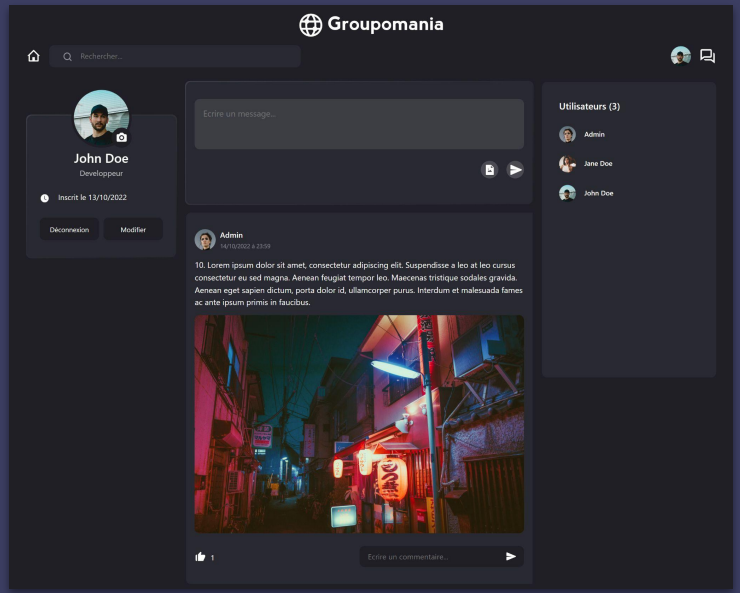
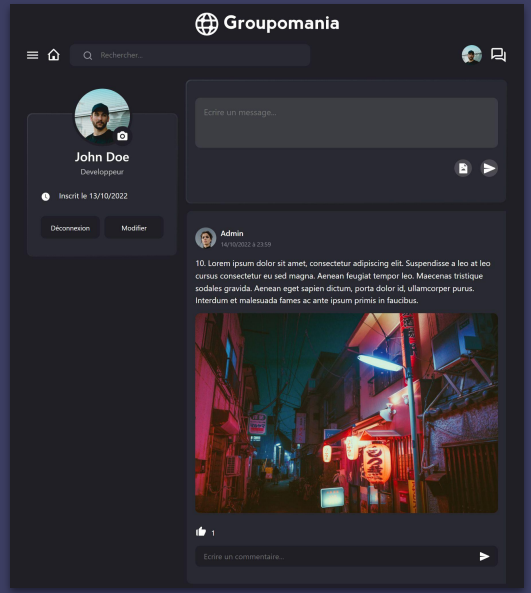
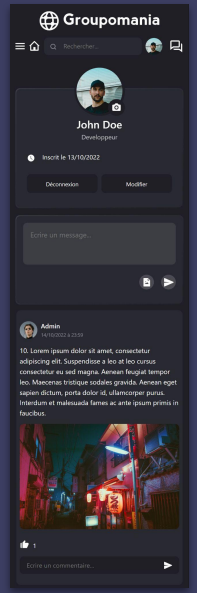
Groupomania







Contexte

- Construire un réseau social interne pour une entreprise
- Réaliser : Backend, Frontend, Base de données

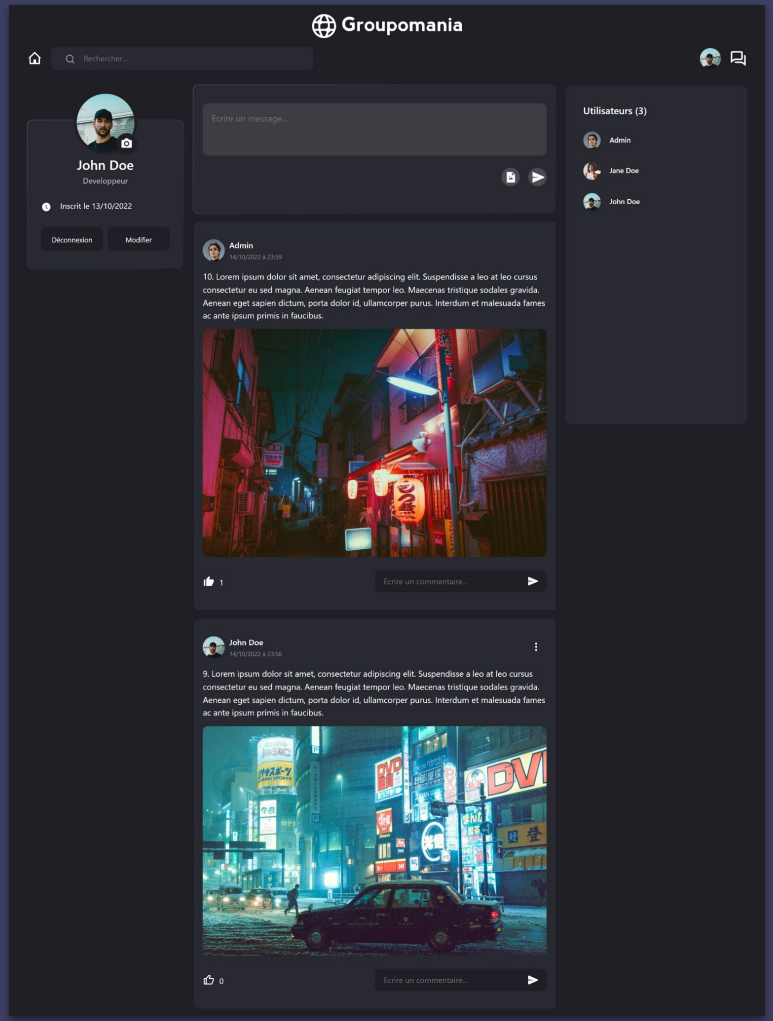
=> Framework frontend : **React**
=> Database : **mongoDB**

Attentes

- **Design** : responsive, logo, police “Lato”, reste libre ;
- **Inscription** : données sécurisées dans la database ;
- **Connexion** : persistante jusqu’à la déconnexion ;
- **Déconnexion** : retour à l’écran de connexion ;
- **Postes** : créer/modifier/supprimer un post (texte/image) ;
- **Likes** : possibilité de liker les postes ;
- **Admin** : peut modifier/supprimer tous les postes.

Présentation

- Démonstration des fonctionnalités
- Présentation du code
- Bilan & Axes d’amélioration

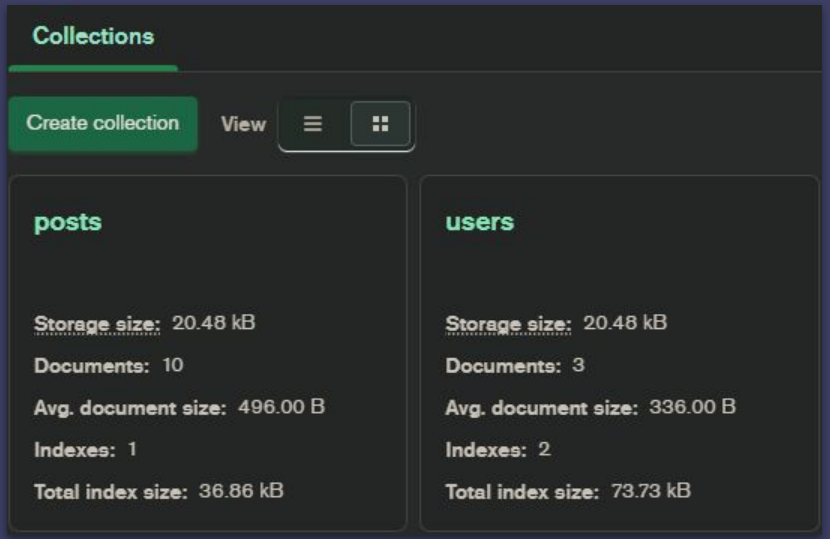


1. Démonstration

2. Code : intro

Database - MongoDB

MongoDB Compass




```
_id: ObjectId('63487d4812a70e0c41e53d94')
userId: "63487bfe12a70e0c41e53d57"
desc: "1. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendiss…"
image: "16657808117717.jpg"
likes: Array
  0: "63487dd012a70e0c41e53da2"
  1: "63487bfe12a70e0c41e53d57"
  2: "63487f0312a70e0c41e53db4"
createdAt: 2022-10-13T21:04:08.214+00:00
updatedAt: 2022-10-15T00:02:21.910+00:00
__v: 0
```

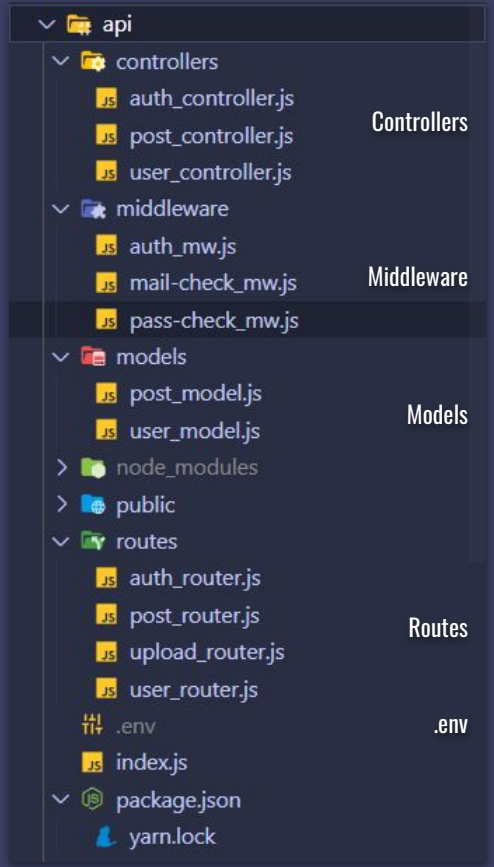

Exemple Post


```
_id: ObjectId('63487f0312a70e0c41e53db4')
admin: false
email: "e3f121a822aada57ae5306e6e7ddc1e42f12f474ec5f079770664e8dbf8c3057"
password: "$2a$10$nnmW9T4C3/6iQ18m7jOuwu9kW3Kkxe9.h0eQjzC3KD7Ofc9bOn1rG"
firstname: "John"
lastname: "Doe"
profilePicture: "1665929427224user(2).jpg"
about: "Developpeur"
createdAt: 2022-10-13T21:11:31.528+00:00
updatedAt: 2022-10-16T14:10:27.877+00:00
__v: 0
```


Exemple User

Backend - Rest Api

Structure



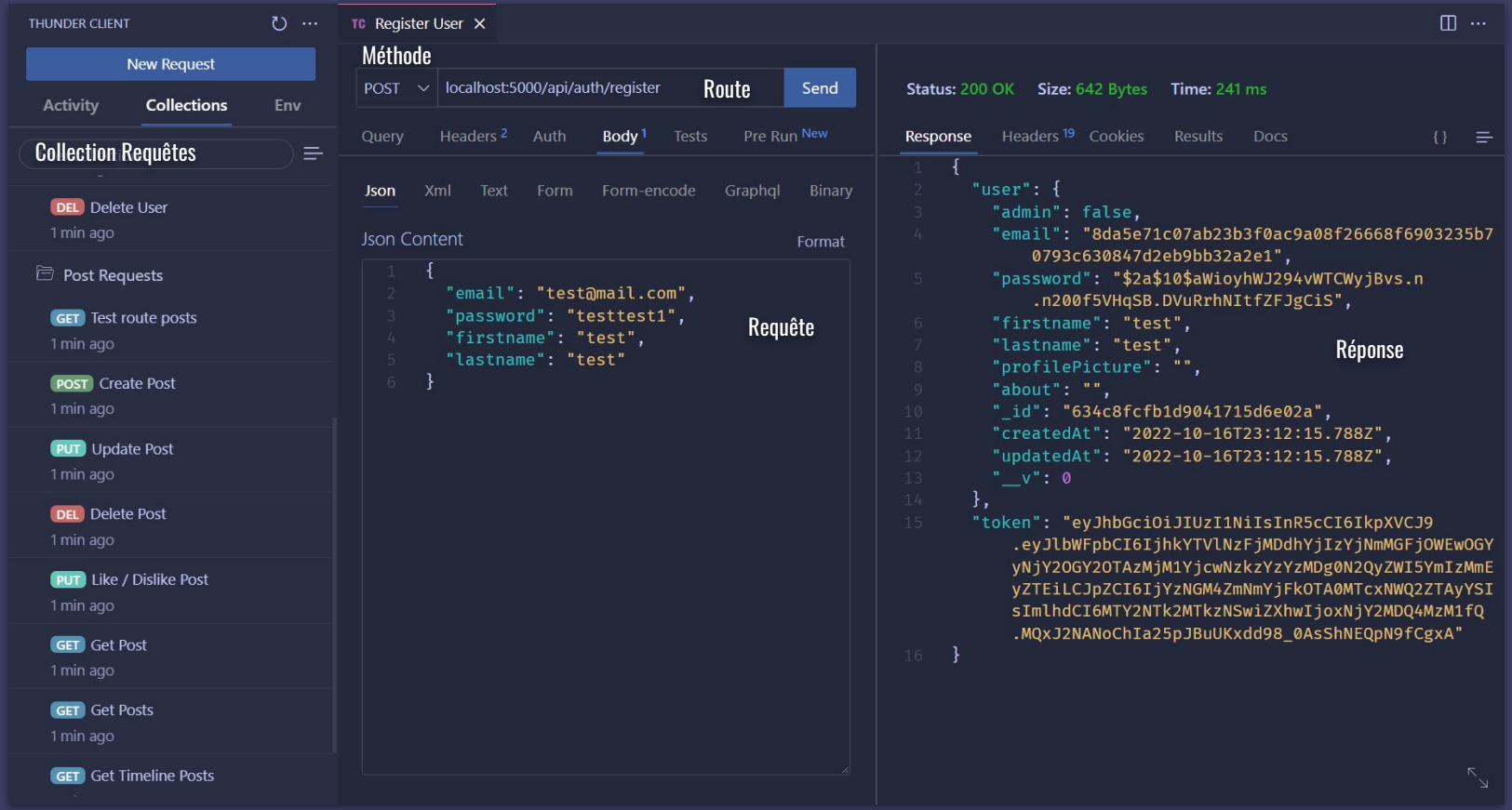
package.json

```
"dependencies": {
    latest
    "bcryptjs": "^2.4.3",          Hashage password
    latest
    "cors": "^2.8.5",              HTTP Headers
    latest
    "crypto-js": "^4.1.1",         Cryptage Email
    latest
    "dotenv": "^16.0.3",           Données sensibles
    latest
    "express": "^4.18.2",          Framework
    latest
    "helmet": "^6.0.0",            Pack Sécurité (XSS...)
    latest
    "jsonwebtoken": "^8.5.1",      Authentification
    latest
    "mongoose": "^6.6.5",          Librairie MongoDB
    latest
    "mongoose-unique-validator": "^3.1.0",
    latest prerelease ^1.4.5-lts.1
    "multer": "^1.4.5-lts.1",      Upload images
    latest
    "nodemon": "^2.0.20",          auto restart server
    latest
    "password-validator": "^5.3.0",
    latest
    "path": "^0.12.7",             Fichiers locaux
    latest
    "validator": "^13.7.0"         Contrôle email
}
```

index.js

```
import dotenv from "dotenv";
import express from "express";
import mongoose from "mongoose";                    import Packages
import helmet from "helmet";
import cors from "cors";
// Routes
import auth_router from "./routes/auth_router.js";
import user_router from "./routes/user_router.js";  import Routers
import post_router from "./routes/post_router.js";
import upload_router from "./routes/upload_router.js";

//? App
dotenv.config();
const app = express();
const PORT = process.env.PORT || 5000;

//? Middleware
app.use(helmet({ crossOriginResourcePolicy: { policy: "same-site" } }));
app.use(cors());
app.use(express.json());                            Middlewares
// Local images
app.use(express.static("public"));
app.use("/images", express.static("images"));
// Routes
app.use("/api/auth", auth_router);
app.use("/api/users", user_router);                 Appel Routes
app.use("/api/posts", post_router);
app.use("/api/upload", upload_router);

//? Start
const connectDB = (uri) => {
    return mongoose.connect(uri);
};
const start = async () => {
    try {                                           Database
        await connectDB(process.env.MONGO_URI);
        app.listen(PORT, () =>                      Lancement
            console.log(`Server is listening on port ${PORT}`)
        );
    } catch (error) {
        console.log({ message: error.message });
    }
};
start();
```

Backend - Thunder Client



Frontend - React : Structure

```
{
  "proxy": "http://localhost:5000",
  "name": "starter",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@iconscout/react-unicons": "^1.1.6",
    "@mantine/core": "^5.5.5",
    "@mantine/hooks": "^5.5.5",
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^14.4.3",
    "axios": "^1.1.3",
    "date-fns": "^2.29.3",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "6.4.2",
    "react-router-hash-link": "^2.4.3",
    "react-scripts": "5.0.1",
    "web-vitals": "^3.0.3"
  },
```

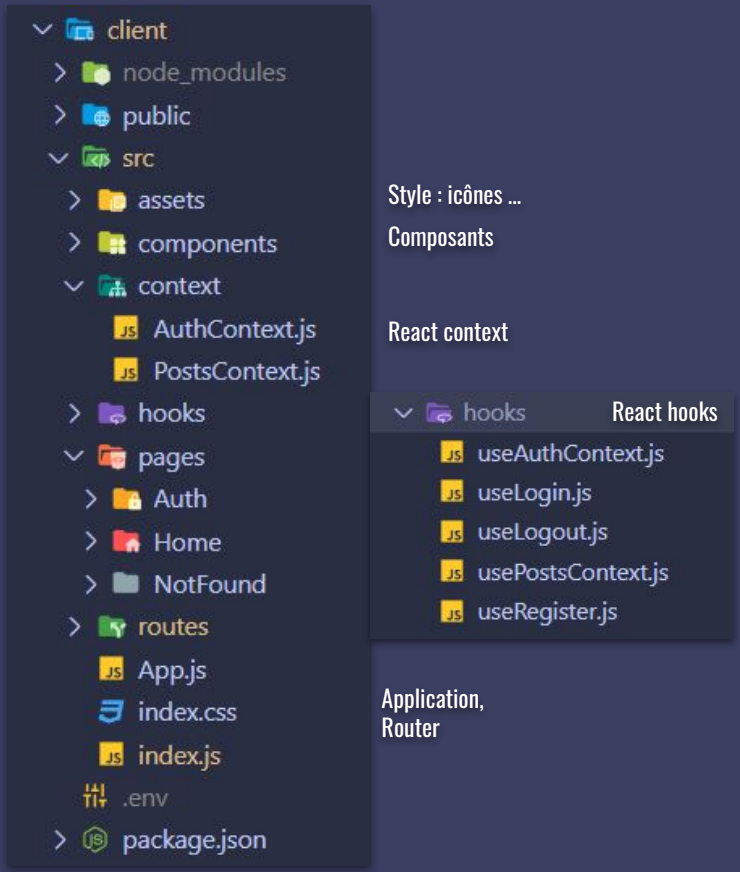
Librairie modals

Requêtes http

Affichage dates

Framework React + Router

Analyser performances



Style : icônes ...

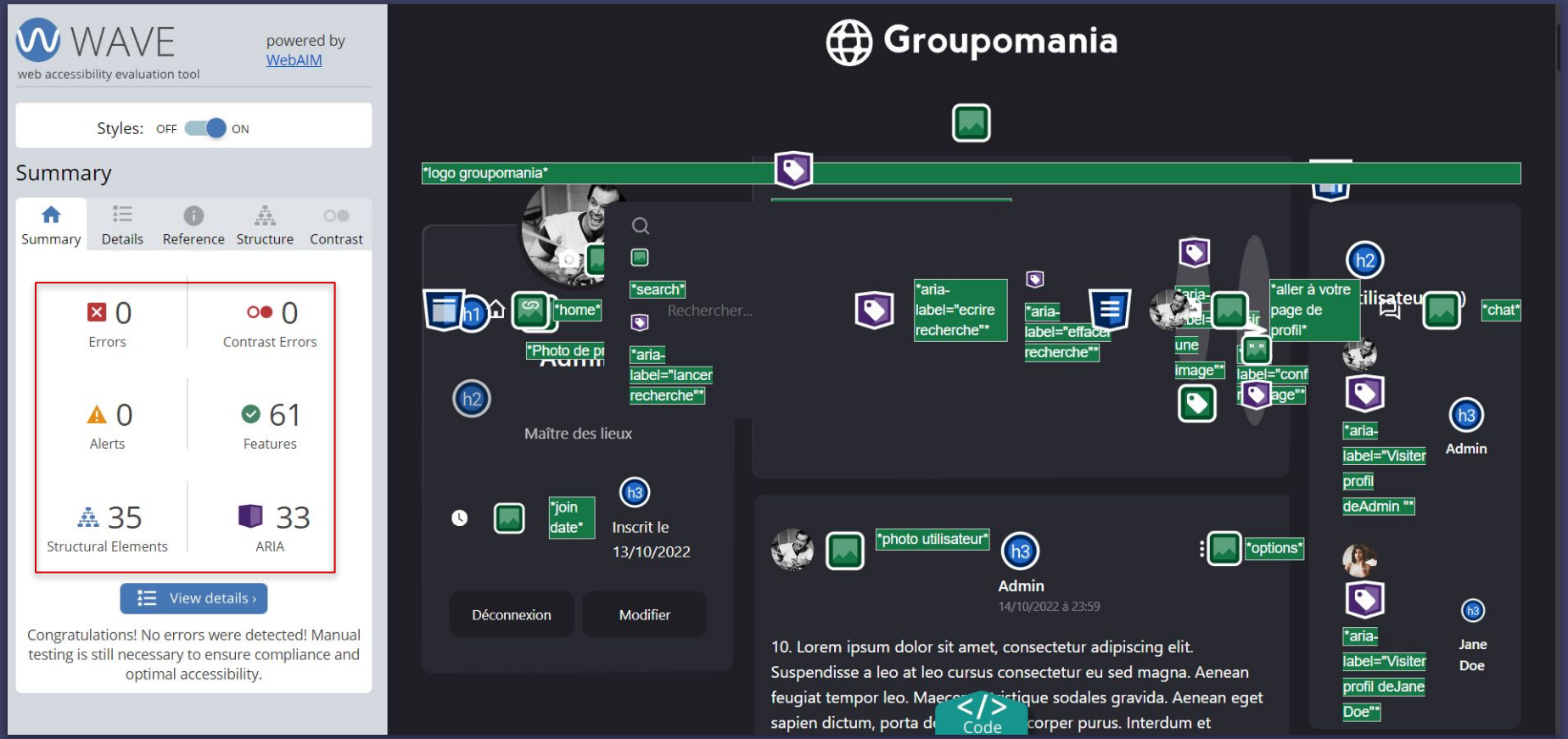
Composants

React context

React hooks

Application, Router

Frontend - Accessibilité : WCAG



Frontend - React : Router (1)

src/public/index.html

```html
<!DOCTYPE html>
<html lang="fr">

<head>
  <meta charset="utf-8" />
  <link rel="icon" href="./favicon.png" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <meta name="theme-color" content="#000000" />
  <meta name="description" content="Web site created using create-react-app" />
  <title>Groupomania</title>
</head>

<body>
  <div id="root"></div>
</body>

</html>
```

Root ->

src/index.js

```jsx
import React from "react";
import ReactDOM from "react-dom/client";
import { BrowserRouter } from "react-router-dom";
// Context
import { PostsContextProvider } from "./context/PostsContext";
import { AuthContextProvider } from "./context/AuthContext";
// Utilities
import { MantineProvider } from "@mantine/core";
import App from "./App";
import "./index.css";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
  <MantineProvider theme={{ colorScheme: "dark" }} withGlobalStyles>
    <AuthContextProvider>
      <PostsContextProvider>
        <BrowserRouter>
          <App />
        </BrowserRouter>
      </PostsContextProvider>
    </AuthContextProvider>
  </MantineProvider>
  </React.StrictMode>,
);
```

Context

Context
Router
Application
Router
Context

Frontend - React : Router (2)

src/app.js

```
import React from "react";
import { useAuthContext } from "./hooks/useAuthContext";
import Router from "./routes";
import Auth from "./pages/Auth/Auth";

function App() {
  const { user } = useAuthContext();

  return (
    <div className="app">
      <div className="container">{user == null ? <Auth /> : <Router />}</div>
    </div>
  );
}

export default App;
```

import : user context

Condition d'affichage

src/routes/index.js

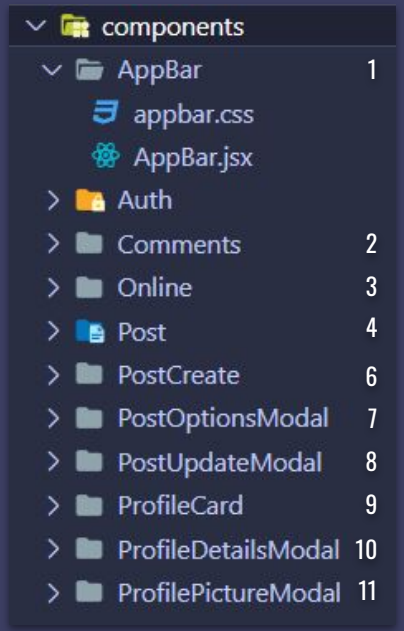
```
import { Routes, Route } from "react-router-dom";
// Components
import AppBar from "../components/AppBar/AppBar";
// Pages
import Home from "../pages/Home/Home";
import NotFound from "../pages/NotFound/NotFound";

const Router = () => {

  return (
    <>
      <AppBar />
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="*" element={<NotFound />} />
      </Routes>
    </>
  );
};

export default Router;
```

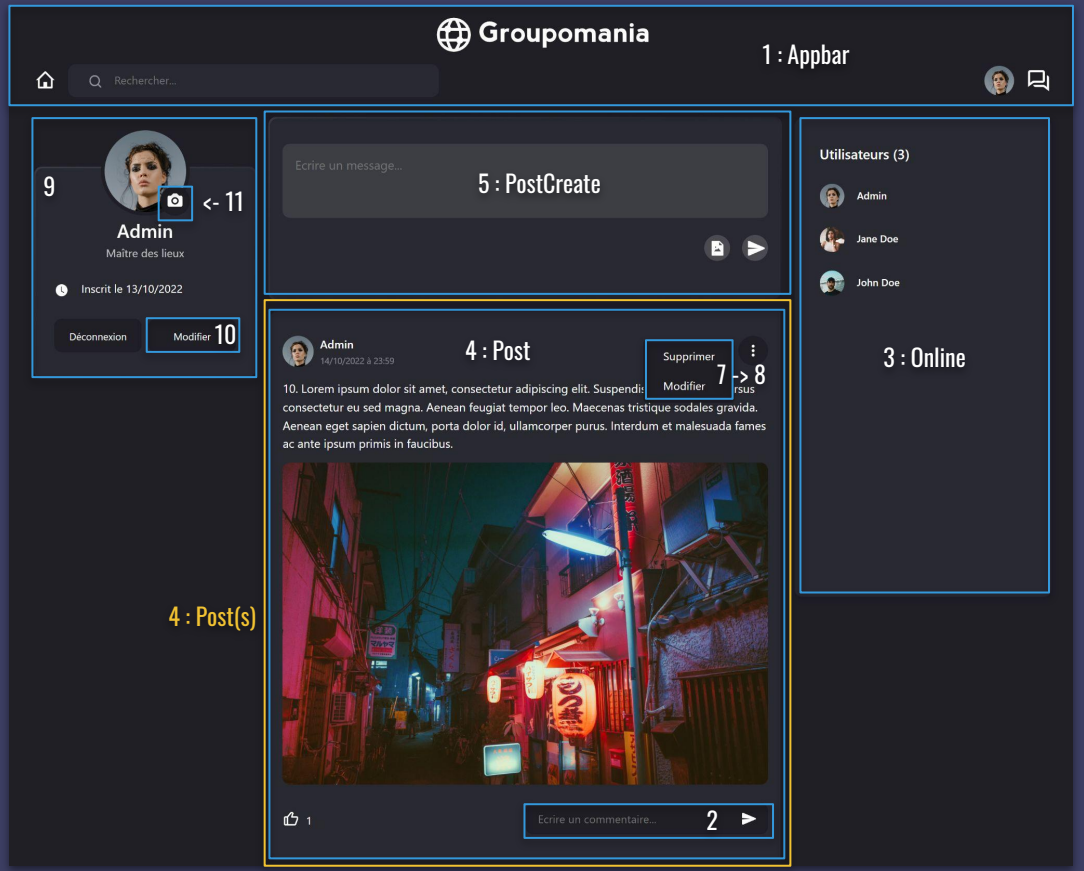
Barre de navigation

Accueil

404

Frontend - React : Composants





2. Code : Authentification

Inscription - Backend (1)

Model : User

```mongoose
import mongoose from "mongoose";
const UserSchema = new mongoose.Schema(
    {
        admin: {
            type: Boolean,
            default: false,
        },
        email: {
            type: String,
            required: [true, "Please provide email"],
            unique: true,
            lowercase: true,
        },
        password: {
            type: String,
            required: [true, "Please provide password"],
            minlength: 8,
        },
        firstname: {
            type: String,
            required: [true, "Please provide firstname"],
            minlength: 2,
            maxlength: 20,
        },
        lastname: {
            type: String,
            required: [true, "Please provide lastname"],
            minlength: 2,
            maxlength: 20,
        },
        profilePicture: {
            type: String,
            default: "",
        },
        about: {
            type: String,
            maxlength: 20,
            default: "",
        },
    },
    { timestamps: true }
);
const UserModel = mongoose.model("User", UserSchema);
export default UserModel;
```

Mongoose

admin ?

mail

password

prenom

nom

image profil

description

Controller User : register

```
import User from "../models/user_model.js";
import bcrypt from "bcryptjs";
import jwt from "jsonwebtoken";
import cjs from "crypto-js";

//? Register
export const register = async (req, res) => {
    // email
    const cryptedMail = cjs.HmacSHA256(req.body.email, process.env.CJS_KEY);
    req.body.email = cryptedMail;
    // password
    const salt = await bcrypt.genSalt(10);
    const hashedPass = await bcrypt.hash(req.body.password, salt);
    req.body.password = hashedPass;

    try {
        let user = await User.findOne({ email: cryptedMail.toString() });
        if (user) {
            return res.status(400).json({ message: "Email already in use" });
        }
        user = await User.create({ ...req.body });

        const token = jwt.sign(
            {
                email: user.email,
                id: user._id,
            },
            process.env.JWT_KEY,
            { expiresIn: process.env.JWT_TIME }
        );
        res.status(200).json({ user, token });
    } catch (error) {
        res.status(500).json({ message: error.message });
    }
};
```

Imports : Model, hash mot de passe, crypt email, token authentification

Cypter email

Hasher password

Vérifier si email déjà utilisée

Créer utilisateur

Créer token authentification (connexion)

Inscription - Backend (2)

Middleware : valider password

```
import passwordValidator from "password-validator"

const passwordSchema = new passwordValidator();

passwordSchema
    .is().min(8)                                    // Minimum length 8
    .is().max(32)                                   // Maximum length 100
    .has().uppercase()                              // Must have uppercase letters
    .has().lowercase()                              // Must have lowercase letters
    .has().digits(1)                                // Must have at least 1 digits
    .has().not().spaces()                           // Should not have spaces
    .is().not().oneOf(['Passw0rd', 'Password123', '123456']); // Blacklist these values

const validPassword = (req, res, next) => {
   if (passwordSchema.validate(req.body.password)) {
      next();
   } else {
      return res
         .status(403)
         .json({
            error: `Le mot de passe n'est pas assez fort`,
         });
   }
};
```

Paramètres

Méthode

Middleware : valider email

```
import validator from "validator"

const validEmail = (req, res, next) => {
   const { email } = req.body;
   if (validator.isEmail(email)) {
      next();
   } else {
      return res
         .status(403)
         .json({ error: `${email} n'est pas un email valide` });
   }
};
```

Méthode

Router : Authentification

```
import express from "express";
import { login, register } from "../controllers/auth_controller.js";
import mail from "../middleware/mail-check_mw.js";
import pass from "../middleware/pass-check_mw.js";

const router = express.Router();

//? Routes
router.post("/register", mail, pass, register);
router.post("/login", login);

export default router;
```

Controller

Middleware

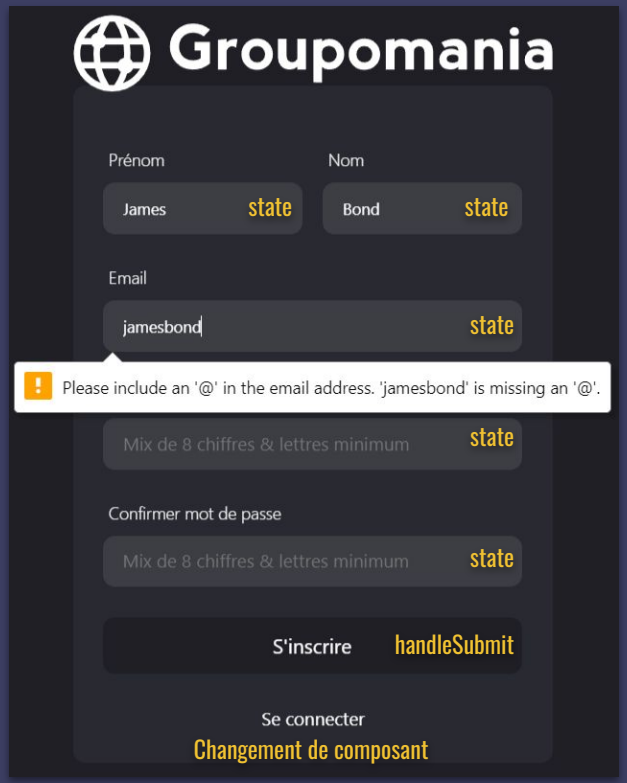
Middleware

Database : User

```
_id: ObjectId('63487f0312a70e0c41e53db4')
admin: false
email: "e3f121a822aada57ae5306e6e7ddc1e42f12f474ec5f079770664e8dbf8c3057"
password: "$2a$10$nnmW9T4C3/6iQ18m7jOuwu9kW3Kkxe9.h0eQjzC3KD7Ofc9bOn1rG"
firstname: "John"
lastname: "Doe"
profilePicture: "1665929427224user(2).jpg"
about: "Developpeur"
createdAt: 2022-10-13T21:11:31.528+00:00
updatedAt: 2022-10-16T14:10:27.877+00:00
__v: 0
```

Inscription - Frontend

Composant : Register



Formulaire d'inscription

```jsx
const Register = ({ setIsRegistering }) => {
  const [firstname, setFirstname] = useState("");
  const [lastname, setLastname] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [passwordConfirm, setPasswordConfirm] = useState("");

  const { register, error, isLoading } = useRegister();

  const handleSubmit = async (e) => {
    e.preventDefault();
    await register(firstname, lastname, email, password);
  };

  return (
    <form onSubmit={handleSubmit} className="register">
      <img src={logo} alt="logo" className="auth-logo" />

      <div className="firstname-lastname-container">
        <div className="name-wrapper">
          <label htmlFor="firstname">Prénom</label>
          <input
            required
            type="text"
            placeholder="James"
            value={firstname}
            onChange={(e) => setFirstname(e.target.value)}
            pattern="^(?![\s.]+$)[A-zÀ-ú\s\-]{1,25}$"
          />
        </div>
```

STATES

appel du Hooks "register"

requis

useState

contrôle

Inscription - Frontend (2)

Hook : useRegister / inscription

```
import { useState } from "react";
import { useAuthContext } from "./useAuthContext";

export const useRegister = () => {
   const [error, setError] = useState(null);
   const [isLoading, setIsLoading] = useState(null);
   const { dispatch } = useAuthContext();

   const register = async (firstname, lastname, email, password) => {
      setIsLoading(true);
      setError(null);

      const response = await fetch('/api/auth/register', {
         method: "POST",
         headers: { "Content-type": "application/json" },
         body: JSON.stringify({firstname, lastname, email, password}),
      });
      const json = await response.json();
      if (!response.ok) {
         setIsLoading(false);
         setError(json.error);
      }
      if (response.ok) {
         // save the user to local storage
         localStorage.setItem("user", JSON.stringify(json));
         // update auth context
         dispatch({type: 'LOGIN', payload: json})
         setIsLoading(false)
      }
   };
   return {register, isLoading, error}
};
```

useState + Context

Données récupérés

req POST api

Connexion

Sauvegarde user dans localStorage

User context : dispatch

Context : userContext / utilisateur

```
import { createContext, useReducer, useEffect } from "react";
import Auth from "../pages/Auth/Auth";

export const AuthContext = createContext();

export const authReducer = (state, action) => {
   switch (action.type) {
      case "LOGIN":
         return { user: action.payload, };
      case "LOGOUT":
         return { user: "null" }
      case "UPDATE":
         return { user: action.payload };
      default:
         return state;
   }
};

export const AuthContextProvider = ({ children }) => {
   const [state, dispatch] = useReducer(authReducer, { user: null });

   useEffect(() => {
      const user = JSON.parse(localStorage.getItem("user"));

      if (user) {
         dispatch({ type: "LOGIN", payload: user });
      }
   }, []);
   return (
      <AuthContext.Provider value={{ ...state, dispatch }}>
         {children}
      </AuthContext.Provider>
   );
};
```

Connexion

Déconnexion

Modifier

Persistance de la connexion

localStorage

Connexion

Appliquer dans le root de src/index.js

Connexion - Backend

User controller : Login

```
//? Login
export const login = async (req, res) => {
   const { email, password } = req.body;

   if (!email || !password) {
      return res.status(400).json({ message: "All fields must be filled" });
   } else {
      const cryptedMail = cjs.HmacSHA256(req.body.email, process.env.CJS_KEY);
      try {
         const user = await User.findOne({
            email: cryptedMail.toString(),
         });

         if (user) {
            const validpass = await bcrypt.compare(password, user.password);

            if (!validpass) {
               res.status(400).json("Incorrect password");
            } else {
               const token = jwt.sign(
                  {
                     mail: user.email,
                     id: user._id,
                  },
                  process.env.JWT_KEY,
                  { expiresIn: process.env.JWT_TIME }
               );
               res.status(200).json({ user, token });
            }
         } else {
            res.status(404).json("Incorrect email");
         }
      } catch (error) {
         res.status(500).json({ message: error.message });
      }
   }
};
```

Requête frontend

Crypter email

Correspondance email database ?

Correspondance password ?

Créer un token d'authentification

Récupérer user + token

User Router

```
import express from "express";
import { login, register } from "../controllers/auth_controller.js";
import mail from "../middleware/mail-check_mw.js";
import pass from "../middleware/pass-check_mw.js";

const router = express.Router();

//? Routes
router.post("/register", mail, pass, register);
router.post("/login", login);

export default router;
```

Connexion

Connexion - Frontend

Composant : Login

Formulaire de connexion

```jsx
const Login = ({ setIsRegistering }) => {
   const [email, setEmail] = useState("");
   const [password, setPassword] = useState("");
   const { login, error, isLoading } = useLogin();

   const handleSubmit = async (e) => {
      e.preventDefault();
      await login(email, password);
   };

   return (
      <form onSubmit={handleSubmit} className="login">
         <img src={logo} alt="logo" className="auth-logo" />

         <label htmlFor="login-email">Email</label>
         <input
            type="email"
            placeholder=""
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            required
         />

         <label htmlFor="login-password">Mot de passe</label>
         <input
            required
            type="password"
            placeholder=""
            value={password}
            onChange={(e) => setPassword(e.target.value)}
         />
         <button disabled={isLoading} type="submit">
            Se connecter
         </button>
         {error && <div classNme="error">{error}</div>}
         <p>
            <span onClick={() => setIsRegistering(true)}>S'inscrire</span>
         </p>
      </form>
   );
};
```

STATES

Appel Hook Login

Récupérer email

requis

requis

Récupérer pass

Hook : useLogin / connexion

```jsx
import { useState } from "react";
import { useAuthContext } from "./useAuthContext";

export const useLogin = () => {
   const [error, setError] = useState(null);
   const [isLoading, setIsLoading] = useState(null);
   const { dispatch } = useAuthContext();

   const login = async (email, password) => {
      setIsLoading(true);
      setError(null);

      const response = await fetch('/api/auth/login', {
         method: "POST",
         headers: { "Content-type": "application/json" },
         body: JSON.stringify({ email, password}),
      });
      const json = await response.json();
      if (!response.ok) {
         setIsLoading(false);
         setError(json.error);
      }
      if (response.ok) {
         // save the user to local storage
         localStorage.setItem("user", JSON.stringify(json));
         // update auth context
         dispatch({type: 'LOGIN', payload: json})
         setIsLoading(false)
      }
   };
   return {login, isLoading, error}
};
```

fetch POST : envoi de la requête

Sauvegarde LocalStorage

Dispatch -> UserContext

Local Storage

Key	Value
user	{"user":{"_id":"63487f0312a70e0c41e53db4","admin":false,"em...

{user: {_id: "63487f0312a70e0c41e53db4", admin: false,…},…
token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJtYWlsIjo
user: {_id: "63487f0312a70e0c41e53db4", admin: false,…}

Déconnexion

Composant : Profile Card

Actions ...

```
<button onClick={handleLogout}>Déconnexion</button>
```

```
//? Logout
const { logout } = useLogout();
const handleLogout = () => {
    logout();
};
```

Hook : Logout / déconnexion

```
import { useAuthContext } from "./useAuthContext";

export const useLogout = () => {
    const { dispatch } = useAuthContext();

    const logout = () => {
        // remove user from storage
        localStorage.removeItem("user");
        // dispatch logout action
        dispatch({ type: "LOGOUT" });
    };

    return { logout };
};
```

Local storage

Context user

Console : localstorage vidé

```
User : ▼ {user: null} i
         user: null
       ▶ [[Prototype]]: Object
```

Retour au login

2. Code : Posts

Posts - Backend : Routeur + Sécurité

Router : Posts + Upload (images)

```
app.use("/api/posts", post_router);
app.use("/api/upload", upload_router);
```

```
import express from "express";
import {
    createPost,
    updatePost,
    deletePost,
    likePost,
    getPost,
    getPosts,
    getUserPosts,
} from "../controllers/post_controller.js";
import auth from "../middleware/auth_mw.js";

const router = express.Router();

//? Routes
router.use(auth);
router.post("/", createPost);
router.put("/:id", updatePost);
router.delete("/:id", deletePost);
router.put("/:id/like", likePost);
router.get("/:id", getPost);
router.get("/", getPosts);
router.get("/from/:id", getUserPosts);

export default router;
```

Fonctions contrôleur

middleware authentification

Middleware : Authentification - Json Web Token

```
import jwt from "jsonwebtoken";

const requireAuth = async (req, res, next) => {
    const { authorization } = req.headers;

    if (!authorization) {
        return res.status(401).json({ error: "Authorization token required" });
    }

    try {
        const token = authorization.split(" ")[1];
        const decodedToken = jwt.verify(token, process.env.JWT_KEY);
        const userId = decodedToken.userId;
        req.auth = {
            userId: userId,
        };
        next();
    } catch (error) {
        res.status(401).json({ error: "Request is not authorized" });
    }
};

export default requireAuth;
```

Authorization header ?

Echec requête : Token absent

Secret

Récupérer id de l'utilisateur à partir du Token

Echec requête : L'utilisateur n'est pas autorisé

Posts - Backend : Model, Créer

Model : Post

```
import mongoose from "mongoose";                    Mongoose

const PostSchema = new mongoose.Schema(
  {
    userId: {
      type: String,                                 ID de l'utilisateur
      required: true,
    },
    desc: {
      type: String,                                 Message
    },
    image: {
      type: String,                                 Image
    },
    likes: {
      type: [],                                     Tableau pour Likes
    },
  },
  { timestamps: true }                              Date création
);

const PostModel = mongoose.model("Post", PostSchema);
export default PostModel;
```

Controller : Créer un message

```
//? Create
export const createPost = async (req, res) => {
  const reqPost = req.body;                         Récupérer la requête
  const newPost = new Post({
    ...reqPost,                                     Appliquer le modèle Mongoose
  });
  try {
    await newPost.save(); Sauvegarder en database
    res.status(200).json(newPost);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};
```

Database : Post

```
_id: ObjectId('63487d4812a70e0c41e53d94')
userId: "63487bfe12a70e0c41e53d57"
desc: "1. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendiss…"
image: "16657808117717.jpg"
likes: Array
  0: "63487dd012a70e0c41e53da2"
  1: "63487bfe12a70e0c41e53d57"
  2: "63487f0312a70e0c41e53db4"
createdAt: 2022-10-13T21:04:08.214+00:00
updatedAt: 2022-10-15T00:02:21.910+00:00
__v: 0
```

Posts - Backend : Route upload + Multer

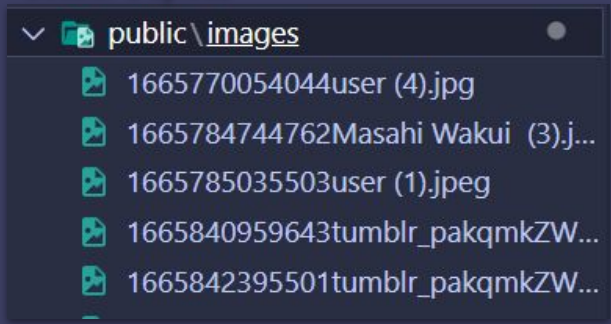
Appel routes dans index.js

```
app.use("/api/posts", post_router);
app.use("/api/upload", upload_router);
```

Accès dossier local

```
app.use(express.static("public"));
app.use("/images", express.static("images"));
```

Dossier d'upload



Router / Controller : Multer

```
import express from "express";
import multer from "multer";
import auth from "../middleware/auth_mw.js";

const router = express.Router();

const storage = multer.diskStorage({
    destination: (req, file, cb) => {
        cb(null, "public/images");
    },
    filename: (req, file, cb) => {
        cb(null, req.body.name);
    },
});
const upload = multer({ storage });

router.post("/", auth, upload.single("file"), (req, res) => {
    try {
        return res.status(200).json("File uploaded successfully");
    } catch (error) {
        console.log({ message: error.message });
    }
});

export default router
```

Configuration de Multer =

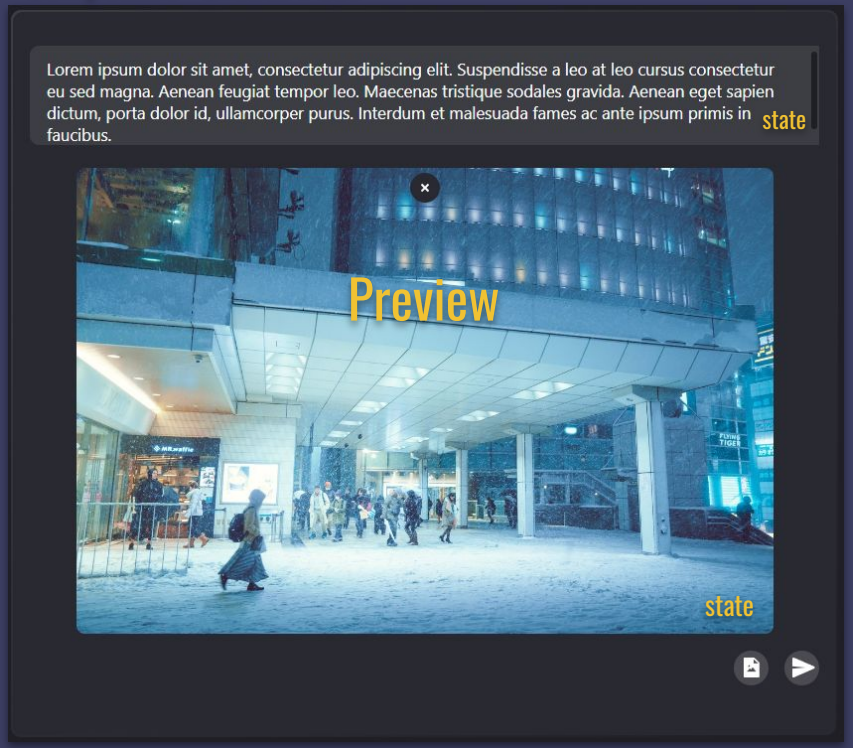
Destination du fichier +

Création du nom du fichier

Route d'upload

Posts - Frontend : Créer (1)

Composant : Create Post




```jsx
return (
  <article className="createpost gradient-border">
    <form onSubmit={handleSubmit}>
      <textarea
        type="text"
        placeholder="Ecrire un message ... "
        value={desc}
        onChange={(e) => setDesc(e.target.value)}
      />
      {file && (
        <div className="uploaded-image">
          <img src={URL.createObjectURL(file)} />
          <div className="close-icon" onClick={() => setFile(null)}>
            {<img src={closeIcon} alt="remove" />}
          </div>
        </div>
      )}

      <div className="btns">
        <label htmlFor={"image"} aria-label="select file">
          <div>
            <img
              src={fileIcon}
              alt="select file"
              onClick={() => imageRef.current.click()}
            />
          </div>
        </label>

        <input
          type="file"
          accept="image/png, image/jpeg, image/jpg, image/webp"
          ref={imageRef}
          onChange={onImageChange}
        />
        <button type="submit" aria-label="submit">
          <img src={sendIcon} alt="send" />
        </button>
      </div>
    </form>
  </article>
);
```

state message

Preview image

Choisir un fichier

Envoyer image dans state

Posts - Frontend : Créer (2)

Envoi du Post

```
//? Submit post
const handleSubmit = async (e) => {
   e.preventDefault();
   if (desc === "" && file === null) {
      return;
   }
   const newPost = {
      userId: auth.user._id,
      desc: desc,
   };

   if (file) {
      const data = new FormData();
      const fileName = Date.now() + file.name;
      data.append("name", fileName);
      data.append("file", file);
      newPost.image = fileName;
      try {
         await axios.post("/api/upload", data, {
            headers: {
               Authorization: `Bearer ${auth.token}`,
            },
         });
      } catch (error) {
         console.log({ message: error.message });
      }
   }
   try {
      const res = await axios.post("/api/posts", newPost, {
         headers: {
            Authorization: `Bearer ${auth.token}`,
         },
      });
      setDesc("");
      setFile(null);
      dispatch({ type: "CREATE_POST", payload: res.data });
   } catch (error) {
      console.log({ message: error.message });
   }
};
```

Annuler si : Post vide

Récupérer id utilisateur

Si image ...

Requête route upload

Requête api

Vider les states

Context : dispatch

Context Post

```
import { createContext, useReducer } from "react";
export const PostsContext = createContext();

export const postsReducer = (state, action) => {
   switch (action.type) {
      case "SET_POSTS":
         return {
            posts: action.payload,
         };
      case "CREATE_POST":
         return {
            posts: [action.payload, ...state.posts],
         };
      case "DELETE_POST":
         return {
            posts: state.posts.filter(
               (post) => post._id !== action.payload._id
            ),
         };
      case "UPDATE_POST":
         return {
            posts: state.posts.map((post) => {
               if (post._id !== action.payload._id) {
                  return post;
               }
               return action.payload;
            }),
         };
      default:
         return state;
   }
};

export const PostsContextProvider = ({ children }) => {
   const [state, dispatch] = useReducer(postsReducer, { posts: null });

   return (
      <PostsContext.Provider value={{ ...state, dispatch }}>
         {children}
      </PostsContext.Provider>
   );
};
```

Afficher

Créer

Supprimer

Modifier

Reducer

Appliquer dans le root de src/index.js

Posts - Backend : Afficher

Controller : Afficher tous les messages antéchronologiquement

```
//? Get all
export const getPosts = async (req, res) => {
   try {
      const posts = await Post.find();
      res.status(200).json(
         posts
            .filter((post) => post !== null)
            .sort((a, b) => {
               return new Date(b.createdAt) - new Date(a.createdAt);
            })
      );
   } catch (error) {
      res.status(500).json({ message: error.message });
   }
};
```

Méthode .find

Classer les messages ...
En fonction de leur date de création

Database : Posts

```
_id: ObjectId('63487d4812a70e0c41e53d94')
userId: "63487bfe12a70e0c41e53d57"
desc: "1. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendiss…"
image: "16657808117717.jpg"
› likes: Array
createdAt: 2022-10-13T21:04:08.214+00:00
updatedAt: 2022-10-15T00:02:21.910+00:00
__v: 0
```

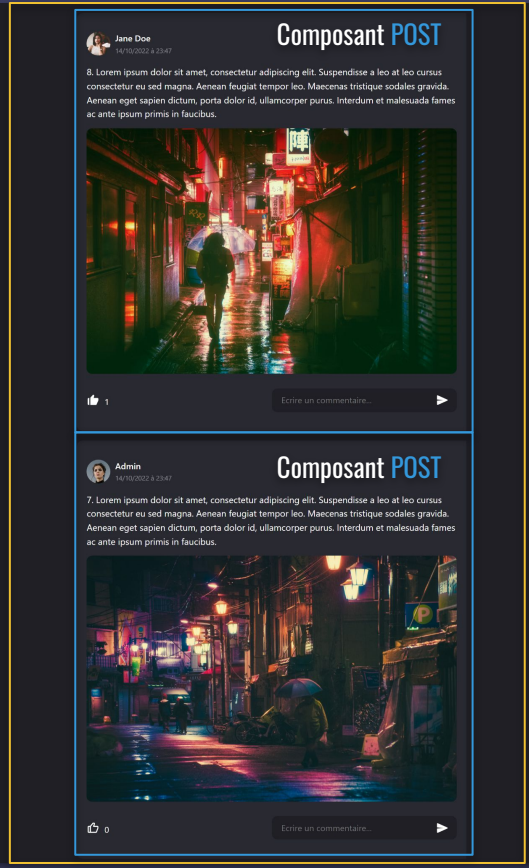
Date de création

```
_id: ObjectId('63487e4c12a70e0c41e53db0')
userId: "63487dd012a70e0c41e53da2"
desc: "2. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendiss…"
image: "16657807817712.jpg"
› likes: Array
createdAt: 2022-10-13T21:08:28.335+00:00
updatedAt: 2022-10-15T00:02:20.490+00:00
__v: 0
```

```
_id: ObjectId('63487ff312a70e0c41e53de0')
userId: "63487f0312a70e0c41e53db4"
desc: "3. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendiss…"
image: "16657808219851.jpg"
› likes: Array
createdAt: 2022-10-13T21:15:31.445+00:00
updatedAt: 2022-10-14T21:18:30.602+00:00
__v: 0
```

```
_id: ObjectId('6349d88a207401d6193dc533')
userId: "63487bfe12a70e0c41e53d57"
desc: "4. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendiss…"
image: "16657839464273.jpg"
› likes: Array
createdAt: 2022-10-14T21:45:46.453+00:00
updatedAt: 2022-10-14T21:46:05.046+00:00
__v: 0
```

Posts - Frontend : Afficher



Composant POST(S)

Composants : POSTS - Affichage de tous les composants POST

```jsx
import { useEffect } from "react";
import axios from "axios";
import { usePostsContext } from "../../hooks/usePostsContext";
import { useAuthContext } from "../../hooks/useAuthContext";
import Post from "./Post";

const Posts = () => {
  const { posts, dispatch } = usePostsContext();
  const { user: auth } = useAuthContext();

  useEffect(() => {
    const getPosts = async () => {
      const res = await axios.get("/api/posts", {
        headers: {
          Authorization: `Bearer ${auth.token}`,
        },
      });
      dispatch({ type: "SET_POSTS", payload: res.data });
    };
    getPosts();
  }, [auth.token, dispatch]);

  return (
    <div className="posts">
      {posts && posts.map((post) => <Post post={post} key={post._id} />)}
    </div>
  );
};

export default Posts;
```

Context Post : affichage dynamique

Context User : autorisation

Requête GET api

Token autorisation

Context : dispatch

Méthode map -> applique le composant “POST” à chaque objet post

Posts - Backend : Modifier + Supprimer

Controller : Modifier un message

```
//? Update
export const updatePost = async (req, res) => {
   const postId = req.params.id;                          Récupérer ID du Post +
   const { userId, admin } = req.body;                    Données de l'utilisateur

   try {
      const post = await Post.findById(postId);           Correspondance ID database
      if (admin || post.userId === userId) {              Autorisation
         if ((req.body.image && post.image) || post.image) {
            const filename = post.image.split("public/images/")[0];
            fs.unlink(`public/images/${filename}`, () => {});
         }                                                Supprimer ancienne image hébergée

         await post.updateOne({ $set: req.body });
         res.status(200).json(req.body);                  Modifier avec nouvelle données
      } else {
         res.status(403).json("You can only update your own posts !");
      }
   } catch (error) {
      res.status(500).json({ message: error.message });
   }
};
```

Controller : Supprimer un message

```
//? Delete
export const deletePost = async (req, res) => {
   const postId = req.params.id;                          Récupérer ID du Post +
   const { userId, admin } = req.body;                    Données de l'utilisateur
   try {
      const post = await Post.findById(postId);           Correspondance ID database
      if (admin || post.userId === userId) {
         if (post.image) {                                Supprimer ancienne image hébergée
            const filename = post.image.split("public/images/")[0];
            fs.unlink(`public/images/${filename}`, () => {
               post.deleteOne();
               res.status(200).json(post);
            });
         } else {
            post.deleteOne();                             Supprimer de la database
            res.status(200).json(post);
         }
      } else {
         res.status(403).json("You can only delete your own posts !");
      }
   } catch (error) {
      res.status(500).json({ message: error.message });
   }
};
```

Posts - Frontend : Modifier + Supprimer

Condition : Affichage modal Modifier / Supprimer

```
{auth.user.admin || (auth.user._id === post.userId) ? (
    <PostOptionsModal options={options} />
) : null}
```

Composant : Post - Requêtes

SUPPRIMER

```
//? Delete Post
const handleDelete = async () => {
    const req = {
        userId: auth.user._id,
        admin: auth.user.admin,
    };
    if (auth.user.admin || auth.user._id === post.userId) {
        try {
            const res = await axios.delete(
                `/api/posts/${post._id}`,
                {
                    data: req,
                    headers: {
                        Authorization: `Bearer ${auth.token}`,
                    },
                }
            );
            dispatch({ type: "DELETE_POST", payload: res.data });
        } catch (error) {
            console.log({ message: error.message });
        }
    }
};
```

Autorisation

Requête DELETE api

Token

Context : dispatch

MODIFIER

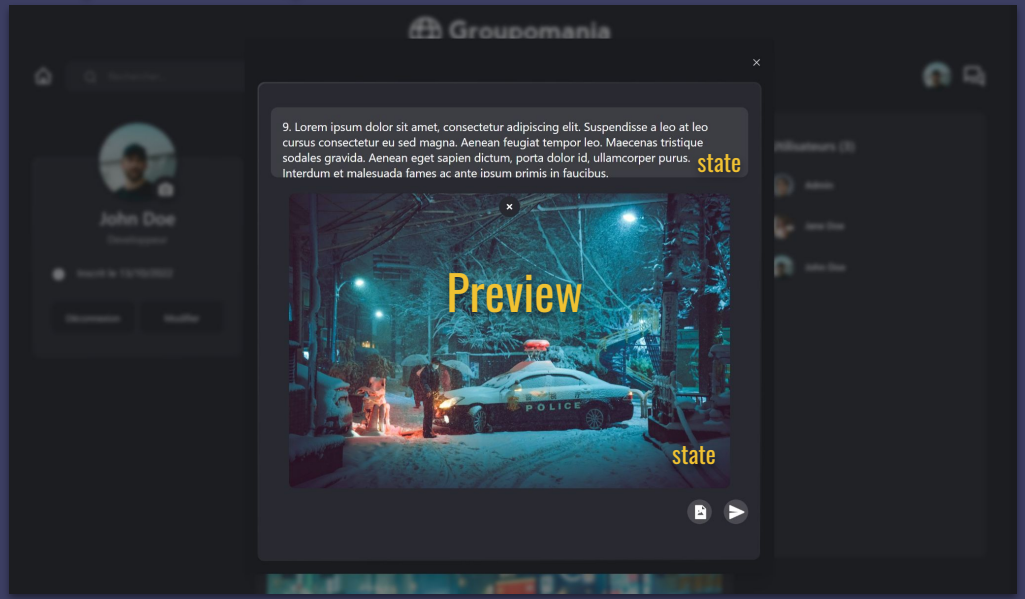
```
//? Update Post
const handleUpdate = () => {
    if (auth.user.admin || auth.user._id === post.userId) {
        setUpdatePostModal(true);
    }
};
```

Autorisation

Afficher Modal de modification

Posts - Frontend : Modifier (modal)

Composant : Modal Update Post




```
//? Submit updated post
const handleSubmit = async (e) => {
   e.preventDefault();
   if (updatePost.desc === "" && file === null) {
      return;
   }
   if (auth.user.admin || auth.user._id === data.userId) {
      try {
         if (file) {
            const data = new FormData();
            const fileName = Date.now() + file.name;
            data.append("name", fileName);
            data.append("file", file);
            updatePost.image = fileName;
            try {
               await axios.post("/api/upload", data, {
                  headers: {
                     Authorization: `Bearer ${auth.token}`,
                  },
               });
            } catch (error) {
               console.log({ message: error.message });
            }
         } else {
            updatePost.image = null;
         }
         const res = await axios.put(`/api/posts/${data._id}`, updatePost, {
            headers: {
               Authorization: `Bearer ${auth.token}`,
            },
         });
         dispatch({ type: "UPDATE_POST", payload: res.data });
         setUpdatePostModal(false);
         setFile(null);
      } catch (error) {
         console.log({ message: error.message });
      }
   }
};
```

Si message vide ...

Si image ...

Requête route upload

Token

Requête PUT api

Context : dispatch

Post - Likes

1 - Backend : Post controller - Likes

```
//? Like / Dislike
export const likePost = async (req, res) => {
   const postId = req.params.id;
   const { userId } = req.body;
   try {
      const post = await Post.findById(postId);
      if (!post.likes.includes(userId)) {
         await post.updateOne({ $push: { likes: userId } });
         res.status(200).json("Post liked !");
      } else {
         await post.updateOne({ $pull: { likes: userId } });
         res.status(200).json("Post disliked !");
      }
   } catch (error) {
      res.status(500).json({ message: error.message });
   }
};
```

Quel post ? Quel utilisateur ?

Correspondance database

Likes n'inclue pas user

Likes inclue déjà user

2 - Frontend : Requête

```
//? Like system
const [liked, setLiked] = useState(false);
const [like, setLike] = useState(post.likes.length);

useEffect(() => {
   setLiked(post.likes.includes(auth.user._id));
}, [auth.user._id, post.likes]);

const handleLikes = () => {
   axios.put(
      "/api/posts/" + post._id + "/like",
      { userId: auth.user._id },
      {
         headers: {
            Authorization: `Bearer ${auth.token}`,
         },
      }
   );
   setLike(liked ? like - 1 : like + 1);
   setLiked(!liked);
};
```

Post est-il liké ?

STATES

Nombre de likes

POST déjà liké ?

Requête PUT api

Ajouter userId au tableau LIKES

Si déjà liké ? = -1 / Sinon +1

Modifier le state de false à true, etc.

3 - Frontend : Bouton / Affichage de l'icône

```
<div className="post__reactions">
   <img
      src={liked ? likeIcon : likeOutlined}
      alt="like"
      onClick={handleLikes}
      title="Aimer ce message"
   />
   <p>{like}</p>
</div>
```

state

1

0

Bilan

Objectifs atteints

- Utilisation d'un framework JavaScript
- Logo + Police respectés, couleurs libres
- Design totalement responsive
- Affichage postes ordre antéchronologique
- Connexion persistante, Déconnexion
- Données du site stockées en database (users/posts)
- Données utilisateurs sécurisées en database (mail/pass)
- Créer / Modifier / Supprimer les postes respectifs
- Un poste peut contenir texte/image
- Système de Likes pour les postes
- Administrateur peut supprimer/modifier tous les postes
- Possibilité de cloner et lancer l'application (test réalisé)
- Respect des standards d'accessibilité WCAG (test Wave)

Axes d'amélioration

- Page de profil pour chaque utilisateur
- Page pour chaque poste (meilleure visibilité)
- Système de commentaires
- Barre de recherche fonctionnelle
- Afficher les utilisateurs connectées (volet droite)
- Implémenter un messenger avec socket.IO
- Améliorer l'édition de messages
- Renforcer la sécurité niveau backend (OWASP)
- Optimiser le code pour meilleures performances
- Choix de thème light/dark
- Guider l'utilisateur dans le formulaire d'inscription
- Contrôler le poids/dimensions max d'une image ajoutée
-
-