afficher.

```
  () => {
    getDownloadURL(uploadTask.snapshot.ref).then((downloadURL) => {
      console.log("DownloadURL - ", downloadURL);
      setImgLink(downloadURL.toString());
    });
  }
```

Lorsque cela s'est bien passé, on récupère l'url de l'image que l'on attribue, convertie en chaine de caractère, à sa variable d'état.

Créer une base de données files avec une collection images dans MongoDB.

Nous envoyons ce lien vers notre API :

```
const response = await fetch(`http://localhost:5000/api/images`, {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify({ img: imgLink }),
});
```

Si la requête s'est bien déroulée, nous récupérons l'image et vidons toutes les variables ainsi que l'input file que nous récupérons depuis le DOM.

```
if (response.ok) {
  const newImage = await response.json();
  console.log(newImage);
  setImg(null);
  setImgLink("");
  setImgProgress(0);
  const input = document.getElementById("img");
  input.value = "";
}
```

Le formulaire :

```
<div className="d-flex flex-column mb-20">
  <label htmlFor="img" style={{ color: "green" }} className="mb-10">
    Image:
  </label>
  {imgProgres > 0 ? "Uploading: " + imgProgres + "%" : ""}
  <input
    type="file"
    accept="image/*"
    id="img"
    onChange={(e) => setImg(() => e.target.files[0])}
  />
</div>
```

Nous affichons la progression avec une ternaire uniquement si elle a débuté.

Nous spécifions que nous autorisons uniquement les fichiers de type image et l'on écoute l'événement onChange pour récupérer l'image ou sa modification.

Nous créons un serveur avec l'architecture habituelle.

Récupération des images pour affichage :

Nous créons une nouvelle variable d'état allImg initialisé à un tableau vide.

Nous récupérons les liens des images de la base de données et l'attribuons à la variable.

```
async function getImages() {
  try {
    const response = await fetch(`http://localhost:5000/api/images`);
    if (response.ok) {
      const newImage = await response.json();
      console.log(newImage);
      setAllImg(newImage);
    }
```

Nous ajoutons l'image si la requête s'est bien déroulée.

```
setAllImg([...allImg, newImage]);
```

Puis si notre tableau d'images n'est pas vide, nous l'affichons

```
<div
  className="d-flex flex-row flex-wrap"
  style={{ minWidth: "1200px", margin: "0 auto" }}
>
  {allImg &&
    allImg.map((img) => (
      <img
        key={img._id}
        style={{
          width: "300px",
          maxHeight: "300px",
          marginRight: "20px",
        }}
        src={img.imgUrl}
        alt="img"
      />
    ))}
</div>
```