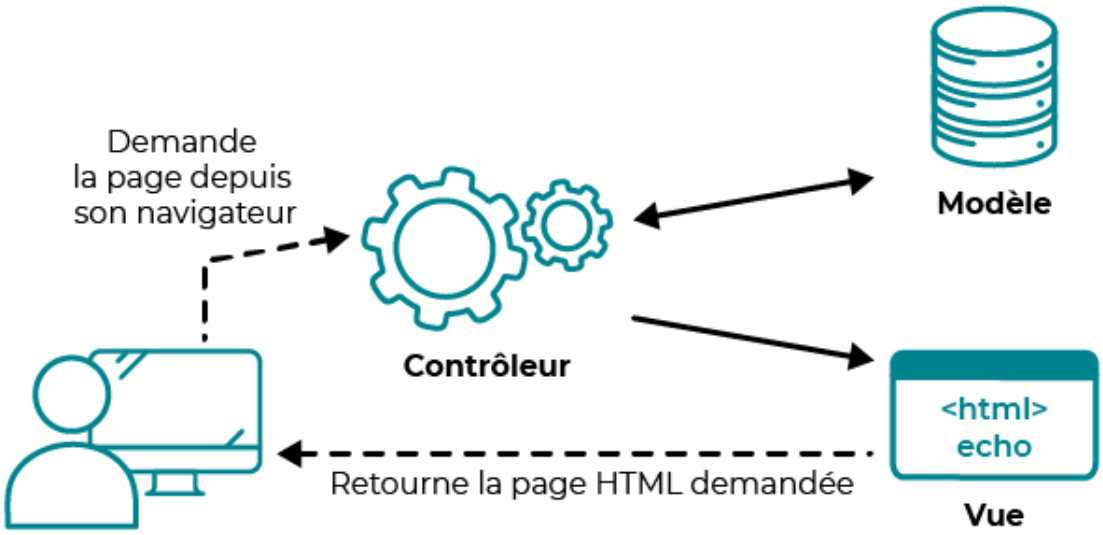
React

npm init -y
npm install express
npm install nodemon --save-dev:
lorsque je modifier au fichier server.js il na pas besoin de redemarer le serveur. Faire automatique recompiler le code lorsque je modifier le code
npx nodemon server.js
npm insttall mongoose:
est une package qui permet de comminique avec mon database.
Mongoose permet des creer les collections
Model:
est intermediate entre backend et le database
model comme les table pour comminiquer avec mon database
nombre de model depend de nombre de collection



les fonction (put, post,delete,get) se trouve dans le dossier controller .
les path se trouve dans le dossier routes .

il ya deux methodes:
methode pour chercher est : find
methode pour ajouter : save
pour parler avec database il faut utiliser le model qui apartient des fonctions

fichier server.js est :

```
const express = require('express');
const mongoose = require('mongoose');
const contactModel = require('./Models/contact');
const app = express();
const port = 3001
app.use(express.json());
app.get('/', (req, res) => {
res.send('Hello World!')
})
```

```
// tache ajouter contact
app.post('/contact/ajouter' , (req,res )=> {
//console.log(req.body);

const contactObjet ={
nom : req.body.nom,
numero : req.body.numero
}
//contact est un model
// contactObjet est objet
const contact = new contactModel(contactObjet)
//inserer data dans database
//contact.save()
contact.save( (error , createdContact) => {
if (error) {
return res.status(400).json({ error })
}
if (createdContact){
return res.status(200).json({ createdContact })
}
})
//les traitements nessecaire pour pour ajouter un nouveau contact
// res.json({message: "donner ajouter"});
})

// tache modifier contact
app.get('/contact/modifier' , (req, res)=> {
//les traitements nessecaire pour pour modifier un contact
})

// tache supprimer contact
app.get('/contact/supprimer' ,(req,res ) => {
//les traitements nessecaire pour pour suprimer un contact
})

// tache lister contact
app.get('/contact/lister' , (req,res) => {
//les traitements nessecaire pour pour lister un contact
})

mongoose.connect('mongodb://localhost:27017/test_data', {
useNewUrLParser:true,
useUnifiedTopology: true,
})

const db = mongoose.connection;
db.on('error',console.error.bind(console,'connection error'))
db.once('open', function(){
console.log('databse connected sucessfully')
})

app.listen(port, () => {
console.log(`Example app listening on port ${port}`)
})
```

```
// tache lister contact
app.get('/contact/lister', (req, res) => {
//res.send('Lister contact');
contactModel.find({}).exec((error , contactListe) => {
if(error) return res.status(400).json(error)
if(contactListe){
return res.status(200).json({contactListe})
}
})
});
```

Endpoint:

GET http://127.0.0.1:3001/contact/lister

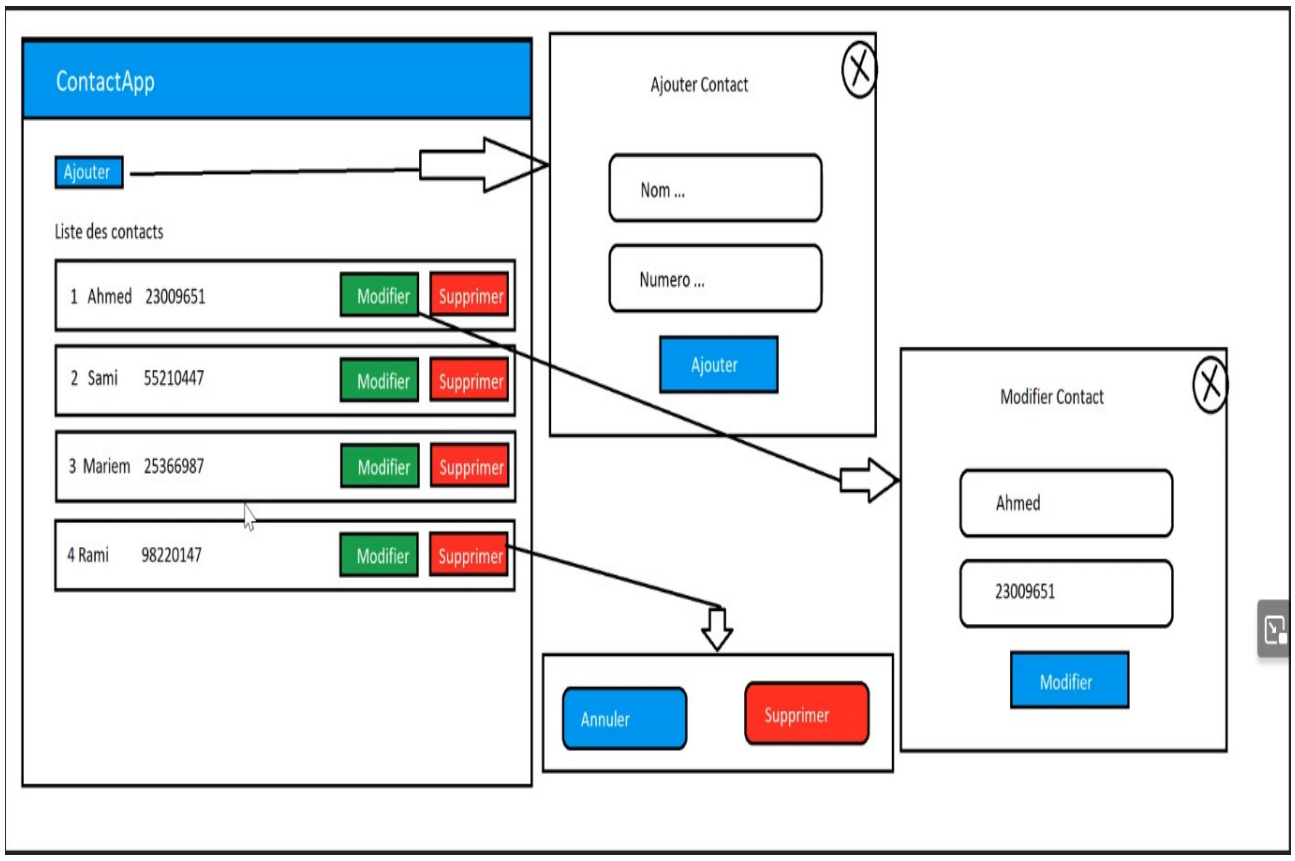
GET http://127.0.0.1:3001/contact/id/supprimer

POST http://127.0.0.1:3001/contact/id/ajouter

```
{
  “nom” = “contact”
  “numero” = “123466789”
}
```

POST http://127.0.0.1:3001/contact/id/modifier

```
{
  “nom” = “contact”
  “numero” = “123466789”
}
```



react se carecterise par single web application :
est une seul page web et le changement seffectue sur le meme page qui se trouve dans le fichier index.html

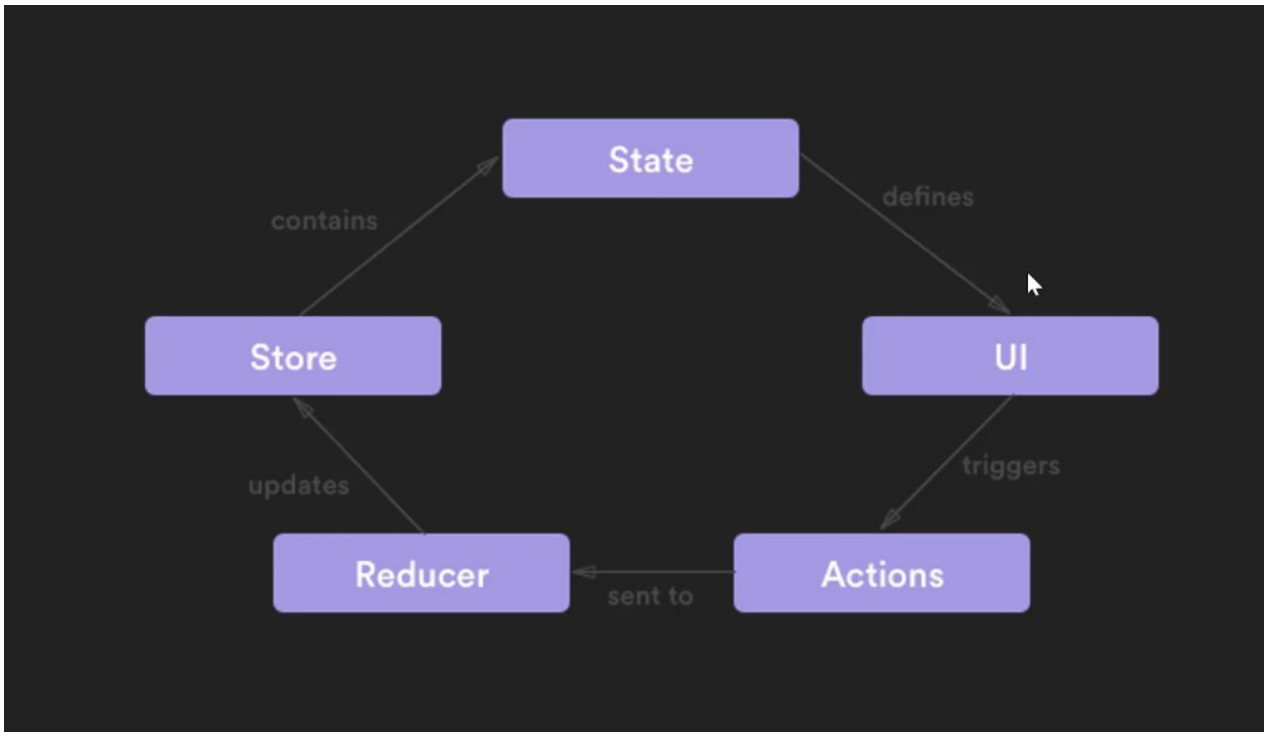
app.js est component qui contient index.html

pour faire save automatique dans vscode
sudo chown -R ubuntu:ubuntu /home/ubuntu/my-app/contactfront/
chmod -R u+w /home/ubuntu/my-app/contactfront

ctrl+k+c pour faire commentaire dans le fichier app.js cote frontend

states:
comment gerer les donees et comment consommer les APIs
les states sont des donnes

redux:
est un state management
qui va gerer les states dans le component



installer redux :
`npm i redux`

install react-redux :
npm i react-redux

install redux-thunk
npm i redux-thunk

install axios: pour prendre les donnees men le backend
qui va consommer les APIs

npm i axios

useDispatch: pour n3aytot 3ala fonction mel actions
useEffect: pour lister les contact dans la page home

le fichier contact.action:

```
import axios from 'axios';

import { contactConstants } from './constante';
//import { createSearchIndex } from '../../../backend/Models/contact';

export const listerContact = () => {
return async dispatch => {
dispatch({type : contactConstants.GET_ALL_CONTACT_REQUEST})
try{
const res = await axios.get('http://127.0.0.1:3001/contact/lister')
if (res.status === 200){
dispatch({type : contactConstants.GET_ALL_CONTACT_SUCCESS,
payload : {contact : res.data }
})
```

```
}

}catch(error){
dispatch({type : contactConstants.GET_ALL_CONTACT_FAILURE,
payload : {error : error.response }
})


}
}
}
```

fichier contact.reducer

```
const initialState = {
contacts : [],
error : null

}
export default(state= initialState , action ) =>{
switch (action.type){
case contactConstants.GET_ALL_CONTACT_REQUEST:
state = {
...state
}
break;
case contactConstants.GET_ALL_CONTACT_SUCCESS:
state = {
...state,
contacts :action.payload.contacts
}
break
case contactConstants.GET_ALL_CONTACT_FAILURE:
state = {
...state,
error :action.payload.error
}
break


default :
console.log('defalt action ');

}
return state;
}
```

1. Modèles (Models)

But : Les modèles représentent la structure de données de votre application. Ils définissent la façon dont les données sont stockées, validées et manipulées.

2. Contrôleurs (Controllers)

But : Les contrôleurs contiennent la logique d'application. Ils reçoivent les requêtes des routes, interagissent avec les modèles, et renvoient les réponses appropriées au client.

3. Routes

But : Les routes définissent les points d'accès (endpoints) de votre API. Elles mappent les URL des requêtes aux méthodes des contrôleurs.

Relation entre Modèles, Contrôleurs, et Routes

1. **Routes** : Les routes reçoivent les requêtes HTTP du client et les transmettent aux contrôleurs appropriés.
2. **Contrôleurs** : Les contrôleurs exécutent la logique d'application nécessaire pour traiter la requête, interagissent avec les modèles pour accéder ou modifier les données, et renvoient une réponse au client.
3. **Modèles** : Les modèles gèrent l'interaction avec la base de données, définissent la structure des données, et fournissent des méthodes pour les manipuler.

Les modèles gèrent les données, les contrôleurs gèrent la logique d'application, et les routes gèrent le routage des requêtes.