Outils :

Visual Studio

Node js 19.7.0

<https://git-scm.com/download/win> git pour github apres

postman : pour tester les requetes backend

Extensions sur Visual Studio code :

ES7 React/Redux...

Bracket Pair colorizer

Auto rename tag

JavaScript (ES6) code snippets

Prettier – code formatter

Dans VS code, settings / rechercher : format on save, puis la cocher

React :

Générer une app depuis le terminal de VS :

npx create-react-app frontend

cd frontend

npm start

Commencer à coder :

D’abord index.html (dans public) contient root, appelé depuis index.js (dans src) qui appelle le app.js (dans src)

Modifier index.html et app.js et index.css

Supprimer des fichiers indésrables

Supprimer le dossier .git dans frontend parce qu’on aura besoin dans le dossier parent et couper .gitignore dans le dossier parent et le modifier

Aller dans le dossier parent dans le terminal et :

git init git add . git status git commit –m ’React setup’

Configuration globale au debut pour que ca marche avec git :

$ git config --global user.name "Tarek Bchini"

$ git config --global user.email tarek.bchini83@gmail.com

cd frontend

npm start

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

commencer à créer les components

rafce permet de créer une structure prete

et importer dans app.js

on cree apres les components : header et footer

maintenant travailler avec bootstrap

react-bootstrap.github.io

bootswatch.com (télécharger un thème bootstrap.min.css qu’on mettra dans src) et l’imoporter depuis index.js

npm install react-bootstrap (dans frontend, et on peut ajouter bootstrap), puis l’importer depuis App.js

travailler sur le footer et le header

par exemple pou le navbar : <https://react-bootstrap.netlify.app/docs/components/navbar/>

aller a cdnjs.com/libraries/font-awesome et copier le premier lien tag dans index.html (pour travailler avec les icones)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

apres travailler sur header et footer.

Après, products.js dans src (contient un tableau de produits) pour l’affichage en front : HomeScreen.js appelé dans App, et Product.js appelé dans HomeScreen.js (ici le props est product passe de HomeScreen à Product.js)

et sur index.css

apres pour les routes npm install react-router-dom et react-router-bootstrap dans frontend (pour lien avec la partie react-bootstrap)

et on travaille les routes surtt vers la racine / (dabord dans app.js, product.js et header.js)

Apres passer au rating pour les etoiles (fichier Rating.js, product.js)

Apres faire l’affichage des produits en detail , on cree le fichier ProductScreen.js et on modifie app.js

Pour le backend, on lance dabord depuis le parent npm init (2eme terminal) (mettre server.js comme point d entree)

Il cree package.json , et faire quelques modif sur le fichier

Puis créer le dossier backend dans la racine et créer dedans server.js, puis dans le terminal (parent) faire npm start ca doit afficher hello worl (dans server.js)

puis dans la racine npm install express

apres modifier server.js pour tester le serveur express , apres sur terminal (parent) faire npm start pour tester, puis tester le lien localhost:5000

créer dossier data dans backend et copier le fichier product.js dans data, et modifier server.js

pour teser : refaire npm start depuis la racine, puis tester le lien localhost:5000/api/products et on voit les produits

puis modifier server.js pour afficher un seul produit, npm start et mettre le lien localhost:5000/api/products/1

Maintenant pour que les deux serveurs back et front tournent au meme temps, il faut faire une configuration : dabord on stoppe le serveur backend et front end

installer nodemon et conurrently dans la racine npm i –D nodemon concurrently (ils s’ajoutent à package.json)

et modifier package.json pour que le serveur démarre automatiquement sur nodemon et démarrer le frontend

apres : npm run server dans un terminal (racine pour backend) et nodemon demarre avec le serveur en continue.

Je modifier encore le fichier package.json (pour démarrer le serveur du client), et j’ouvre un autre terminal (du front) et npm run client (depuis la racine) et ca demarre le client.

apres pour que les deux démarrent ensemble, on utilise concurrently (modifiation dans package.json) npm run dev demarre les deux serveurs (mais on arrete d’abord les deux serveurs).

maintenant regler variable d’environnement : dotenv

d’abord arreter le serveur (les 2 vont s’arreter) npm i -D dotenv (il aparaitera dans package.json)

creer le fichier .env dans la racine et le remplir, puis modifier server.js, puis dans le terminal npm run server ca demarre c bon. On ferme le serveur et npm run dev

Maintenant pour utiliser axios pour les requetes (entre client et serveur) : fermer les serveurs, dna sl terminal se positionner sur le frontend et : npm install axios (on va le trouver ajouté dans package.sjon de frontend), et on modifie package.json (de frontend)

Apres on redémarre les 2 serveurs depuis la root avec npm run dev

Aller modifier HomeScreen.js (dans frontend) pour qu’il fonctionne avec axios pour récupérer les données depuis le backend (avec useEffect et useState) (dans inspecter, network on pour vérifier qu’ils parviennent depuis le backend). Après modifier ProductScreen

commencer a travailler avec mongodb (BD)

créer un compte dans mongodb.com se connecter avec mon compte google

tarekbchini83 QTNBuQc6dAUkxGkk

nouveau projet (pour créer un nouveau projet) , puis build a database, choisir free et Aws, et créer

ca genere un login et utilisateur (on peut les changer) et créer sur create user, apres finish and close ca va diriger vers la BD, Network access : permet d’acceder de n’importe ou

(<https://adresseip.com/> pour voir l’adresse ip de son routeur)

ici j crée BD test avec : tarekbchini83, D6AObD6CxZoZXcuc

je clique sur connect, drivers et je copie le lien en bas

aller dans le fichier .env dans l’appli et le copier, et changer le password, et avant la ? mettre le nom de la BD (ici elle sera entreprises)

Puis retourner au lien de mongodb et cliquer sur browse collections et cliquer sur add my own data, ajouter le nom de la BD, ici entreprises, collection names : products, et créer, et on trouve la table products dans la BD entreprises

Apres s’assurer que dans le fichier .gitignore il y a .env (pour pouvoir le pusher apres dans git)

telecharger et installer mongodb compass en local

dans la page web de mongodb : Overview, connect, compass, copier le lien en bas et le mettre dans le compass en local pour la connexion (ajouter le mot de passe) et connecter

maintenant on peut fermer le lien web de mongodb

Maintenant installer : npm install mongoose dans la root

Créer un fichier de configuration db.js dans backend/config pour configurer la BD et modifier server.js pour la connection DB.

Pour tester, depuis la racine : npm run server (donne mongodb connected)

Dans le backend, créer le dossier models et créer dedeans productModel.js, qui va contenir les produits ainsi que l’utilisateur qui a acces, apres créer userModel.js qui va contenir les utilisateurs, puis c reer aussi orderModel.js (ici comme si nous avons créé les données de notre base de données)

Maintenant on va préparer quelques données pour tester : donc on va créer un fichier users.js dans le dossier data, et dans le fichier products.js dans data nous allons supprimer les \_id (mais avant, nous devons faire npm i bcryptjs pour crypter les mots de passe)

Maintenant on va inserer les données dans la BD (mais avant, on va faire npm i colors pour génératon des couleurs dans les resultata d’execution dans le terminal). Après : dans le dossier backend, créer le fichier seeder.js (qui est un script d’insertion et de destruction des données) , et ajouter deux lignes dans package.json (apres tester dans le terminal avec npm run data:import ca doit etre bon, aller a compass pour vérifier (il faut qu’on soit connecté à la BD avec compass et la page web de mongo pour importer les données, apres il faut ouvrir une autre page de compass et on trouve les données dans la BD et aussi dans la page web de mongo)) : et après pour supprimer les donnes de la BD faire dans le terminal : npm run data:destroy (on peut vérifier les données dans la BD). Maintenant refaire import pour remplir les données.

Maintenant onva travailler avec Postman pour les requetes http avc le backend. Télécharger postman et l ouvrir. Workspace et créer un nouveau workspace. L’appeler entreprises et metre personnel et créer. Après Environments, et créer environnement, l’appeler Entreprises environment ; apres ajouter une variable basURL ; initial value : <http://localhost:5000/api> dans current value aussi et enregistrer ; et elle apparit dans la liste des environnements qu’on peut utiliser. Après, dans postman aller à APIs, créer une api, la nommer entreprises API, donner une version et la créer. Maintenant on va commencer à manipuler les collections. Aller à collection haut (dans postman), new collection, la premiere la nommer products ; puis quand la collection products est créé, clic que les 3 points et new request ; on l’appelle GET All Products ; mettre {{baseURL}}/products (la choisir de la liste); puis cliquer sur send (il faut que le serveur soit en marche) et il va aficher les informations sur tous les produits.

Maintenant on va chercher les données depuis la BD, créer un dossier routes dans le backend, et le fchier productRoutes.js, et copier dedans deux requetes de produits depuis server.js (et ajouter deux lignes dans server.js). Tester localhost/3000 ca marche. Après, dans backend, créer un dossier middleware, et dans le dossier middleware, créer un fichier asyncHandler.js. Et en testant, a ce stade ca marche, meme en cliquant sur un produit, ses informations s’affichent sur localhost/3000.

Maintenant ouvrir postman, nouvelle requete, l’appeler Get Single Product, donner {{baseURL}}/products/ :id ; apres dans Path variable copier depuis l’URL d’un produit son ID, le mettre dans value dans Postman et faire send et ca sort les informations sur le produit. Puis l’enregistrer.

Maintenant on vaa faire un fichier pour les erreurs : dans le dossier middleware, créer un fichier : errorMiddleware.js et modifier server.js, et modifier productRoutes.js (maintenant quand on teste un ID inexistant dans postman il génére l erreur).

Maintenant on va créer le controleur de produit : dans backend, on cree le dossier controllers, ensuite créer le fichier productController.js, et aller modifier productRoutes.js. En testant la page web tout fonctionne avec la BD, bonne organisation MVC et postman marche.

Passons maintenant à Redux : librairie JS pour gerer les applications complexes.

Aller dans frontend dans le terminal et : npm i @reduxjs/toolkit react-redux (il s’ajoute dans package.json de frontend), dans le dossier frontend/src créer un fichier store.js. Dans visual studio ajouter l’extension : code spell checker. Aller apres sur index.js. Après ajouter Redux devtools à chrome (navigateur) et inspecter puis redux (notre page web store) juste pour voir.

Apres dans src créer fichier constants.js, puis dans src créer le dossier slices, puis dans slices, créer le fichier apiSlice.js, apres aller a store.js. Apres dans inspecter, redux (State) on voit un api.

Après dans le dossier slices, créer le fichier productsApiSlice.js, après aller à HomeScreen.js , et enfin aller modifier constants.js pour enlever un blocage de proxy.

Maintenant aller modifier productsApiSlice.js , puis ProductScreen.js.

A CE STADE TOUT EST VISIBLE GRACE A REDUX (PLUS D AXIOS)

Maintenant dans src/components créer le fichier Loader.js pour créer un loader (de chargement) (display erreur pour projet deployé), et aller faire des modifications dans HomeScreen.js ; et meme modification dans ProductScreen.js . Maintenant dans components créer Message.js (pour les messages de success ou error) ; apres faire les modif sur HomeScreen.js et ProductScreen.js

Maintenant on va ajouter l’ajout au panier (ou la carte), dans slices, créer cartSlice.js, aller apres modifier store.js. a ce stade si j ouvre redux dans inspecter, on trouve cart (dans state). Maintenant on continue a travailler sur cartSlice.js ; apres aller modifier ProductgScreen.js (il va afficher le nb du produit choisit dans le stock). On peut aller verifier dans inspecter/components (pour react), choisir ProductScreen et on peut voir le nombre dans le stock. Apres on peut faire inspecter/redux on trouvera les details des prix (tax…) (quand on clique sur le panier), apres dans inspecter/application dans le localstorage on trouve aussi des details sur le produit.

1. Il faut tenir compte pour github pour pouvoir pusher le code

* Une seule fois faire : dans VS code, connecter avec gihub : turn on settings sync (dans VS code), sign in turn on, sign in with github, connecter, apres utilisateur tarek772001 sur github est synchronisé avec VS code ; et il faut supprimer le dossier .git (une seule fois).
* git init dans terminal dans le dossier parent
* supprimer dans le fichier .gitignore les MISC
* dans source control (dans VS code) icone de gihub, ecrire le message de commit (ecemple : tarek), publish et allow et permission
* apres dans VS code il va demander public ou prive, choisir public pour pouvoir le pusher dans le serveur VPS
* Aller à <https://github.com/tarek772001?tab=repositories> pour voir les repositories

1. Docker (dans le serveur VPS) :

* Faire un Dockerfile dans frontend et un Dockerfile dans le backend
* Créer le dockercompose.yaml dans le dossier parent

Après :

1. afficher tous les containers pour les conflits de ports
2. pour créer l image
3. supprimer une image sur laquelle il y aura la modification

sudo docker container ls

sudo docker-compose up --build –d

sudo docker rm -f "Name Of Container"

1. NGINX (dans le serveur VPS) :

suivre les étapes pour déposer le projet depuis le local dans github

acceder au serveur distant et récupérer le projet depuis le lien de github

git clone  lien\_vers\_git

(il faut ajouter un token de sécurité pour le mot de passe)

dans frontend   creer le DockerFile et pareil pour backend

et le fchier yaml  compose dans le parent en vérifiant les ports

aussi changer les noms des images et containers

Questions slim :