 **Initiation à Reactjs**

« Un voyage de mille kilomètres commence par un seul pas » (citation Lao Tzu).

Sur cette bonne parole, nous commençons notre tutoriel. Dans ce tutoriel nous allons voir comment construire une application React en utilisant Typescript.

Je vais vous guider tout au long ce tutoriel étape par étape de la façon la plus simple et intelligente de développer une application React.

Nous verrons le concept React (TSX, components, props, state et lifecycle) à travers d’un exemple d’une application de gestion de contact.

Pour ce tuto nous allons utiliser l’api « **localStorage »** qui permet de stocker les données dans le navigateur sous forme de session (voir : <https://developer.mozilla.org/fr/docs/Web/API/Storage>)

**Les prérequis :**

Pour comprendre ce tuto, vous aurez besoin d'une connaissance de base en HTML, CSS et JavaScript.

**React : c’est quoi ?**

Reactjs est une librairie libre créée par Facebook en 2013 pour faciliter la création d’application monopage (SPA : Single Page Application). React n’est pas un vrai framework MVC (Model View Controller) comme Angular. Il fournit seulement la vue (V) du MVC. React est basé sur la philosophie Component-Driven Development (CDC).

Par défaut React utilise le langage javascript/ES6 (<http://www.ecma-international.org/ecma-262/6.0/>) pour développer un code orienté objet. Il peut être également utilisé avec le langage Typescript (<https://www.typescriptlang.org/docs/home.html>) qui est un superset de javascript.

React se démarque des autres concurrents sur beaucoup de points mais surtout au niveau de la performance car React utilise un virtual DOM (<https://la-cascade.io/comprendre-le-virtual-dom/>) basé sur l’algorithme « diffing » pour mettre efficacement à jour l’interface utilisateur (UI).

(<https://reactjs.org/docs/reconciliation.html>)

**Qui utilise React ?**

Facebook a développé pour leurs applications plus 50 milles composant en utilisant React. Bon nombre d’entreprise come **Twitter**, **Airbnb**, **Uber**, **Netflix**, **Instagram** etc… utilisent React.

Ceci explique la souplesse, la simplicité et surtout la performance que les entreprises ont optés pour développer leur application en React.

**Préparation de l'environnement de développement :**

Certaine préparation est requise pour le développement du React.

**Installation Node.js**

Le développement React se base sur cet outil.

Node.js a été créé en 2009 par Ryan Dahl, est une plate-forme libre en javascript orienté serveur.

Il est basé sur le moteur Javascript V8 libre qui est développé par Google.

(<https://nodejs.org/en/>)

Tester la version installée :

$ node -v

*$ : votre terminal de commande favorite*

Si tout s’est bien passé vous devez voire afficher la version du node, dans mon cas c’est la version suivante :

v10.16.3

Si ce n’est pas le cas, veuillez vérifier que Node est bien configuré dans le chemin (PATH) d’environnement de votre système.

Tester NPM :

$ npm -v

Dans mon cas c’est la version :

6.9.0

*Note* : NPM (Node Package Manager) est un gestionnaire de paquets pour Node.js et depuis la version 0.6.3 npm fait partie du Node.js est installé par défaut.

(<https://docs.npmjs.com/>)

**Installation Create-React-App**

Create-React-App est un outil standard qui facilite le développement d’application React.

Ce dernier permet de générer automatiquement la structure du projet et fournit bon nombre de scripts pour la gestion de l’application.

(<https://github.com/facebook/create-react-app>)

Pour l’installer :

$ npm install –global create-react-app

*Note* : Si pour quelque raison le npm n’est pas installé, vous avez toujours la possibilité de faire manuellement :

$ npm install --global npm

**Installation Git**

Git est nécessaire pour gérer les packages qui sont utilisés pour le développement React.

Pour l’installer, rendez sur le <https://git-scm.com/> et en fonction de votre système d’exploitation suivez les instructions. Dans mon cas, j’utilise mac os et la commande pour mac os est :

$ sudo apt-get install git

ou

$ brew install git

Vérification de la version :

$ git --version

**Installation Visual Studio Code**

VS Code est un éditeur de code développé par Microsoft.

(<https://code.visualstudio.com/>)

Pour ce tuto, nous allons utiliser cet éditeur de code pour développer notre application React.

Vous pouvez si vous le souhaitez, utiliser un autre éditeur de votre choix

**Projet création :**

Notre environnement de développement est prêt, on peut commencer à créer notre premier projet, pour cela, nous allons utiliser la commande suivante :

$ npx create-react-app react-typescript-starter --typescript

La commande **npx** fait partie de l’outil node/npm. Il est utilisé pour lancer les packages Node.js

*Note* : S’il est n’est dispo dans l’outil Node.js, installez en faisant :

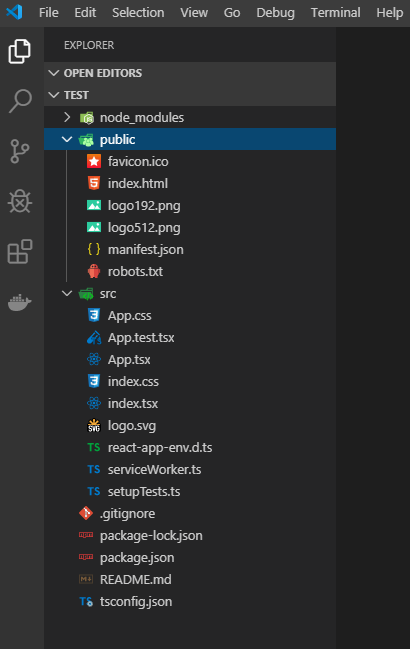
$ npm install --global npx

Le premier argument de la commande npx est le nom du package à lancer, ici c’est le package create-react-app, le deuxième argument est le nom du projet « react-typescript-starter » et le dernier argument est un flag qui indique l’utilisation du Typescript par défaut à la place du Javascript.

La génération du projet prend quelques minutes.

**Structure du projet :**

Ouvrez le dossier « react-typescript-starter » crée par l’outil create-react-app dans l’éditeur vs code.



|  |  |
| --- | --- |
| **Name** | **Description** |
| public/index.html | C’est le fichier qui est chargé dans le navigateur lorsque lancer l’application. L’élément « <div id="root"></div> » dans lequel le code React est exécuté. L’id « root » est utilisé par le fichier src.index.tsx pour le rendu html et javascript.  Plus d’info : <https://github.com/jantimon/html-webpack-plugin#options> |
| src/index.tsx | Le fichier typescript qui est responsable de démarrer l’application React. Il utilise le dom id « root » qui se trouve dans le fichier public/index.html |
| src/App.tsx | C’est le composant root qui contient le code html et typescript afficher. Il peut contenir d’autres composant React. |
| package.json | Le fichier de config pour la gestion des packages et dans celui-ci vous trouverez les scripts par défaut pour lancer l’application en local, publication, etc… |

**Lancer l’application :**

Pour tester l’application

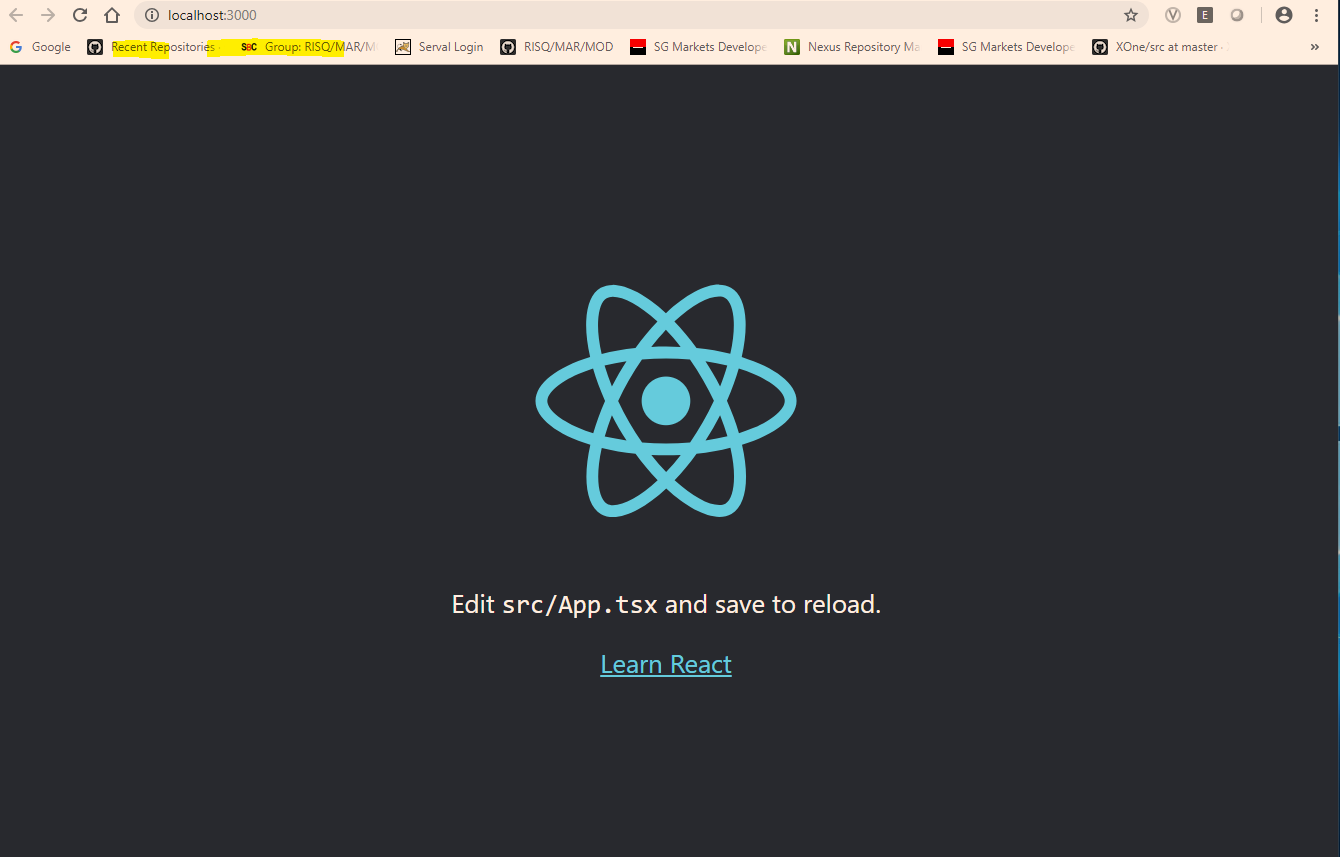
$ cd react-typescript-starter

$ npm start

Cette commande lance l’application et si tout se passe bien vous verrez la page ci-dessous affichée :

La page est ouvert sur <http://localhost:3000>, le port par défaut est 3000. Vous pouvez changer le port en créant un fichier .env dans la racine du projet, c’est-à-dire au même niveau que « package.json »

Dans ce fichier écrivez la ligne : « PORT=400 ». Pour voir le changement il suffit de relancer l’appli.



**Notre première app CRUD :**

Nous allons créer une application pour gérer le contact. Un formulaire d’ajout et un tableau qui affichera la liste des contacts.

**Framework CSS Bootstrap :**

Pour la présentation de notre application nous allons utiliser le framework css boostrap (<https://getbootstrap.com/>).

$ npm install bootstrap

Cette commande installe le packages dans le dossier node\_modules de notre projet puis ajoute une ligne dans le « package.json » dans la section « dependencies »

Pour l’interface, je me suis basé sur l’exemple bootstrap qui se trouve sur le lien ci-après <https://getbootstrap.com/docs/4.4/examples/dashboard/>

**Font Awesome Icons :**

$ npm install font-awesome

(<https://fontawesome.com/icons?d=gallery&m=free>)

**Préparation de la page :**

Créer un dossier « styles », un dossier « models » et un dossier « components » dans le dossier « src »

Dans le dossier « src/styles » ajoutez un fichier « dashboard.css » puis copier le code ci-dessous :

body {

    font-size: .875rem;

  }

  .feather {

    width: 16px;

    height: 16px;

    vertical-align: text-bottom;

  }

  /\*

   \* Sidebar

   \*/

  .sidebar {

    position: fixed;

    top: 0;

    bottom: 0;

    left: 0;

    z-index: 100; /\* Behind the navbar \*/

    padding: 48px 0 0; /\* Height of navbar \*/

    box-shadow: inset -1px 0 0 rgba(0, 0, 0, .1);

  }

  .sidebar-sticky {

    position: relative;

    top: 0;

    height: calc(100vh - 48px);

    padding-top: .5rem;

    overflow-x: hidden;

    overflow-y: auto; /\* Scrollable contents if viewport is shorter than content. \*/

  }

  @supports ((position: -webkit-sticky) or (position: sticky)) {

    .sidebar-sticky {

      position: -webkit-sticky;

      position: sticky;

    }

  }

  .sidebar .nav-link {

    font-weight: 500;

    color: #333;

  }

  .sidebar .nav-link .feather {

    margin-right: 4px;

    color: #999;

  }

  .sidebar .nav-link.active {

    color: #007bff;

  }

  .sidebar .nav-link:hover .feather,

  .sidebar .nav-link.active .feather {

    color: inherit;

  }

  .sidebar-heading {

    font-size: .75rem;

    text-transform: uppercase;

  }

  /\*

   \* Content

   \*/

  [role="main"] {

    padding-top: 133px; /\* Space for fixed navbar \*/

  }

  @media (min-width: 768px) {

    [role="main"] {

      padding-top: 48px; /\* Space for fixed navbar \*/

    }

  }

  /\*

   \* Navbar

   \*/

  .navbar-brand {

    padding-top: .75rem;

    padding-bottom: .75rem;

    font-size: 1rem;

    background-color: rgba(0, 0, 0, .25);

    box-shadow: inset -1px 0 0 rgba(0, 0, 0, .25);

  }

  .navbar .form-control {

    padding: .75rem 1rem;

    border-width: 0;

    border-radius: 0;

  }

  .form-control-dark {

    color: #fff;

    background-color: rgba(255, 255, 255, .1);

    border-color: rgba(255, 255, 255, .1);

  }

  .form-control-dark:focus {

    border-color: transparent;

    box-shadow: 0 0 0 3px rgba(255, 255, 255, .25);

  }

Ce css est dans l’exemple <https://getbootstrap.com/docs/4.4/examples/dashboard/>

Je juste récupérer le css pour utiliser dans notre application.

Créer un autre fichier css « common.css » toujours dans le dossier « src/styles » puis copier le code suivant :

.datatable-toolbar-actions {

  padding: "15px 0px 10px";

}

.btn-grey {

  border-color: #acb5c3;

  color: #fafcfb;

  background-color: #acb5c3;

  margin-right: 5px;

  cursor: pointer;

}

.btn-blue {

  border-color: #007bff;

  color: #fafcfb;

  background-color: #007bffa6;

  margin-right: 5px;

  cursor: pointer;

}

.btn-navy {

  border-color: #354860;

  color: #fafcfb;

  background-color: #354860;

  margin-right: 5px;

  cursor: pointer;

}

.btn-circle {

  border-radius: 25px !important;

  overflow: hidden;

}

Ce css est fait par moi 😊.

Dans le dossier « src/models », nous allons créer une interface typescript « contact.ts » qui sera notre objet qui va gérer le contact.

Insérer le code suivant dans ce fichier :

export interface IContact {

    id: number; // Identifier uniq

    name: string; // Name or fullname

    email: string; // Email

    dateOfBirth: string; // Date fo birth

}

Cette interface est notre model pour binder le formulaire contact nous allons créer prochainement.

**Première composant React :**

Remplacez le contenu du fichier « src/App.tsx » par le contenu suivant :

import \* as React from "react";

import "bootstrap/dist/css/bootstrap.min.css";

import "font-awesome/css/font-awesome.min.css";

import "./styles/dashboard.css";

import "./styles/common.css";

class App extends React.Component<{}, {}> {

  render() {

    return (

      <>

        <nav className="navbar navbar-dark fixed-top bg-dark flex-md-nowrap p-0 shadow">

          <a className="navbar-brand col-sm-3 col-md-2 mr-0" href="/#/">

            <i className="fa fa-home">{``}</i>React Typescript App

          </a>

        </nav>

        <div className="container-fluid">

          <div className="row">

            <nav className="col-md-2 d-none d-md-block bg-light sidebar">

              <div className="sidebar-sticky">

                <ul className="nav flex-column">

                  <li className="nav-item">

                    <a className="nav-link active" href="/">

                      <i className="fa fa-users">{``}</i>Users{` `}

                      <span className="sr-only">(current)</span>

                    </a>

                  </li>

                </ul>

              </div>

            </nav>

          </div>

          <div role="main" className="col-md-9 ml-sm-auto col-lg-10 px-4">

            {/\* <div className="row" style={{ paddingTop: "2px" }}>

              <div className="col-md-12">{``}</div>

            </div> \*/}

            <div className="d-flex justify-content-between flex-wrap flex-md-nowrap align-items-center pt-3 pb-2 mb-3 border-bottom">

              <h1 className="h2">Contact</h1>

            </div>

            <div className="row" style={{ paddingTop: "50px" }}>

              <div className="col-md-12">{``}</div>

            </div>

            <div className="row">

              <div className="col-md-12" style={{ marginBottom: "10px" }}>

                My first React Component

              </div>

            </div>

          </div>

        </div>

      </>

    );

  }

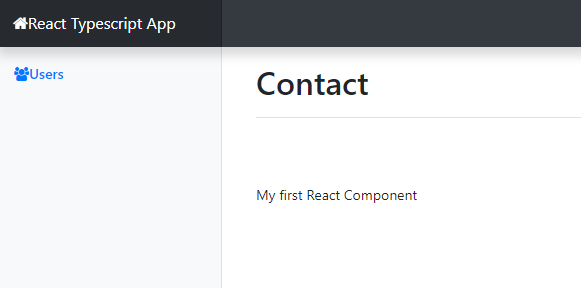
}

export default App;

Ouvrez le terminal et tapez la commande :

$ npm start

Si tout se passe vous verrez l’écran ci-dessous :



**Explications :**

Le fichier App.tsx contient le composant React « App ». L’extension tsx (XML-like) est extension syntaxique typescript. Cela permet de créer une vue html avec les expressions typescript.

En premier lieu on importe la librairie « React » :

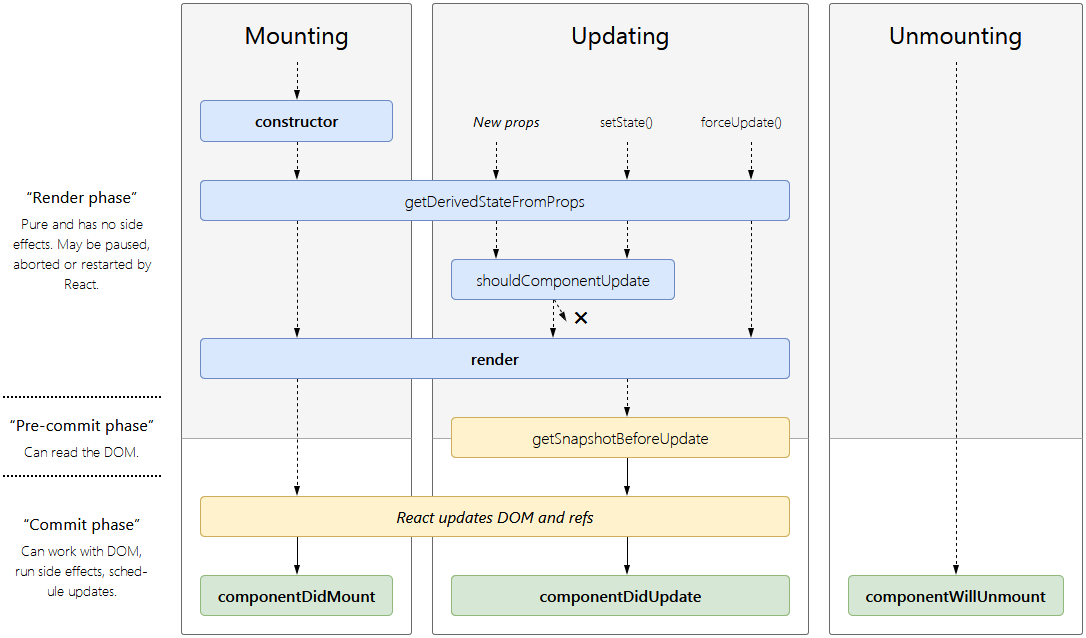
import \* as React from "react";

Puis on importe les styles css.

React autorise de créer des composants en tant que classes ou fonctions, notre composant App est un composant classe qui hérite de la classe « React.Component ». Le composant classe possède des « props » (propriétés), « state » (état) et ainsi que le cycle de vie. Le composant classe doit implémenter la méthode « render() », cette méthode est appelé par React pour afficher le contenu à l’utilisateur.

Le cycle de vie d’un composant :

* Initialisation
* Mise à jour
* Destruction



(image : <https://stackoverflow.com/questions/52123837/what-are-the-lifecycle-hooks-in-react-when-they-are-used?noredirect=1&lq=1>)

À chaque mise à jour du « state », la méthode « render() » est appelée. Pour mettre à jour le «state » vous devez utiliser la méthode « setState() ».

**Préparation service « ContactLocalStorageService » :**

Créer un dossier « services » dans « src ». Créer un fichier « contactLocalStorageService.ts » dans le dossier « services » puis ajouter le code ci-dessous :

import { IContact } from "../models/contact";

class ContactLocalStorage {

  private key: string = "contacts"

  //#region query

  fetchContacts = () => {

    const dbcontacts = localStorage.getItem(this.key);

    if (dbcontacts) {

      const contacts: IContact[] = JSON.parse(dbcontacts);

      return contacts;

    }

    return null;

  };

  getById = (id: number) => {

    const dbcontacts = this.fetchContacts()

    if (dbcontacts) {

      const contact = dbcontacts.filter(item => item.id === id)

      return contact[0]

    }

    return null

  }

  //#endregion

  //#region command

  saveContact = (contact: IContact) => {

    let auto\_id: number = 1;

    // console.log(`localStorageService::saveContact=>contact ${JSON.stringify(contact)}`);

    let contacts: IContact[] = [];

    const dbcontacts = this.fetchContacts();

    if (dbcontacts) {

      contacts = dbcontacts;

      auto\_id = contacts.length + 1

    }

    contact.id = auto\_id;

    contacts.push(contact);

    this.saveContacts(contacts)

  };

  deleteById = (id: number) => {

    const dbcontacts = this.fetchContacts()

    if (dbcontacts) {

      const contacts = dbcontacts.filter(item => item.id !== id)

      this.saveContacts(contacts)

    }

  }

  saveContacts = (contacts: IContact[] ) => {

    localStorage.setItem(this.key, JSON.stringify(contacts));

  }

  clearData = () => {

    localStorage.removeItem(this.key)

  }

  //#endregion

}

const ContactLocalStorageService = new ContactLocalStorage();

export default ContactLocalStorageService;

Ce service nous permet de gérer le stockage des informations du contact dans le localStorage.

**Composant formulaire ajout contact :**

Nous allons créer notre composant Contact en utilisant cette fois-ci un composant fonction « React.FunctionComponent ». Nous allons utiliser les nouvelles fonctionnalités React Hooks.

Hooks est introduite à partir de la version React 16.8. Hooks possède un état local et d’autres fonctionnalités sans passer par une classe.

(<https://fr.reactjs.org/docs/hooks-intro.html>)

Nous allons explorer deux méthodes Hook « useState » et « useEffect ». Vous pouvez créer votre propre hook si vous le souhaitez.

Ok maintenant, il est de créér notre composant « ContactCreation », créons un dossier « contact » dans « src/components » ensuite un fichier « contact.creation.tsx » dans le dossier « contact » puis copiez le code ci-dessous dans ce fichier :

import \* as React from "react";

import { IContact } from "../../models/contact";

interface IOwnProps {

  handleSaveContact: (contact: IContact) => void;

}

const ContactCreation: React.FunctionComponent<IOwnProps> = (

  props: IOwnProps

) => {

  const initialContcatState: IContact = {

    id: 0,

    name: "",

    email: "",

    dateOfBirth: ""

  };

  // contact is a state variable

  const [contact, setContact] = React.useState(initialContcatState);

  // Similar to componentDidMount, componentDidUpdate and componentWillUnmount

  React.useEffect(() => {

    // Update document title via browser API

    if (contact.name !== "") {

      document.title = `The entered name is ${contact.name}`;

    }

  }, [contact.name]); // Execute the effect only if contact.name has changed

  const handleReset = () => {

    setContact(initialContcatState);

  }

  const handleSubmit = (

    e: any // React.SyntheticEvent<HTMLInputElement | HTMLButtonElement>

  ) => {

    e.preventDefault();

    // console.log(`ContactCreation::handleSubmit=>contact ${JSON.stringify(contact)}`);

    props.handleSaveContact(contact);

    handleReset();

  };

  /\*\*

   \* Common input change event

   \* When the field is entered an event is raised and update the state

   \*/

  const handleInputChange = (

    e: React.SyntheticEvent<HTMLInputElement | HTMLButtonElement>

  ): void => {

    e.preventDefault();

    const target = e.target as any;

    setContact(contact => ({ ...contact, [target.name]: target.value }));

  };

  return (

    <>

      <form onSubmit={handleSubmit}>

        <div className="card bg-light mb-3">

          <div className="card-header">New contact</div>

          <div className="card-body">

            <div className="card-text">

              <div className="form-row">

                <div className="form-group col-md-4">

                  <label>Name</label>

                  <input

                    type="text"

                    name="name"

                    value={contact.name}

                    placeholder="Enter name"

                    className="form-control"

                    required

                    onChange={handleInputChange}

                  />

                </div>

                <div className="form-group col-md-4">

                  <label>Email address</label>

                  <input

                    type="email"

                    name="email"

                    value={contact.email}

                    placeholder="Enter email"

                    className="form-control"

                    aria-describedby="emailHelp"

                    required

                    onChange={handleInputChange}

                  />

                  <small id="emailHelp" className="form-text text-muted">

                    We'll never share your email with anyone else.

                  </small>

                </div>

                <div className="form-group col-md-4">

                  <label>Date of birth</label>

                  <input

                    type="date"

                    max="2020-12-01"

                    min="1900-12-01"

                    name="dateOfBirth"

                    value={contact.dateOfBirth}

                    placeholder="Enter date of birth"

                    className="form-control"

                    required

                    onChange={handleInputChange}

                  />

                </div>

              </div>

              <input className="btn btn-outline-primary" type="submit" value="Save" />{` `}

              <input className="btn btn-outline-warning" type="button" value="Reset" onClick={handleReset} />{` `}

            </div>

          </div>

        </div>

      </form>

    </>

  );

};

export default ContactCreation;

Voilà comment on crée une fonction composant, et on lui passe une interface de propriétés « IOwnProps » avec deux méthodes « handleSaveContact: (contact: IContact) => void » avec un paramètre contact du type IContact.

Lorsque que ce composant est appelé depuis un autre composant il ne faut oublier de lui transmettre les propriétés pour le bon fonctionnement de celui-ci. On verra par la suite comment l’utiliser.

Regardons un peu la ligne :

const [contact, setContact] = React.useState(initialContcatState);

On déclare la fonctionnalité « React.useState ». Ce hook est utilisé dans une fonction composant pour gérer l’état locale du composant. Car les hooks ne peuvent pas être utilisés dans une classe de composant.

Il y a des règles qui existent à propos des hooks que vous pouvez consulter sur ce lien <https://reactjs.org/docs/hooks-rules.html>.

Pour notre hook, nous avons crée une variable contact et une méthode setContact pour modifier la valeur de contact et pour finir on l’initialise la variable avec l’objet « initialContcatState ».

Le dernier point :

Pour utiliser ce composant ailleurs, il faut utiliser l’instruction « export ». Et le mot clé « default » signifie que vous pouvez utiliser plusieurs exports dans le même fichier.

export default ContactCreation;

Exemple utilisation avec default :

import ContactCreation from "./components/contact/contact.creation";

Exemple utilisation sans default :

import { ContactCreation } from "./components/contact/contact.creation";

Sans le mot clé default vous devez utiliser les accolades.

**Import composant « ContactCreation » :**

Ouvrez le fihcier App.tsx, commencez à importer le composant :

import ContactCreation from "./components/contact/contact.creation";

Ensuite ajoute le composant à la ligne « My first React Component » :

<ContactCreation />

Sauvez le fichier et lancer la commande ci-dessous si ce n’est pas fait :

$npm start

Le compilateur échouera avec l’erreur suivante :

Type '{}' is missing the following properties from type 'IOwnProps': handleSaveContactTS2739

L’erreur est explicite le composant attends les propriétés « handleSaveContact »  .

Remplacez la ligne par ce code :

<ContactCreation handleSaveContact={this.handleSaveContact} />

Méthode « handleSaveContact » :

Importer les fichiers nécessaires

import { IContact } from "./models/contact";

import ContactLocalStorageService from "./services/contactLocalStorageService";

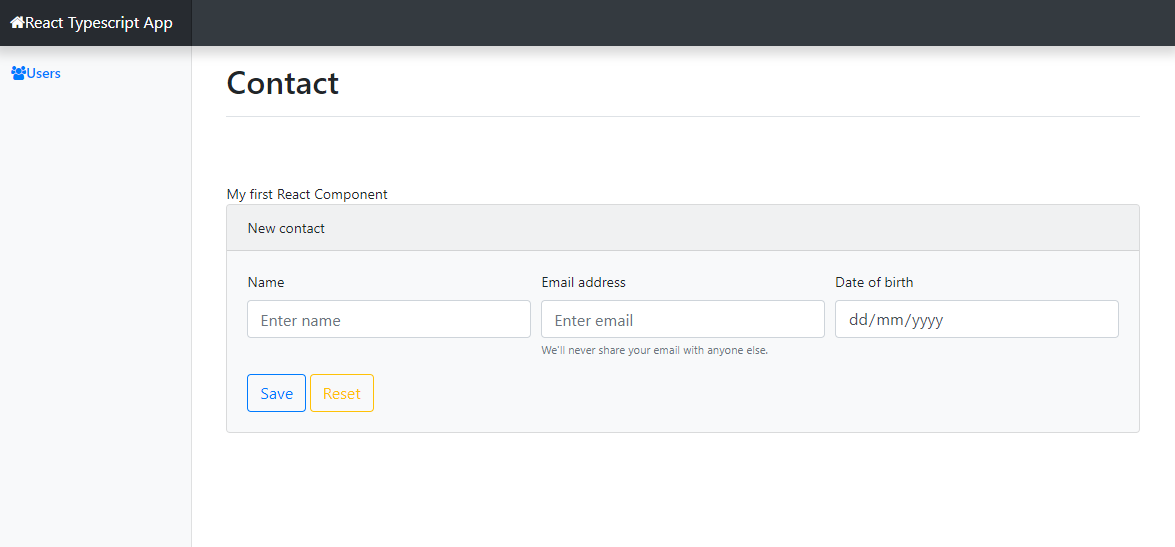
Puis ajoutez la méthode après la méthode render()

handleSaveContact = (contact: IContact) => {

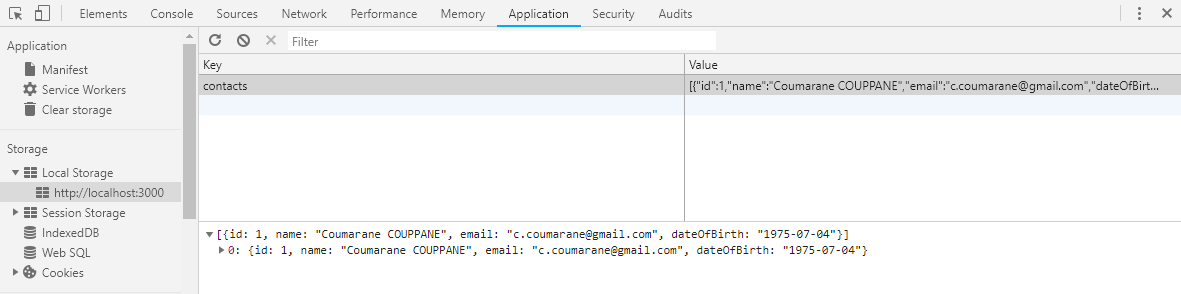
    ContactLocalStorageService.saveContact(contact);

  };

Nous avons fini notre componsat « ContactCreation ». Si vous sauvez et si tout s’est bien passé vous verrez l’écran suivant sur le navigateur :



Replissez le formulaire puis sauvez en cliquant sur le bouton « Save ». Les données sont sauvegardées dans « Local Storage » du navigateur. Appuyez sur F12 (chrome dans mon cas) puis « Application->Storage->Local Storage » cliquez sur la clé « contacts » et vous verrez vos données (voir l’écran ci-dessous).



**Composant liste des contacts :**

Plutôt que de voir les données dans « Local Storage » à chaque fois, c’est mieux qu’on crée un composant pour afficher ces données sous forme de tableau.

Dans le dossier « src/components/contact » créez un fichier « contact.list.tsx » puis coller le code suivant :

import \* as React from "react";

import { IContact } from "../../models/contact";

interface IOwnProps {

 datas: IContact[];

 handleDelete: (id: number) => (e: React.MouseEvent) => void;

}

const ContactList: React.FunctionComponent<IOwnProps> = (props) => {

  return (

    <>

      <h6>Contact list</h6>

      <div className="table-responsive">

        <table className="table table-striped table-sm">

          <thead>

            <tr>

              <th>#</th>

              <th>Name</th>

              <th>Email</th>

              <th>Date of birth</th>

              <th>Actions</th>

            </tr>

          </thead>

          <tbody>

            {props.datas &&

              props.datas.length > 0 &&

              props.datas.map((item, index) => {

                return (

                  <React.Fragment key={index}>

                    <tr>

                      <td>{item.id}</td>

                      <td>{item.name}</td>

                      <td>{item.email}</td>

                      <td>{item.dateOfBirth}</td>

                      <th>

                        <i className="fa fa-trash" onClick={props.handleDelete(item.id)} style={{color: 'red', cursor: 'pointer'}}>&nbsp;</i>

                        </th>

                    </tr>

                  </React.Fragment>

                );

              })}

          </tbody>

        </table>

      </div>

    </>

  );

};

export default ContactList;

Ce composant a besoin de deux propriétés une variable liste contact « datas: IContact[] » et une méthode de « handleDelete: (id: number) => (e: React.MouseEvent) => void » qui prends en paramètre l’id contact à supprimer.

Et voici le code finale du fichier App.tsx :

import \* as React from "react";

import ContactCreation from "./components/contact/contact.creation";

import { IContact } from "./models/contact";

import ContactLocalStorageService from "./services/contactLocalStorageService";

import "bootstrap/dist/css/bootstrap.min.css";

import "font-awesome/css/font-awesome.min.css";

import "./styles/dashboard.css";

import "./styles/common.css";

import ContactList from "./components/contact/contact.list";

interface IOwnState {

  contacts: IContact[];

}

class App extends React.Component<{}, IOwnState> {

  constructor(props: {}) {

    super(props);

    this.state = {

      contacts: [],

    };

  }

  componentDidMount() {

    this.loadContacts();

  }

  loadContacts = () => {

    const dbcontacts = ContactLocalStorageService.fetchContacts();

    this.setState({ contacts: dbcontacts! });

  };

  render() {

    return (

      <>

        <nav className="navbar navbar-dark fixed-top bg-dark flex-md-nowrap p-0 shadow">

          <a className="navbar-brand col-sm-3 col-md-2 mr-0" href="/#/">

            <i className="fa fa-home">{``}</i>React Typescript App

          </a>

        </nav>

        <div className="container-fluid">

          <div className="row">

            <nav className="col-md-2 d-none d-md-block bg-light sidebar">

              <div className="sidebar-sticky">

                <ul className="nav flex-column">

                  <li className="nav-item">

                    <a className="nav-link active" href="/">

                      <i className="fa fa-users">{``}</i>Users{` `}

                      <span className="sr-only">(current)</span>

                    </a>

                  </li>

                </ul>

              </div>

            </nav>

          </div>

          <div role="main" className="col-md-9 ml-sm-auto col-lg-10 px-4">

            {/\* <div className="row" style={{ paddingTop: "2px" }}>

              <div className="col-md-12">{``}</div>

            </div> \*/}

            <div className="d-flex justify-content-between flex-wrap flex-md-nowrap align-items-center pt-3 pb-2 mb-3 border-bottom">

              <h1 className="h2">Contact</h1>

            </div>

            <div className="row" style={{ paddingTop: "50px" }}>

              <div className="col-md-12">{``}</div>

            </div>

            <div className="row">

              <div className="col-md-12" style={{ marginBottom: "10px" }}>

                My first React Component

                <ContactCreation handleSaveContact={this.handleSaveContact} />

              </div>

            </div>

            <div className="row">

              <div className="col-md-12" style={{ marginBottom: "10px" }}>

              <ContactList

                datas={this.state.contacts}

                handleDelete={this.handleDelete}

              />

              </div>

            </div>

          </div>

        </div>

      </>

    );

  }

  handleSaveContact = (contact: IContact) => {

    ContactLocalStorageService.saveContact(contact);

    this.loadContacts();

  };

  handleDelete = (id: number) => (e: React.MouseEvent) => {

    ContactLocalStorageService.deleteById(id);

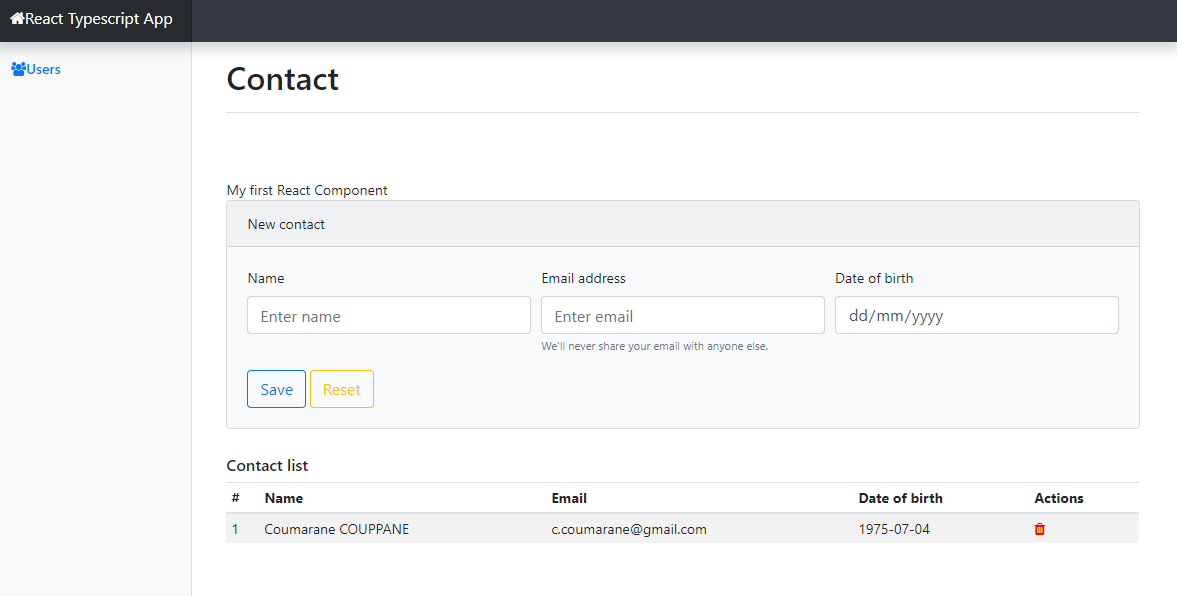
    this.loadContacts();

  };

}

export default App;

La page finale est :



Nous avons terminé l’application React. Il reste juste la fonctionnalité pour éditer un contact. Je vous laisse développer, j’ai confiance en vous les geeks.

**BRAVO !!! VOUS ÊTES LE MEILLEUR …**

Code source sur github :

<https://github.com/coumarane/react-typescript-starter>