MERN Stack Project : Build a Modern Real Estate Marketplace with react MERN (jwt, redux, toolkit)

Nom de la chaine Youtube : React & Next js Projects with Sahand

On va créer une application robuste avec une authentification et des fonctionnalités de recherche.

On va installer React, TailwindCss, React Router DOM

1. Créer le dossier mern-estate
2. L’ouvrir dans VSCode

Installation de React, Tailwind CC

1. Installer vite dans le folder client :

npm create vite@latest client

client est le nom du dossier

latest pour la dernière version

Taper y pour yes

Il va me demander quel framework je veux installer. Choisir React, Puis choisir le type de Script : JavaScript + SWC (qui est la version la plus rapide)

Puis le dossier client sera créé.

1. Puis aller à l’intérieur du folder :

cd client

npm install ou npm i . Si cela ne marche pas, ça veut dire que je n’ai pas node d’installé.

1. Installer Tailwind CSS

Rechercher sur Google, « Install Tailwind CSS with Vite ».

Sur le site, choisir : Using React

Il ya 6 étapes. La prochaine est la deuxième :

npm install -D tailwindcss postcss autoprefixer

npx tailwindcss init -p (va initialiser les fichiers : tailwind.config.js et postcss.config.js )

Etape 3 : Configurer le fichier tailwind.config.js

Effacer tout ce qui y est et coller ceci : /\*\* @type {import('tailwindcss').Config} \*/

export default {

content: [

"./index.html",

"./src/\*\*/\*.{js,ts,jsx,tsx}",

],

theme: {

extend: {},

},

plugins: [],

}

Etape 4 : Remplacer le contenu du fichier «./src/index.css » par :

@tailwind base;

@tailwind components;

@tailwind utilities;

1. Supprimer les fichiers qui ne sont pas nécessaires :

App.css

vite.svg dans le dossier public

react.svg dans le dossier ./src/assets

1. Dans le fichier App.jsx, tout effacer et remplacer par un React Functional Component (rfc)
2. Installer l’extension : ‘Auto Rename Tag’, qui va effectuer des changements dans la balise de fermeture en fonction de ce qui est effectué dans la balise d’ouverture.
3. Installer l’extension ‘Console Ninja’, pour console logger à l’intérieur de mon code
4. Installer GitHub Copilot qui te fait des suggestions pour coder rapidement moyennant 10$/mois. GitHub Copilot Chat est un équivalent de Chat GPT
5. Installer Tailwind CSS IntelliSense qui va me suggérer des classes de Tailwind CSS, par exemple : la palette de couleur
6. Dans le terminal, taper :

npm run dev

Création du Dépôt GitHup

1. Initialiser un repo github :

git init

1. Stagger/indexer tous les fichiers avec :

git add . ou git add all

1. Commit les fichiers avec :

git commit -m ‘’install react js and tailwind css and create the first template’’

1. Sur le site de GitHub :

-créer un nouveau repository

-nom : mern-estate

-description : Real Estate app using MERN and tailwind css

-Comme on a fait la partie d’initialisation, on va directement ‘push an existing repository from the command line’

git remote add origin https://github.com/cindybibou/mern-estate.git

git branch -M main

git push -u origin main

Création des pages et routes de notre application :

1. Dans le dossier src, créer le dossier : ./src/pages
2. A l’intérieur, créer les pages: About, SignIn, SignOut, Profile, Home est fichier .jsx

rfc pour créer les composants correspondants

1. Installer React Router Dom :

npm i react-router-dom

1. Dans App.jsx :

-Effacer ‘import React as ‘react’ ‘

-écrire : import { BrowserRouter as Router, Routes, Route } from ‘react-router-dom’ ;

-Rajouter toutes les routes

NB : Pour copier-coller une ligne : Alt + Shift +flèche du bas ou du haut

1. Stager, commit et push tous les fichiers dans la rubrique « changes » de l’extension GitLens Inspect.

Création du Header :

1. Le Header est composé du logo, de la barre de recherche, des menus Home, About, SignIn. Le site étant responsive, en version mobile, les menus Home et About disparaissent, il ne reste plus que SignIn. De plus en taille mobile, les textes sont de taille différente.
2. Dans ./src, créer une un autre dossier nommé « components ».Y créer un fichier « Header.jsx » .
3. Pour rajouter ce Header à toutes les pages, aller dans App.jsx, en dehors de toutes les routes, et en-dessous de « Router », rajouter le composant Header.
4. Modifier le Header :

-Remplacer les balises <div> par des balises <header>

-On doit avoir le logo à gauche, la barre de recherche au milieu et les menus sur la droite

-Le logo est un texte

-Pour rajouter la search bar, on utilise les balises <form>.Et à l’intérieur, un <input> de type text et un placeholder

NB : pour commit les changements, je vais dans l’extension gitLens pour index les new changements, puis je clique sur « Explorateur », puis « contrôle de code source » pour rentrer un message valider et envoyer.

Créer et démarrer le serveur

1. Dans la section « Explorateur », au même niveau que client, créer un dossier séparé nommé « api ».
2. Initialiser un package.json à la racine de mon projet et non dans le dossier « api ». Car, quand on déploiera notre site, la plateforme de rendu va vérifier la racine de notre site web pour prendre en compte le front et le back-end.

-faire : cd.. et m’assurer que je suis bien dans le fichier « mern-estate ».

-npm init -y pour initialiser le package.json

1. Dans ce fichier « package.json », il est indiqué « main : index.js », je dois alors créer un fichier « index.js » à l’intérieur du dossier « api ».
2. Toujours à la racine « mern-estate », installer express :

npm i express

1. Dans index.js, importer express .

-Créer une application express.

Const app= express()

Elle nous donnera plusieurs méthodes à utiliser. Dont écouter un port :

-app.listen(3000, () => {

console.log(‘Server is running on port 3000’) ;

}

) ;

1. Pour démarrer notre serveur :

node api/index.js

1. Pour résoudre l’erreur :

Dans le fichier package.json à la racine, en-dessous de « ‘’main’’ : ‘’index.js’’ », rajouter :

-‘’type’’ : ‘’module’’

Si je ne veux pas rajouter ce type, je dois utiliser la syntaxe common js ‘require’ dans mon fichier ‘api/index.js’.

1. Installer nodemon :

npm i nodemon

1. Dans package.json, effacer el script présent et rajouter celui-ci :

‘’dev’’ : ‘’nodemon api/index.js’’,

‘’start’’ : ‘’node api/index.js’’ // très important, car le déploiement a besoin d’un rendu qui utilise node au lieu de nodemon.

Maintenant, pour démarrer « nodemon api/index.js », je dois juste taper dans le terminal :

npm run dev

1. Au départ, mon repo github a été crée dans le dossier client, pour le déplacer à la racine mern-estate :

cd client

mv .git ../

Connect to database

1. On va utiliser le package mongoose pour échanger facilement avec la base de données MongoDB .
2. Par défaut, on ne peut pas utiliser de variable d’environnement dans le back-end. Pour être capable de faire ça, on a besoin d’installer le package « npm i dotenv » et l’importer.Et l’initialiser en utilisant : dotenv.config()
3. Pour détecter si on est connecté ou pas,

Création de notre premier modèle de données (pour les utilisateurs) : the user model.

1. La personne qui veut interagir avec ma database, doit respecter des règles et exigences. Je vais donc créer des variables et données basées sur ces règles.
2. Importer mongoose et créer les règles à travers un schéma.

On utilise la méthode .Schema pour spécifier ces règles.

1. Demander à mongo DB d’enregistrer l’heure de la création du user ou l’heure à la quelle le user à été mis à jour. Ce qui nous permettra de trier ces informations en fonction de l’heure.

La commande est timeStamps : true

1. Créer le modèle User à l’aide de la méthode model de mongoose, qui prend comme paramètre : le nom du modèle : ici c’est ‘User’, et le deuxième paramètre est le nom du schéma : userSchema
2. Exporter ce modèle : User pour être capable de l’utiliser n’importe où ailleurs dans mon application.

Création d’une route API de test

1. Dans le fichier index.js, créer une route API get.

On créer un proxy pour éviter le spécifier le port à chaque fois dans l’url.

1. Créer un dossier séparé pour les routes API.

Dans le dossier API, créer un dossier nommé « routes » et un fichier nommé : user.route.js. Il contiendra toutes les routes de notre API.

-importer express

-Créer un router

-créer notre route API de test

-exporter

1. Toutes les routes doivent être définies à l’intérieur de index.js.

-A l’intérieur, importer le router. Et changer son nom pour « userRouter »

1. Dans le dossier api, créer un dossier controllers et y mettre toute la logique et les fonctions. Y créer le fichier user.controller.js

Création de la route API pour le sign up. Temps : 1 :22 :37

Pour lancer les serveurs :

npm run dev dans le dossier mern-estate (terminal du milieu et de droite)

npm run dev dans le dossier client (dans le terminal de gauche)

1. Nous voulons avoir les informations : username, email, mot de passe de notre client. Puis, on va hash le mot de passe et l’enregistrer à l’intérieur de la base de données

-On va créer la route sign up de l’API.

-Dans le dossier routes, on va créer le fichier : auth.route.js

-dans controller, on va créer un fichier : auth.controller.js

1. Comme je n’ai pas encore de formulaire, je vais utiliser Insomnia pour tester les routes de l’API.

On aura différentes routes : pour le listing, le user, l’authentification. Donc, on va les séparer dans Insomnia en utilisant un dossier.

1. Il faut hash (encrypter) le mot de passe.

On va utiliser le package bcrypt.js

Créer le middleware et une fonction pour gérer les différentes erreurs possibles

1. Pour chaque route, on ne va pas créer un try catch. Cà sera trop répétitif. Pour cela, on va créer un middleware à l’intérieur du fichier « index.js »
2. Dans api, créer un dossier utils et un fichier error.js qui va contenir la function pour l’erreur créer par le développeur par exemple pour le mot de passe pas assez long.

Création de l’interface utilisateur de la page sign up

Création de la fonctionnalité de la page sign up complète

Ajout de la fonctionnalité et connexion à la base de données.