Documentation: Photothèque Node Js

Groupe 4: Etudiants

- FOMETIO MESMIN JORDAN
- ABA NKASSE MARCELLUS LORIC
- KEMO FRANC LOÏC

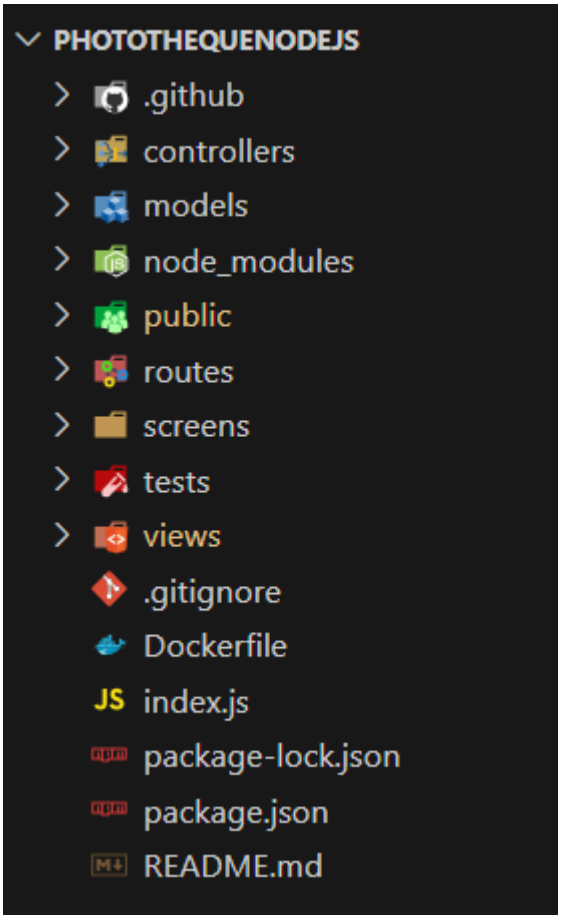
Il était question pour nous dans ce mini projet de réaliser une application web permettant de gérer une photothèque. L'application devant permettre essentiellement de créer des albums de photos dans lequel on pourrait ajouter, afficher, modifier et supprimer des photos.

I- Mise en place de l'environnement

Il est ici question de créer le projet en respectant l'arborescence de dossiers et de fichiers précédemment déclarée, d'installer les dépendances nécessaires pour la réalisation de notre projet.

```
PS C:\Users\marce\Desktop\IT-works\photothequeNodeJs> npm init -y
Wrote to C:\Users\marce\Desktop\IT-works\photothequeNodeJs\package.json:

{
  "name": "photothequenodejs",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
```

```
PS C:\Users\marce\Desktop\IT-works\photothequeNodeJs> npm install express body-parser mongoose ejs multer nodemon

added 144 packages, and audited 145 packages in 5s

20 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\marce\Desktop\IT-works\photothequeNodeJs>
```

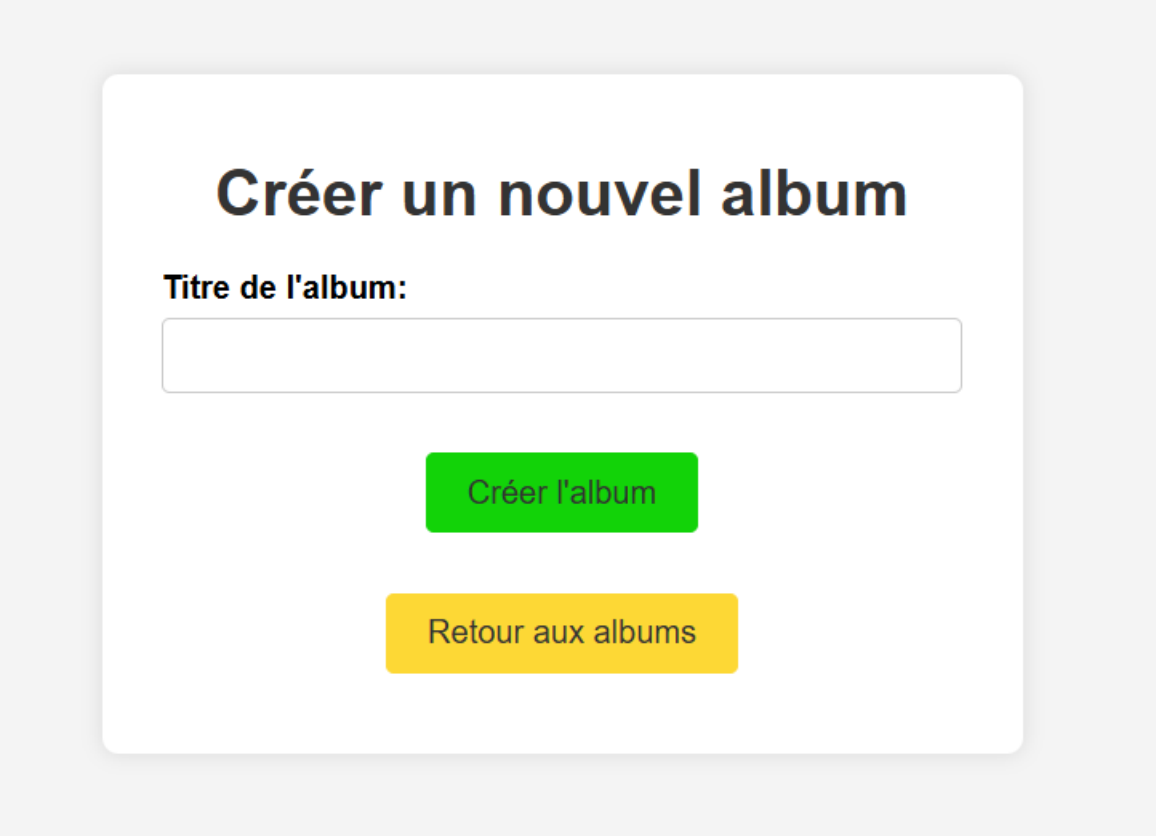


II- Développement des fonctionnalités

La mise sur pied de ces fonctionnalités se décline à chaque fois en deux étapes à savoir le développement de la route back-end et celui de la vue via respectivement js et ejs.

- Création d'un album :

```
// Afficher le formulaire pour ajouter un nouvel album
router.get("/new", (req, res) => {
  res.render("new_album", { cssFile: "new_img.css" });
});
// Ajouter un nouvel album
router.post("/new", async (req, res) => {
  try {
    const newAlbum = new Album({ title: req.body.title });
    await newAlbum.save();
    res.redirect("/albums");
  } catch (err) {
    //console.error(err);
    res.status(500).json({ error: "Une erreur est survenue ..." });
  }
});
```



- Ajouter une photo à un album :

Nous utiliserons ici et dans toutes les autres routes enregistrant les images le module multer de Node Js pour gérer la sauvegarde des photos ajoutés par l'utilisateur dans notre répertoire nommé public.

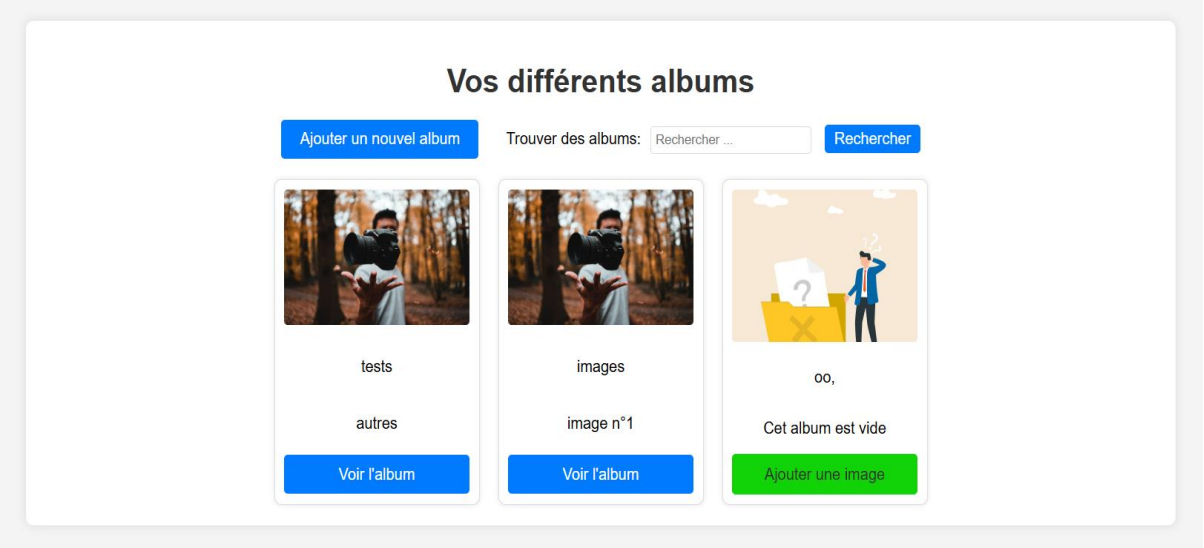
```
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, "public/uploads/");
  },
  filename: (req, file, cb) => {
    cb(null, Date.now() + "-" + file.originalname);
  },
});
const upload = multer({ storage: storage });      You, 5 days ago • premier commit
```

```
router.get("/newImgs/:albumId/:albumTitle", async (req, res) => {
  const albumId = req.params.albumId;
  const albumTitle = req.params.albumTitle;
  res.render("new_img", { albumId, cssFile: "new_img.css", albumTitle });
});
// Enregistrer les images dans un album précis
router.post("/imgs/:albumId", upload.single("image"), async (req, res) => {
  try {
    const albumId = req.params.albumId;
    const { title, description } = req.body;
    const imagePath = `/uploads/${req.file.filename}`;
    const date = Date.now();
    const newImg = new Img({
      title,
      description,
      date,
      imagePath,
      album: albumId,
    });
    await newImg.save();
    // Ajouter la nouvelle photo à l'album
    const album = await Album.findById(albumId);
    // Si l'album n'existe pas on ne sait enregistrer
    if (!album) {
      return res
        .status(404)
        .json({ message: "Aucun album correspondant trouvé" });
    }
    // Si l'album existe on n'y ajoute une image
    album.imgs.push(newImg);
    await album.save();
    res.redirect("/albums");
  } catch (err) {
    //console.error(err);
    res.status(500).json({ error: "Une erreur est survenue ..." });
  }
});
```

La vue permettant d'enregistrer une photo ainsi que ses métadonnées à un album précédemment choisi est la suivante:

- Affichage des différentes photos:

Nous afficherons également un filtre utile pour trier les photos dépendamment de leur titre. A noter que nous affichons les photos par album et chaque album est représenté par une miniature la dernière photo qu'elle contient.




```
const upload = multer({ storage: storage });
// Afficher les albums et leur dernière image
router.get("/", async (req, res) => {
  try {
    const searchQuery = req.query.search || ""; // Les caractères que doivent contenir les résultats
    const albums = await Album.find({ title: { $regex: searchQuery, $options: 'i' } })
                .populate('imgs')
                .exec();
    const results = albums.map((album) => {
      const lastImg =
        album.imgs.length > 0
          ? album.imgs[album.imgs.length - 1]
          : {
              imagePath: "/no-results.jpg",
              title: "Cet album est vide",
              date: new Date(),
            };
      return { ...album._doc, lastImg };
    });
    //console.log(results)
    res.status(200).render("albums", {
      albums: results,
      searchQuery,
      cssFile: "album.css",
    });
  } catch (err) {
    console.log(err);
    res.status(500).json({ error: "Une erreur est survenue ..." });
  }
});
```

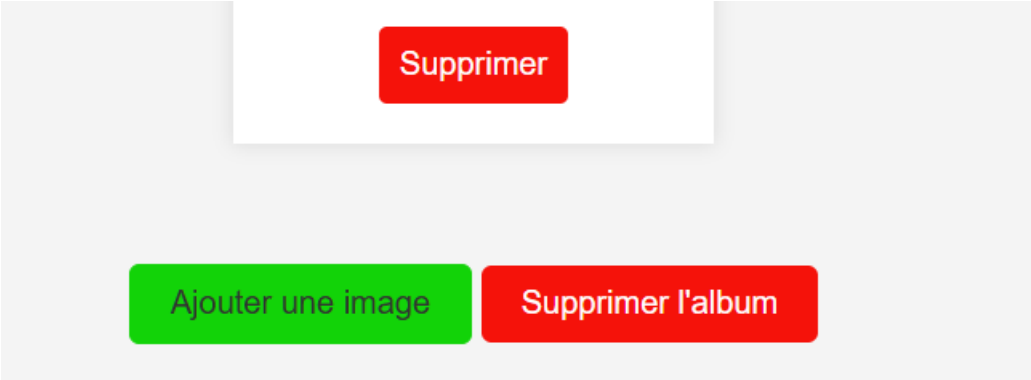
- Modification des Photos:

Il s’agit d’éditer les métadonnées d'une photo existante (son titre et sa description).


```
// Afficher le formulaire de mise à jour des métadonnées d'une image
router.get("/editImg/:imgId", async (req, res) => {
  try {
    const img = await Img.findById(req.params.imgId);
    if (!img) {
      return res.status(404).json({ message: "Image non trouvée" });
    }
    res.render("edit_img", { img, cssFile: "new_img.css" });
  } catch (err) {
    //console.error(err);      You, 5 days ago • premier commit
    res.status(500).json({ error: "Une erreur est survenue ..." });
  }
});
// Route utilisée par le formulaire pour mettre à jour des métadonnées d'une image
router.post("/editImg/:imgId", async (req, res) => {
  try {
    const { title, description } = req.body;
    const img = await Img.findByIdAndUpdate(
      req.params.imgId,
      { title, description },
      { new: true }
    );
    if (!img) {
      return res.status(404).json({ message: "Image non trouvée" });
    }
    res.redirect("/albums"); // Redirection vers la liste des albums après la modification
  } catch (err) {
    //console.error(err);
    res.status(500).json({ message: "Server error" });
  }
});
```

- Suppression d’une photo ou d’un album:

Implémentation de boutons permettant la suppression d'une image ou d'un album. A noter que le bouton supprimer un album n'apparaitra sur l'interface uniquement que si l'album contient au moins une image.




```
// Supprimer une image
router.post("/deleteImg/:imgId", async (req, res) => {
  try {
    const imgId = req.params.imgId;
    const img = await Img.findByIdAndDelete(imgId);
    if (!img) {
      return res.status(404).json({ message: "Image non trouvée" });
    }

    // Supprimer la référence de l'image dans l'album
    await Album.updateOne({ imgs: imgId }, { $pull: { imgs: imgId } });

    res.redirect("/albums"); // Redirection vers la liste des albums mise à jour
  } catch (err) {
    //console.error(err);
    res.status(500).json({ error: "Une erreur est survenue ..." });
  }
});
// Route pour supprimer un album (l'interface ne supprime un album que si celui ci est vide)
router.post('/delete/:id', async (req, res) => {
  try {
      const albumId = req.params.id;
      await Album.findByIdAndDelete(albumId);
      res.redirect('/albums');
  } catch (error) {
      res.status(500).send(error.message);
  }
});
```

Notre travail est disponible à: https://github.com/Loric-it/photoapp.git