Developpement Web

Chapitre 6 : JS/TS en Front-End avec React.js

M. Mohammed BELATAR

m.belatar@emsi.ma

avec Mme. FZ MOUTAI et M. YF EBOBISSE

3ème Année Ingénierie en Informatique et Réseaux

24 décembre 2024

Sommaire

Qu'est-ce que React ?

- **React** est une bibliothèque JavaScript pour construire des interfaces utilisateur interactives.
- Développé par **Facebook** en 2013 et utilisé dans des projets célèbres comme Instagram, WhatsApp Web.
- Basé sur une approche déclarative : Vous décrivez "quoi" afficher et React s'occupe du "comment".

Principe Fondamental :

- Diviser une interface en **composants réutilisables**.
- Manipuler efficacement l'affichage grâce au **Virtual DOM**.

Pourquoi utiliser React ?

Avantages clés :

- **Réutilisabilité :** Évitez la duplication de code grâce aux composants.
- **Performances optimisées :** React utilise un **Virtual DOM** pour minimiser les modifications du DOM réel.
- **Simplicité :** Un flux de données clair et prévisible avec unidirectional data flow.
- **Écosystème riche :** Compatible avec des bibliothèques comme Redux, React Router.
- **Facilité d'apprentissage :** Basé sur JavaScript avec une syntaxe intuitive (**JSX**).

Pourquoi apprendre React avec TypeScript ?

- Améliore la robustesse du code avec un typage statique.
- Réduit les erreurs et facilite la maintenance.

Introduction à JSX (JavaScript XML)

- **Qu'est-ce que JSX ?**
  - Une extension syntaxique pour JavaScript.
  - Permet d'écrire du HTML directement dans le code JavaScript.
  - Traduit en appels `React.createElement()` pendant la compilation.
- **Pourquoi utiliser JSX ?**
  - Syntaxe lisible et proche du HTML.
  - Simplifie la création des interfaces utilisateur.
  - Supporte les expressions JavaScript intégrées.

Exemple avec JSX

```
 const element = <h1>Bienvenue dans JSX!</h1>;
ReactDOM.render(element, document.getElementById("root"));
```

Note : JSX n'est pas obligatoire mais est largement utilisé avec React.

Caractéristiques de JSX

- **Expressions JavaScript dans JSX :**
  - Utilisez des accolades {} pour insérer des expressions JS.

Exemple

```
const nom = "Aïcha";
const element = <h1>Bonjour, {nom}!</h1>;
```

- **JSX doit avoir un parent unique :**

Exemple

```
return ( <div> <h1>Bonjour!</h1>
  <p>Ceci est un exemple.</p> </div> );
```

- **Attributs en JSX :** Utilisez le camelCase pour les noms d'attributs.

Exemple

```
const element = <button onClick={handleClick}>Cliquez
ici</button>;
```

Composants Réutilisables

- **Qu'est-ce qu'un composant ?**
  - Un composant est une partie autonome et réutilisable de l'interface utilisateur.
  - Permet de diviser l'application en petites unités logiques.
- **Pourquoi la réutilisation est-elle importante ?**
  - Réduit la duplication du code.
  - Facilite la maintenance et l'évolutivité de l'application.
- **Exemple :**

Composant Réutilisable

```
 function Bouton(props) {
  return <button>{props.label}</button>;
}
export default Bouton;
```

Avantage : Ce composant peut être utilisé avec différentes étiquettes (`label`).

Flux de Données Unidirectionnel

- **Principe :**
  - Les données circulent dans une seule direction, du parent vers l'enfant.
  - Simplifie la compréhension du flux des données dans l'application.
- **Pourquoi ce modèle est-il efficace ?**
  - Évite les bugs causés par des modifications imprévisibles des données.
  - Facilite le débogage et le test.
- **Exemple :**

Exemple de Flux de Données

```
 function Parent() {
  const message = "Bonjour React";
  return <Enfant message={message} />;
}
function Enfant(props) {
  return <h1>{props.message}</h1>;
}
```

Note : Le composant `Enfant` reçoit les données de `Parent` via les props.

Écosystème Riche

- **Large compatibilité avec des bibliothèques tierces :**
  - `Redux` : Gestion avancée de l'état global.
  - `React Router` : Gestion de la navigation et des routes.
- **Avantages :**
  - Flexibilité dans le choix des outils pour chaque projet.
  - Support communautaire étendu avec des solutions éprouvées.

Exemple : Gestion des Routes avec **React Router** :

```
 import { BrowserRouter, Route } from "react-router-dom";
function App() {
  return (
    <BrowserRouter>
      <Route path="/" component={Accueil} />
      <Route path="/contact" component={Contact} />
    </BrowserRouter>   ); }
```

Résultat : Permet une navigation fluide entre les pages.

Facilité d'Apprentissage

- **Basé sur JavaScript :**
  - Utilise les concepts fondamentaux de JavaScript (ES6+).
  - Facile à apprendre pour les développeurs connaissant déjà JavaScript.
- **Documentation complète et communauté :**
  - Site officiel avec des guides interactifs.
  - Nombreux tutoriels et forums disponibles en ligne.
- **Exemple :**

Composant de Base en React

```
 function Bonjour() {
  return <h1>Bienvenue à React</h1>;
}
export default Bonjour;
```

Résultat : Code intuitif et facile à comprendre.

Cas d'Utilisation de React.js

Applications Riches et Interactives

- **Applications Web Monopage (SPA)** :
  - React est idéal pour créer des applications qui chargent une seule page HTML et mettent à jour dynamiquement les données.
  - Exemples : Gmail, Trello, ou Google Maps.
- **Applications Interactives** :
  - Interfaces dynamiques avec mises à jour instantanées.
  - Exemples : Chats en direct, tableaux de bord interactifs.
- **Interfaces Utilisateur Complexes** :
  - Gestion efficace de grandes quantités de données et interactions utilisateur.
  - Exemples : Outils de gestion de projet comme Asana ou Jira.

Cas d'Utilisation de React.js

Portabilité et Écosystème Étendu

- **Applications Mobile avec React Native** :
  - Utilisez React pour développer des applications mobiles natives.
  - Code partagé entre le web et le mobile.
- **Prototypes et MVP (Minimum Viable Product)** :
  - Rapidité de développement grâce à la réutilisation des composants.
  - Idéal pour tester des idées de produits rapidement.
- **Portails et Applications Multiplateformes** :
  - Créez des applications accessibles via différents navigateurs ou plateformes.
  - Compatible avec des frameworks comme Electron pour des applications desktop.

Note : La flexibilité de React le rend adapté à de nombreux cas d'utilisation.

Sommaire

Qu'est-ce qu'un Composant ?

- Un composant est une **brique de construction** de React.
- Chaque composant est une fonction ou une classe JavaScript qui retourne du **JSX**.
- Un composant peut recevoir des **props** (paramètres) pour personnaliser son comportement.

Types de Composants :

- **Fonctionnel :** Écrit comme une fonction JavaScript.
- **Classique :** Écrit avec une classe (moins courant avec React moderne).

Exemple : Composant Fonctionnel

Code :

Composant de Bouton

```
 function Button({ label }) {
   return <button>{label}</button>;
}
function App() {
   return (
     <div>
       <Button label="Envoyer" />
       <Button label="Annuler" />
     </div>
   );
}
```

Résultat :

- Affiche deux boutons : "Envoyer" et "Annuler".
- Chaque bouton est une instance du même composant.

Sommaire

Virtual DOM

- **Qu'est-ce que le Virtual DOM ?**
  - Une copie en mémoire du DOM réel.
  - React utilise cette copie pour minimiser les manipulations du DOM réel.
- **Comment ça fonctionne ?**
  - React compare la version actuelle et la version précédente du Virtual DOM.
  - Applique uniquement les changements nécessaires au DOM réel (*diffing algorithm*).
- **Avantages :**
  - Améliore les performances des mises à jour.
  - Réduit le nombre de manipulations coûteuses du DOM.

Illustration

```
 const element = <h1>Hello, world!</h1>;
ReactDOM.render(element, document.getElementById('root'));
```

Note : React ne met à jour que ce qui a changé.

Exemple : Mise à jour avec le Virtual DOM

Code :

Mise à Jour

```
function Counter() {
  const [count, setCount] = React.useState(0);
  return (
    <div>
      <p>Compteur : {count}</p>
      <button onClick={() => setCount(count +
1)}>Incrémenter</button>
    </div>
  );
}
```

Résultat :

- Le DOM réel est mis à jour uniquement pour la valeur du compteur.

Sommaire

Étapes pour créer un projet React

1. **Installer Node.js et npm :**
   - Téléchargez et installez depuis `https://nodejs.org/`.
   - Vérifiez l'installation avec : `node -v` et `npm -v`.
2. **Créer un projet avec Create React App :**

Commande

```
npx create-react-app mon-projet //ou bien :
npx create-react-app mon-projet --template typescript
```

3. **Démarrer le serveur de développement :**

Commande

```
cd mon-projet && npm start
```

Structure d'un Projet React

- `/src/` : Contient les fichiers source.
- `index.js` : Point d'entrée principal.
- `App.js` : Composant racine.

Exemple : Point d'Entrée (`index.js`)

Code

```
 import React from "react";
import ReactDOM from "react-dom";
import App from "./App";
ReactDOM.render(<App />, document.getElementById("root"));
```

Composant racine App.js

Un Exemple Simple

- Voici un exemple d'application React affichant un message de bienvenue.
- La fonction App représente un composant React qui retourne du JSX.

Exemple : App.js

```
 import React from "react";
function App() {
   const message = "Bienvenue dans React!";
   return (
     <div>
       <h1>{message}</h1>
       <p>Ceci est une application simple en React.</p>
     </div>
   );
}
export default App;
```

Sommaire

Introduction aux Routes et à la Navigation

- Les routes permettent de naviguer entre différentes pages ou composants dans une application React.
- React utilise `react-router-dom` pour gérer les routes.
- Avantages :
  - Navigation dynamique sans rechargement de la page.
  - Gestion simplifiée des URL.
  - Support pour des routes imbriquées et des paramètres dynamiques.

Commande

```
npm install react-router-dom
```

Exemple d'Application : Structure des Routes

- Pages de l'application :
  - Page d'accueil (`/`).
  - Page contact (`/contact`).
  - Page services (`/services`).
- Structure des routes dans `App.js` :

Exemple : Routes

```
 import React from "react";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Home from "./Home"; import Contact from "./Contact";
import Services from "./Services";
function App() { return ( <BrowserRouter> <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/contact" element={<Contact />} />
        <Route path="/services" element={<Services />} />
      </Routes> </BrowserRouter> );
}
```

Création des Composants de Pages

- Exemple de composant pour la page d'accueil (`Home.js`) :

Code : Home.js

```
 import React from "react";
function Home() {
  return <h1>Bienvenue sur la Page d'accueil</h1>;
}
export default Home;
```

- Répétez pour `Contact.js` et `Services.js` :
  - Modifiez simplement le contenu de chaque page (Composant).

Ajout de Navigation entre les Pages

- Utilisez le composant `Link` pour naviguer entre les pages.
- Exemple de menu de navigation :

Code : Menu de Navigation

```
 import React from "react";
import { Link } from "react-router-dom";
function Navbar() {
  return (
    <nav>
      <ul>
        <li><Link to="/">Accueil</Link></li>
        <li><Link to="/contact">Contact</Link></li>
        <li><Link to="/services">Services</Link></li>
      </ul>
    </nav>
  );} export default Navbar;
```

Intégration de la Navigation dans l'Application

- Ajoutez le composant `Navbar` dans `App.js` :

Exemple : App.js avec Navbar

```
 import React from "react";
import { BrowserRouter, Routes, Route } from "react-router-dom";
function App() {
  return (
    <BrowserRouter>
      <Navbar />
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/contact" element={<Contact />} />
        <Route path="/services" element={<Services />} />
      </Routes>
    </BrowserRouter>
  ); }
```

Gestion des Routes Inexistantes (404)

- Ajoutez une route pour les pages non trouvées.

Exemple : Route 404

```
<Route path="*" element={<h1>Page non trouvée (404)</h1>} />
```

- Placez cette route à la fin des définitions dans `App.js`.

Composant <Outlet> dans React Router

- **Définition** :
  - <Outlet> est un composant fourni par React Router.
  - Sert de point de montage pour afficher les composants des routes enfants.
- **Quand utiliser** <Outlet> **?**
  - Lorsque vous définissez des **routes imbriquées**.
  - Pour créer une mise en page partagée, comme dans un **Dashboard**.
- **Avantages** :
  - Organisation claire des composants dans une application complexe.
  - Réduction de la duplication des éléments partagés (menus, en-têtes, etc.).

Exemple : Utilisation de <Outlet> dans un Dashboard

Code du composant `Dashboard.tsx` :

Exemple

```
 import { Link, Outlet } from "react-router-dom";
function Dashboard() {
  return (
    <div> <h1>Dashboard</h1> <nav>
        <Link to="users">Liste des utilisateurs</Link> |
        <Link to="add-user">Ajouter un utilisateur</Link>
</nav> <hr />
      <Outlet />  // Affiche le contenu des routes enfants
    </div> ); }
```

Cas d'utilisation :

- Créer des sections d'application avec des routes imbriquées (e.g., un Dashboard avec un menu latéral).
- Afficher dynamiquement du contenu en fonction de la route active.

useNavigate() : Navigation Programmatique

- useNavigate() est un hook fourni par **React Router** pour naviguer entre les pages de manière programmatique.
- Permet de rediriger l'utilisateur en réponse à des événements.

Utilisation :

Exemple : Redirection après une Action

```
 import React from "react";
import { useNavigate } from "react-router-dom";
function Login() {
  const navigate = useNavigate();
  const handleLogin = () => { // Après une connexion réussie
   navigate("/dashboard"); };
 return (<button onClick={handleLogin}>Connecter</button>);}
```

Avantages :

- **Flexibilité** : Navigation conditionnelle basée sur l'état de l'application.
- **Expérience utilisateur fluide** : Évite le rechargement complet de la page.

Sommaire

Architecture du Projet Combiné

- **Structure du projet** :
  - `/server` : Contient le Back-End (Express).
  - `/client` : Contient le Front-End (React).
- **Communication** : Le Back-End expose des APIs REST, consommées par le Front-End.
- **Avantages** :
  - Séparation claire des responsabilités.
  - Facilite la maintenance et l'évolutivité.

Configuration du Back-End avec Express

Fichier /server/src/index.ts

```
 import express from "express";
import cors from "cors";
const app = express();
app.use(cors());
app.use(express.json());
const PORT = 3001;
// Route API pour récupérer des données
app.get("/api/messages", (req, res) => {
   res.json({ message: "Bienvenue dans le Back-End Express !" });
});
app.listen(PORT, () => console.log(`Serveur Back-End sur
http://localhost:${PORT}`));
```

Commandes nécessaires :

- `npm install express cors`
- `npm install --save-dev @types/express @types/cors`

Configuration du Front-End avec React

Fichier /client/src/App.tsx

```
 import React, { useState, useEffect } from "react";
function App() {
   const [message, setMessage] = useState("");
   useEffect(() => {
     fetch("http://localhost:3001/api/messages")
      .then((res) => res.json()).then((data) =>
setMessage(data.message));
   }, []);
   return ( <div>
       <h1>Front-End React</h1>
       <p>{message}</p>
     </div> ); }
export default App;
```

Commandes nécessaires :

- `npx create-react-app client --template typescript`
- Ajouter les dépendances nécessaires : `npm install --save cors`

Exécution et Test du Projet

- **Étapes pour exécuter le projet** :
  1. Démarrer le Back-End :

Commandes

```
cd server
npm run dev
```

  2. Démarrer le Front-End :

Commandes

```
cd client
npm start
```

- **Vérification** :
  - Accédez à `http://localhost:3000`.
  - Assurez-vous que le message du Back-End s'affiche correctement dans le Front-End.

Sommaire

Qu'est-ce que Socket.IO ?

- **Socket.IO** est une bibliothèque permettant une communication bidirectionnelle en temps réel entre client et serveur.
- Basée sur les **WebSockets**, elle offre des mécanismes supplémentaires :
  - Repli sur le long-polling si les WebSockets ne sont pas disponibles.
  - Gestion automatique des reconnexions.
  - Événements personnalisés pour faciliter les échanges.
- Compatible avec Node.js pour le serveur et des frameworks front-end comme React.

Cas d'utilisation de Socket.IO

- Applications nécessitant des **mises à jour en temps réel** :
  - Chats en ligne.
  - Jeux multijoueurs.
  - Notifications en temps réel.
- Suivi en direct :
  - Suivi de positions géographiques.
  - Moniteurs de performance ou d'activité.
- Collaboration en temps réel :
  - Outils de collaboration (Google Docs).
  - Partage d'écran.

Configuration du projet

Commandes

```
npm install express socket.io
npm install --save-dev @types/socket.io
```

Créer un serveur Node.js avec Socket.IO :

```
 import express from "express";
import  Server  from "socket.io";
import http from "http";
const app = express(); const server = http.createServer(app);
const io = new Server(server);
io.on("connection", (socket) => {
   console.log("Un utilisateur connecté");
   socket.on("message", (msg) => {
     console.log("Message reçu :", msg);
     socket.broadcast.emit("message", msg); }); });
server.listen(3000, () => console.log("Serveur démarré"));
```

Intégration de Socket.IO dans React

Commande

```
npm install socket.io-client
```

Exemple d'intégration dans un composant React :

```
 import React, { useState, useEffect } from "react";
import  io  from "socket.io-client";
const socket = io("http://localhost:3000");
function Chat() { const [messages, setMessages] = useState([]);
   const [input, setInput] = useState("");
   useEffect(() => { socket.on("message", (msg) => {
       setMessages((prev) => [...prev, msg]); }); }, []);
   const sendMessage = () => {
     socket.emit("message", input); };
return ( <div> <ul> {messages.map((msg, idx) => ( <li
key={idx}>{msg}</li> ))} </ul>
       <input value={input} onChange={(e) =>
setInput(e.target.value)} />
       <button onClick={sendMessage}>Envoyer</button> </div> ); }
```

Avantages de Socket.IO avec React

- **Temps réel** : Mise à jour instantanée des données côté client et serveur.
- **Facilité d'intégration** : Compatible avec les composants fonctionnels React.
- **Gestion robuste** :
  - Reconnexion automatique.
  - Compatibilité avec les anciens navigateurs grâce au fallback.
- Événements personnalisés pour une communication plus claire.

Sommaire

Gestion de l'État

- Utilisez `useState` pour gérer des états simples dans les composants.
- `useReducer` est adapté pour des états complexes ou logiques conditionnelles.
- Exemple :

Exemple : `useState`

```
 const [count, setCount] = useState(0);
return <button onClick={() => setCount(count +
1)}>Incrémenter</button>;
```

Hooks : useEffect

- Gère les effets secondaires comme les appels API ou les abonnements.
- Remplace les méthodes du cycle de vie comme componentDidMount.
- Exemple :

Exemple : Appel API avec useEffect

```
 useEffect(() => {
   fetch("/api/data").then(res => res.json());
}, []);
```

Context API

- Partagez des données globales sans passer par les props (*prop drilling*).
- Utile pour les thèmes, authentifications, etc.
- Exemple :

Exemple : Création d'un Contexte

```
 const UserContext = React.createContext();
const App = () => (
  <UserContext.Provider value="BELATAR">
    <Component />
  </UserContext.Provider>
);
```

Requêtes HTTP avec `fetch`

- Gère les appels API avec `fetch` ou `axios`.
- Exemple :

Exemple : Appel API avec `fetch`

```
const App = () => {
 const [data, setData] = useState([]);
 useEffect(() => {
   fetch("/api/data")
   .then(res => res.json())
   .then(setData);
 }, []);
};
```

Performance et Optimisation

- Utilisez `React.memo` pour éviter les rendus inutiles.
- `useCallback` et `useMemo` pour mémoriser les fonctions et valeurs.
- Exemple :

Exemple : `useMemo`

```
const expensiveCalculation = useMemo(() => calculate(a, b), [a, b]);
```

Tests avec React

- Utilisez Jest et React Testing Library pour tester vos composants.
- Exemple :

Exemple : Test d'un Composant

```
 test("affiche le message", () => {
  render(<App />);
  expect(screen.getByText("Bienvenue")).toBeInTheDocument();
});
```

- Typage des props et états pour plus de sécurité.
- Exemple :

Exemple : Typage des Props

```
 interface Props { message: string; }
const App: React.FC<Props> = ( message ) =>
<h1>message</h1>;
```

Cycle de Vie des Composants

- Les hooks comme `useEffect` remplacent les méthodes des classes.
- Exemple :

Exemple : `useEffect`

```
 useEffect(() => {
  console.log("Monté du composant");
}, []);
```

Structure des Dossiers

- Organisez vos fichiers selon le modèle Atomic Design.
- Exemple :

Structure Recommandée

```
src/
 components/
   atoms/
   molecules/
   organisms/
```

Style et CSS

- Utilisez CSS Modules ou Styled Components.
- Exemple :

Exemple : CSS Module

```
 import styles from "./App.module.css";
const App = () => <div
className={styles.container}>Bienvenue</div>;
```

Gestion Avancée de l'État

- Utilisez Redux ou Recoil pour des états complexes ou partagés.
- Exemple :

Exemple : Redux

```
 import  createStore  from "redux";
const reducer = (state = 0, action) => action.type ===
"INCR" ? state + 1 : state;
const store = createStore(reducer);
```

Ressources et Références

- **Documentation Officielle :**
  - React.js : https://reactjs.org/docs/getting-started.html
  - Express.js : https://expressjs.com/
  - TypeScript : https://www.typescriptlang.org/docs/
- **Tutoriaux et Guides :**
  - FreeCodeCamp : https://www.freecodecamp.org/
  - MDN Web Docs (Mozilla) : https://developer.mozilla.org/
  - Traversy Media (YouTube) : https://www.youtube.com/c/TraversyMedia
- **Livres Recommandés :**
  - *Learning React* - Alex Banks et Eve Porcello
  - *Pro TypeScript* - Steve Fenton
- **Communautés :**
  - Reactiflux (Discord) : https://www.reactiflux.com/

Conseil : Explorez, expérimentez et pratiquez régulièrement pour approfondir vos connaissances !