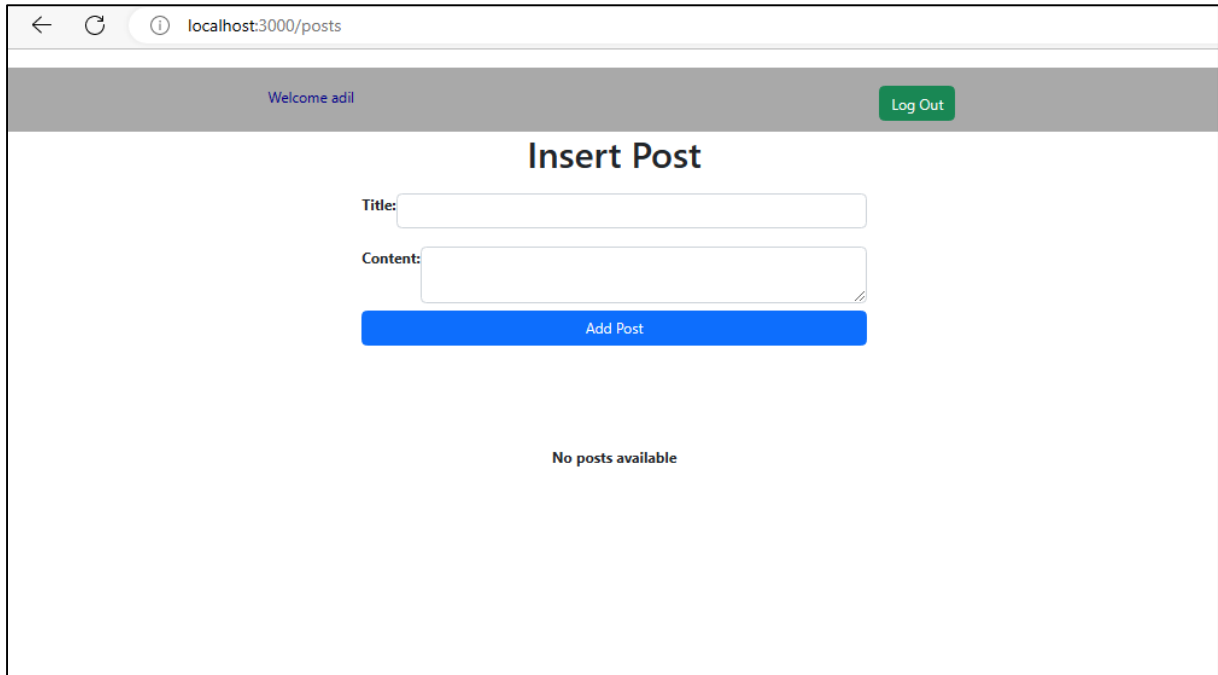


# Application web développée avec React Js+Redux Toolkit+ Middleware Thunk



localhost:3000/posts

Welcome adil [Log Out](#)

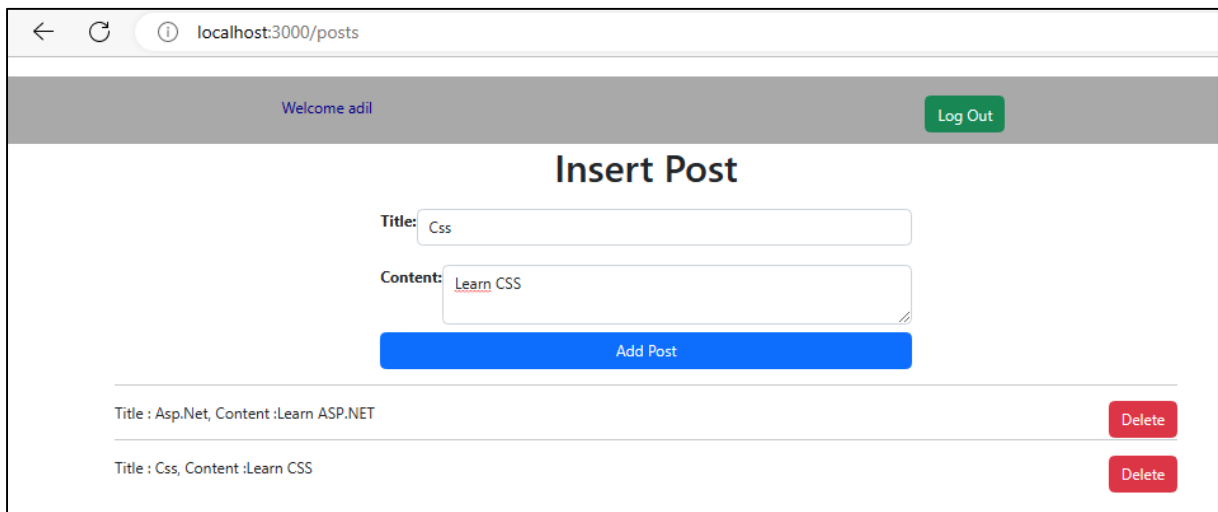
## Insert Post

Title:

Content:

[Add Post](#)

No posts available



localhost:3000/posts

Welcome adil [Log Out](#)

## Insert Post

Title:

Content:

[Add Post](#)

---

Title : Asp.Net, Content :Learn ASP.NET [Delete](#)

Title : Css, Content :Learn CSS [Delete](#)

Le travail est organisé en deux parties :

**BackEnd et le FrontEnd.**

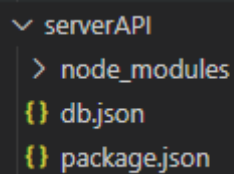
**BackEnd**

## Etape 1 :

Installer le serveur JSON

```
npm install -g json-server
```

Créer un fichier db.json avec quelques données



```
serverAPI
├── node_modules
├── db.json
└── package.json
```

```
{
  "users": [
    {
      "name": "adil adil",
      "username": "adil",
      "email": "adil@ofppt-edu.ma",
      "password": "123",
      "id": 1
    },
    {
      "name": "nada nada",
      "username": "nada",
      "email": "nada@ofppt-edu.ma",
      "password": "123",
      "id": 2
    },
    {
      "name": "nabil",
      "username": "nabil",
      "email": "email1@gmail.com",
      "password": "123",
      "id": 3
    }
  ]
}
```

```
"posts": [  
  {  
    "title": "Asp.Net",  
    "content": "Learn ASP.NET",  
    "idUser": 1,  
    "id": 1  
  },  
  {  
    "title": "Css",  
    "content": "Learn CSS",  
    "idUser": 1,  
    "id": 2  
  }  
]  
}
```

## package.json

```
{  
  "dependencies": {  
    "json-server": "^0.17.0"  
  },  
  "scripts": {  
    "start": "json-server -p 3006 -w db.json"  
  }  
}
```

# FrontEnd

Dans cette partie on veut développer une application web à l'aide de ReactJS qui gère les posts via API.

Dans cet application, vous allez utiliser Axios pour accéder à API JSON dans une application React et le middleware Redux-Thunk pour gérer les événements asynchrones

## Etape 1 :

- Créer un nouveau projet React Js

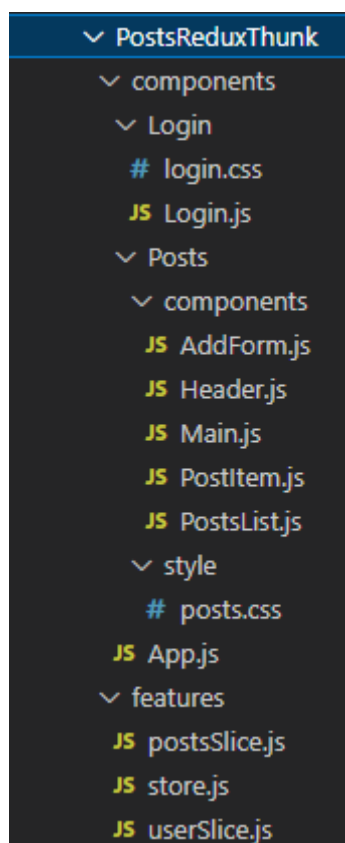
```
npx create-react-app posts-redux-thunk
```

- Installer les bibliothèques suivantes

```
npm i @reduxjs/toolkit react-redux axios
```

```
npm i react-router-dom@6
```

La structure du dossier :



## Etape 2 :

- 1- Créer le store avec la fonction **configureStore**.
- 2- Créer le slice **userSlice** contenant le state et les extraReducers. Vous pouvez utiliser l'option **extraReducers** dans **createSlice** pour écouter ces types d'action (**pending**, **fulfilled** et **rejected**)) et mettre à jour l'état.
- 3- Créer la fonction **login** à l'aide de **createAsyncThunk** pour récupérer les données à partir du serveur JSON. Définir les actions correspondantes (**pending**, **fulfilled** et **rejected**).

### userSlice.js

```
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";
import axios from "axios";
const initialState={value:null,isLoading:null,error:null}
export const login=createAsyncThunk("user/login",
async(arg,thunkAPI)=>{
  const {rejectWithValue}=thunkAPI
  try{
    // Add code here...
  }catch(error)
  {
    return rejectWithValue(error.message)
  }
})

const userSlice=createSlice({
  name:"user",
  initialState,
  reducers:{
    logout:(state)=>{
      // Add code here...
    }
  },
  extraReducers:{
    [login.pending]:(state,action)=>{
      // Add code here...
    },
    [login.fulfilled]:(state,action)=>{
      // Add code here...
    },
    [login.rejected]:(state,action)=>{
      // Add code here...
    }
  }
})
```

```

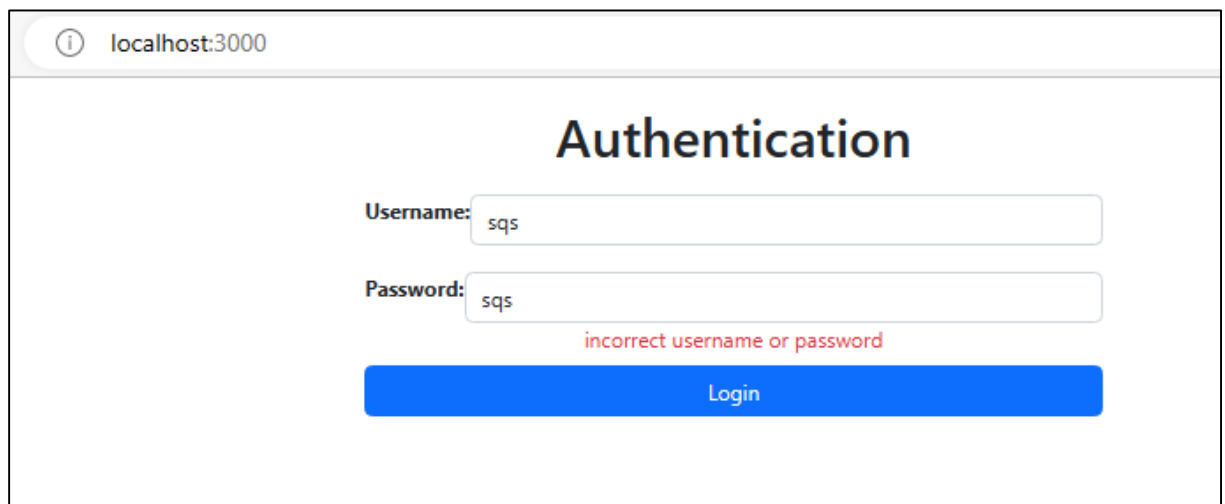
    }
  })

  export default userSlice.reducer;
  export const {logout}=userSlice.actions

```

### Etape 3 :

- Créer le composant **<Login/>**
- **Login.js** : qui authentifie l'utilisateur. Elle contient un formulaire renfermant une zone de texte, une zone de mot de passe et un bouton d'envoi. Si l'authentification échoue alors un message d'erreur s'affiche.



localhost:3000

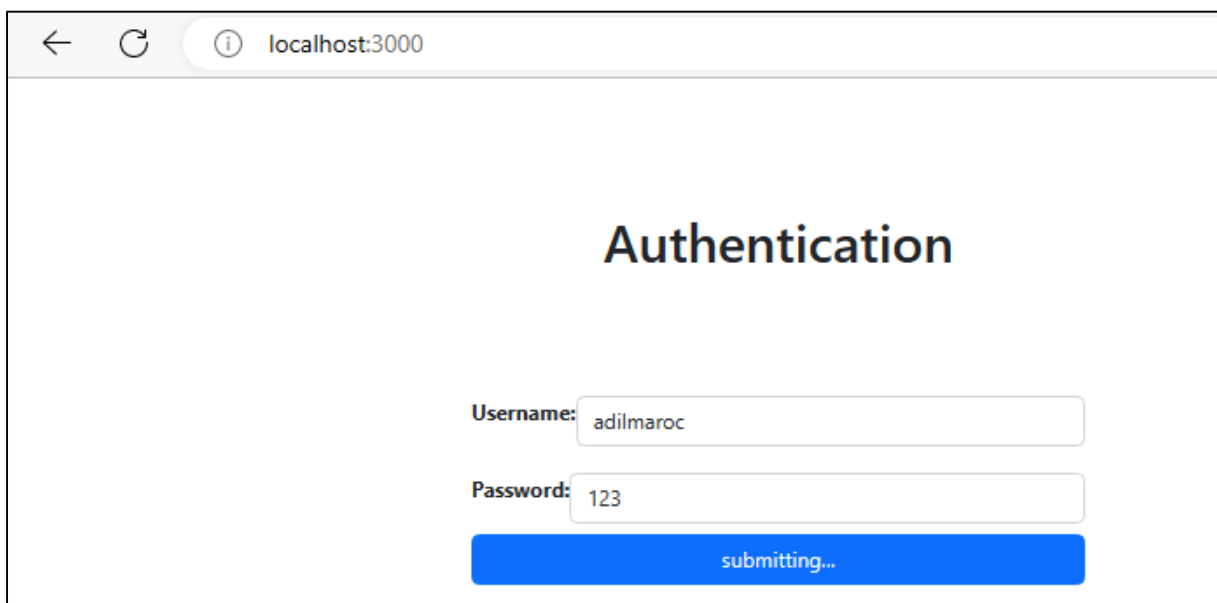
## Authentication

Username:

Password:

incorrect username or password

Login



localhost:3000

## Authentication

Username:

Password:

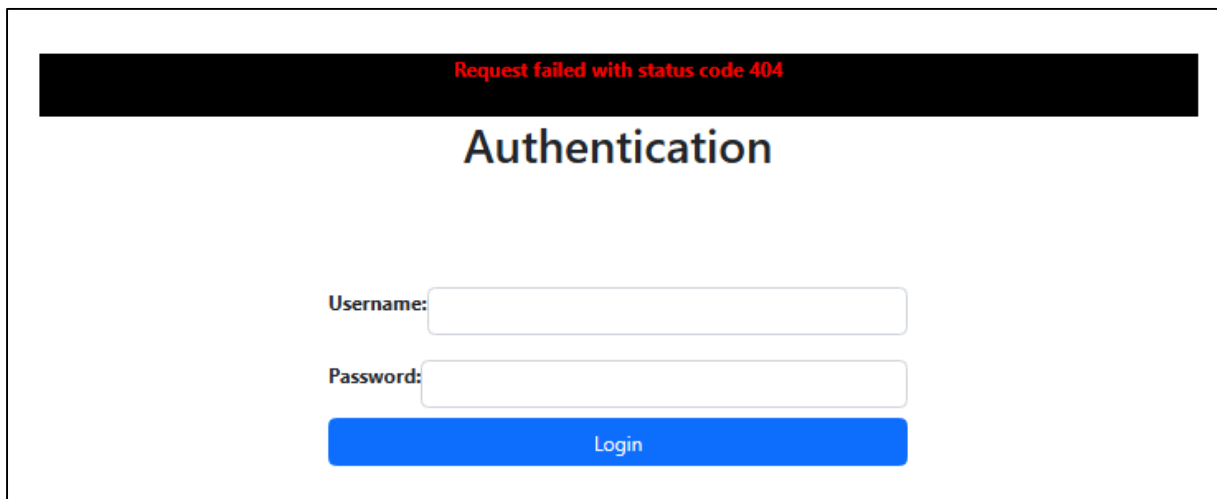
submitting...



Si le client tente d'accéder directement à la page **posts** alors qu'il n'est pas authentifié, il sera aussitôt redirigé vers la page **login**. S'il fournit un bon login et un bon mot de passe alors il sera redirigé vers la page **posts**.

La propriété **isLoading** permet d'indiquer le statut de la requête asynchrone. Elle prend la valeur **True** dans *pending* et **false** dans *fulfilled* et *rejected*.

La propriété **error** stocke l'erreur retournée par le serveur en cas d'échec.



#### Etape 4 :

Dans le component **App** vous configurez le router de l'application et aussi le **Provider**.

```
import React from "react";
import { BrowserRouter, Route, Routes } from "react-router-dom";
import Login from './Login/Login'
import Main from './Posts/components/Main'
import {Provider } from "react-redux";
import store from '../features/store'
export default function App(){

  return (

    <Provider store={store}>
      <BrowserRouter>

        <Routes>
          <Route path="" element={<Login />} />
          <Route path="/posts" element={<Main />} />
        </Routes>
      </BrowserRouter>
    </Provider>

  )
}
```

## Etape 5 :

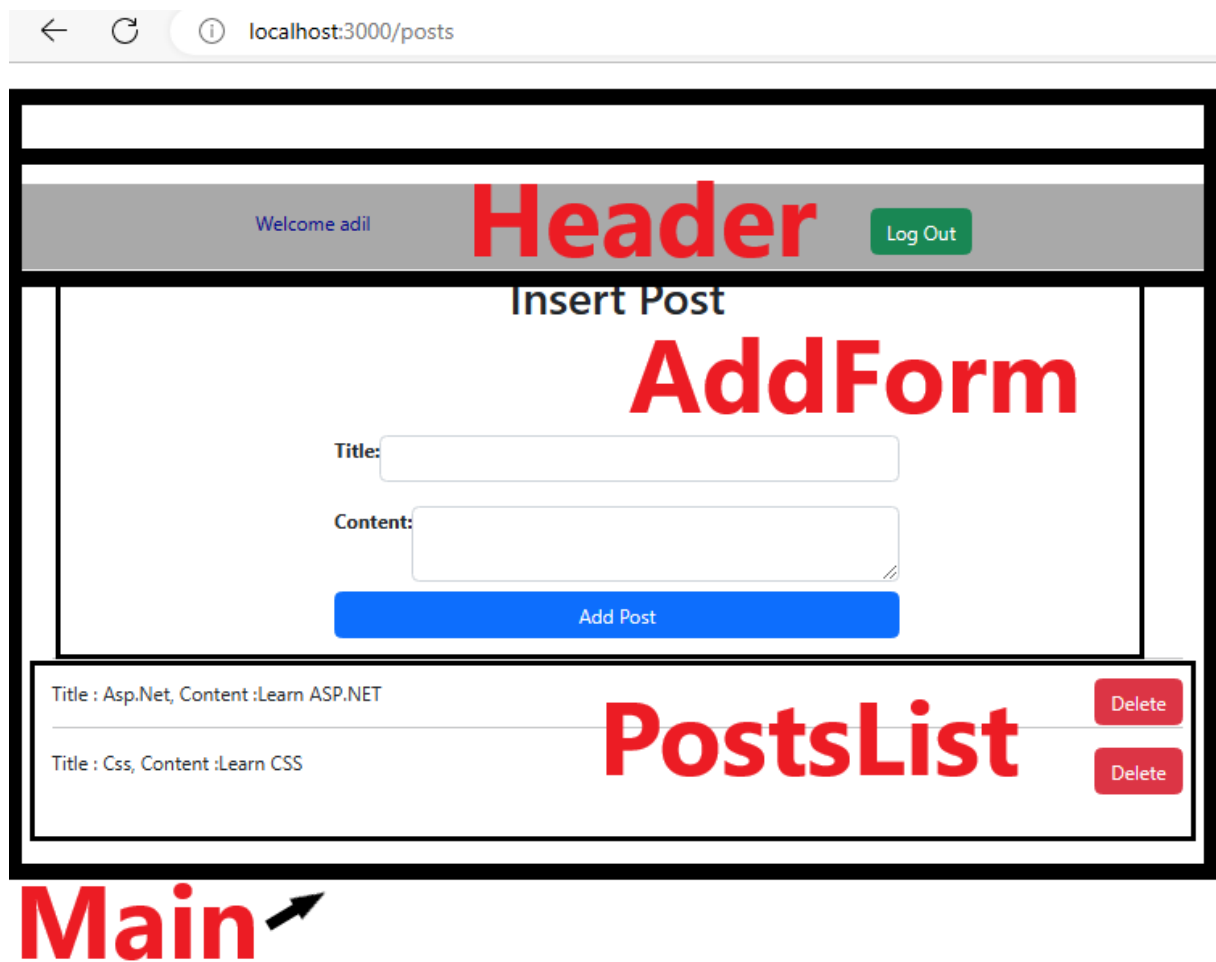
Ecrire les codes des composants :

AddForm et PostsList, Header sont des childs components de Main.

PostItem est un child component de PostsList.

Maintenant implémenter le component **Header** contenant :

- un titre h2 affichant le nom de l'utilisateur.
- **Navlink** Log Out.



## Etape 6 :

- 1- Créer le slice **postsSlice** contenant le state et les extraReducers. Vous pouvez utiliser l'option `extraReducers` dans `createSlice` pour écouter ces types d'action (`pending`, `fulfilled` et `rejected`)) et mettre à jour l'état.
- 2- Créer la fonction **getPosts**, **addPost**, **deletePost** à l'aide de `createAsyncThunk` pour récupérer les données à partir du serveur JSON. Définir les actions correspondantes (`pending`, `fulfilled` et `rejected`).
- 3- Ajouter le reducer `posts` au store.

### postsSlice.js

```
import { createAsyncThunk, createSlice, isRejectedWithValue } from
"@reduxjs/toolkit"
import axios from "axios"
const initialState={value:[],isLoading:null,err:null}
export const getPosts=createAsyncThunk('posts/getPosts',
async(_,thunkAPI)=>{
  const {rejectWithValue,getState}=thunkAPI
  try{
    const resp=await axios.get('http://localhost:3006/posts')
    const posts=resp.data.filter((item)=>item.idUser==getState().user.value.id)
    console.log('postsslice',posts)
    return posts
  }catch(error){
    return rejectWithValue(error.message)
  }
})
export const addPost=createAsyncThunk('posts/addPost',
async(post,thunkAPI)=>{
  const {rejectWithValue,getState}=thunkAPI
  try{
    post.idUser=getState().user.value.id
    const resp=await axios.post('http://localhost:3006/posts',post)
    return resp.data
  }catch(error){
    return rejectWithValue(error.message)
  }
})
export const deletePost=createAsyncThunk('posts/deletePost',
async(post,thunkAPI)=>{
  const {rejectWithValue,getState}=thunkAPI
```

```

try{
  await axios.delete(`http://localhost:3006/posts/${post.id}`)
  return post
}catch(error){
  return rejectWithValue(error.message)
}
})
const postsSlice=createSlice({
  name:"posts",
  initialState,
  extraReducers:{
    [getPosts.pending]:(state,action)=>{
      state.isLoading=true
    },
    [getPosts.fulfilled]:(state,action)=>{
      state.value=action.payload
      state.isLoading=false
    },
    [getPosts.rejected]:(state,action)=>{
      state.err=action.payload
      state.isLoading=false
    },
    [addPost.pending]:(state,action)=>{
      state.isLoading=true
    },
    [addPost.fulfilled]:(state,action)=>{
      state.isLoading=false
      state.value.push(action.payload)
    },
    [addPost.rejected]:(state,action)=>{
      state.isLoading=false
    },
    [deletePost.pending]:(state,action)=>{
      state.isLoading=true
    },
    [deletePost.fulfilled]:(state,action)=>{
      state.isLoading=false
      state.value=state.value.filter(item=>item.id!=action.payload.id)
    },
    [deletePost.rejected]:(state,action)=>{
      state.isLoading=false
    }
  }
})
export default postsSlice.reducer

```