TP5 – Taskme : Migration vers React

Objectif général

Restructurer l'application Taskme (développée en JavaScript dans les TPs précédents) en une application React moderne, en utilisant les composants, hooks, etc

---

Partie 1 – Initialisation du projet React

1.1 Créer l'application React

```
npx create-react-app tp5_ts_react --template typescript
```

- **Téléchargement de Create React App** (outil de génération).
- **Création du dossier `tp5_ts_react/`** avec toute la structure d'un projet React adapté à TypeScript.
- **Installation automatique des dépendances** : React, TypeScript, etc
- **Initialisation d'un projet fonctionnel**, prêt à être lancé avec :

```
cd tp5_ts_react
```

```
npm start
```

1.2 Installer les dépendances nécessaires

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

- `react-router-dom` permet de créer une navigation entre plusieurs pages dans une application React **sans recharger la page**, en affichant chaque vue selon l'URL.

```
npm install --save-dev @types/react-router-dom
```

- permet d'**ajouter les types TypeScript** pour `react-router-dom`, afin que la navigation entre plusieurs pages dans React puisse bénéficier du **typage fort et de l'auto-complétion** dans l'éditeur.

1.3 Structure du projet

On vise à créer la structure de dossiers présentée dans le cours:

```

```

Pour ce faire, Dans le projet généré par défaut, vous pouvez garder les éléments de base suivants:

```
public/
├── index.html
src/
├── App.tsx
├── index.tsx
├── index.css
tsconfig.json
```

avec ce contenu minimaliste:

Index.html

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Taskme React TypeScript</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

app.tsx

```
import React from "react";
import './index.css';

const App: React.FC = () => {
  return (
    <div className="app">
      <h1>Bienvenue sur Taskme React TypeScript!</h1>
    </div>
  );
};

export default App;
```

index.tsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";

const root = ReactDOM.createRoot(
  document.getElementById("root") as HTMLElement
);
```

```
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

index.css

```
body {
  margin: 0;
  font-family: Arial, sans-serif;
}

.app {
  text-align: center;
  padding: 2rem;
}
```

Partie 2 – Création des Composants du layout

Sous le dossier `src`, créer le dossier `components` → `layout` puis les dossiers `Header`, `Footer`, `Sidebar`. Dans chacun de ces dossier on aura le fichier tsx et css correspondant (e.g Header.tsx et Header.css).

2.1 Créer les fichiers du Header

Header.tsx

```
import React from "react";
import "./Header.css";

const Header: React.FC = () => {
  return (
    <header className="header">
      <button id="open-sidebar" className="sidebar-toggle">
        ☰
      </button>

      <div className="header-title">
        <h1>Taskme</h1>
        <p>Gérez et affectez des tâches facilement.</p>
      </div>

      <button id="theme-toggle" className="theme-btn">
        Changer de thème
      </button>
```

```
    </header>
  );
};

export default Header;
```

Header.css

```
header {
  background-color: var(--primary-color);
  color: white;
  padding: var(--spacing);
  box-shadow: var(--shadow);
  text-align: center;
}

header h1 {
  font-size: var(--font-size-title);
  margin-bottom: 0.5rem;
}

header p {
  font-weight: 500;
}

nav ul {
  list-style: none;
  display: flex;
  justify-content: center;
  gap: 1rem;
  margin-top: 0.5rem;
}

nav a {
  color: white;
  text-decoration: none;
  font-weight: 500;
}

nav a[aria-current="page"] {
  font-weight: bold;
  text-decoration: underline;
  color: var(--accent-color);
}

nav a:hover {
  text-decoration: underline;
}
```

```css
/* Bouton menu dans le header */
.menu-btn {
  background: none;
  border: none;
  font-size: 1.8rem;
  cursor: pointer;
  margin-right: 1rem;
  color: var(--text-color, #333);
  transition: transform 0.2s;
}

.menu-btn:hover {
  transform: scale(1.1);
}

/* Mode Sombre/clair */
.theme-btn {
  position: absolute;
  top: 1rem;
  right: 1rem;
  background-color: rgba(255, 255, 255, 0.2);
  color: #ffffff;
  border: 1px solid white;
  padding: 0.5rem;
  cursor: pointer;
  font-size: 0.9rem;
  width: auto;
}

.theme-btn:hover {
  background-color: rgba(255, 255, 255, 0.3);
}

/* Dark mode pour le header */
body.dark-mode header {
  background-color: #0d47a1;
}

/* Responsive */
@media (max-width: 768px) {
  nav ul {
    flex-direction: column;
    gap: 0.25rem;
  }

  .menu-btn {
    font-size: 1.6rem;
```

```
  }
}
```

2.1 Créer les fichiers du Footer

Footer.tsx

```
import React from "react";
import './Footer.css';

const Footer: React.FC = ()=>{
    return (
        <footer className="footer">
      <p>Tous droits réservés © 2025 - Taskme</p>
      <p>
        <a href="https://example.com/mentions" target="_blank" rel="noopener noreferrer">
          Mentions légales
        </a>{" "}
        —{" "}
        <a href="https://example.com/privacy" target="_blank" rel="noopener noreferrer">
          Politique de confidentialité
        </a>{" "}
        —{" "}
        <a href="https://example.com/terms" target="_blank" rel="noopener noreferrer">
          Conditions d’utilisation
        </a>
      </p>
    </footer>
    );
}

export default Footer;
```

Footer.css

```
footer {
  background-color: var(--primary-color);
  color: white;
  text-align: center;
  padding: var(--spacing);
}

footer a {
  color: white;
  text-decoration: none;
  margin: 0 0.5rem;
}
```

```
footer a:hover {
  text-decoration: underline;
}

/* Dark mode pour le footer */
body.dark-mode footer {
  background-color: #0d47a1;
}
```

2.3 Créer les fichiers du Sidebar

Sidebar.tsx

```
import React from "react";
import './Sidebar.css'

const Sidebar: React.FC = () => {
    return (
        <aside id="sidebar" className="sidebar">
        <h2>Taskme</h2>
            <ul>
                <li><a href="/"> Accueil</a></li>
                <li><a href="/profile"> Profil</a></li>
                <li><a href="/tasks"> Tâches</a></li>
            </ul>
        </aside>
    );
};

export default Sidebar;
```

Sidebar.css

```
.sidebar {
  position: fixed;
  top: 0;
  left: -250px; /* cachée par défaut */
  width: 250px;
  height: 100vh;
  background-color: var(--primary-color, #007bff);
  color: #fff;
  display: flex;
  flex-direction: column;
  padding: 2rem 1rem;
  box-shadow: 2px 0 6px rgba(0, 0, 0, 0.15);
  transition: left 0.3s ease;
  z-index: 1000;
}
```

```
.sidebar.open {
  left: 0; /* affichée */
}

.sidebar h2 {
  font-size: 1.5rem;
  margin-bottom: 2rem;
  text-align: center;
  font-weight: bold;
}

.sidebar ul {
  list-style: none;
  padding: 0;
}

.sidebar li {
  margin-bottom: 1rem;
}

.sidebar a {
  display: block;
  padding: 0.75rem 1rem;
  border-radius: 8px;
  text-decoration: none;
  color: #fff;
  font-weight: 500;
  transition: background-color 0.2s;
}

.sidebar a:hover,
.sidebar a[aria-current="page"] {
  background-color: rgba(255, 255, 255, 0.2);
}

/* Contenu principal décalé quand la sidebar est ouverte */
.main-content {
  transition: margin-left 0.3s ease;
  padding: 2rem;
  margin-left: 0;
}

.sidebar.open + .main-content {
  margin-left: 250px;
}

.sidebar-toggle {
```

```
  position: fixed;
  top: 1rem;
  left: 1rem;
  z-index: 1100;
  background-color: var(--primary-color, #007bff);
  color: #fff;
  border: none;
  border-radius: 6px;
  padding: 0.6rem 0.9rem;
  font-size: 1.2rem;
  cursor: pointer;
  transition: background-color 0.3s ease;
  width: auto;
}

/* Mode sombre */
body.dark-mode .sidebar {
  background-color: #1f1f1f;
  color: #eee;
}

body.dark-mode .sidebar a {
  color: #eee;
}

body.dark-mode .sidebar a:hover {
  background-color: rgba(255, 255, 255, 0.15);
}

/* Responsive : sur mobile, la sidebar recouvre le contenu */
@media (max-width: 768px) {
  .sidebar {
    width: 220px;
  }

  .sidebar.open + .main-content {
    margin-left: 0;
  }
}
```

2.4 Créer le composant du bouton BackToTopBtn

Dans le dossier layout ⇒ BackToTopBtn, créer et compléter le composant BackToTopBtn.tsx ainsi que la feuille de style associée BackToTopBtn.css .

2.5 Importer les composants dans App

Importer et organiser les composants, ainsi créés, dans App.tsx dans le but d'avoir **une vue similaire** des anciennes versions de l'application.

N'oubliez pas de restaurer le style global: **index.css**

```
/* ============================================== */
/* Reset & Variables                              */
/* ============================================== */
* {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
}

:root {
  --primary-color: #2b8aed;
  --secondary-color: #f0f4ff;
  --accent-color: #ffb400;
  --success-color: #27ae60;
  --danger-color: #e74c3c;
  --text-color: #333;
  --bg-color: #ffffff;
  --border-color: #dddddd;

  --font-main: 'Segoe UI', Arial, sans-serif;
  --font-size-base: 16px;
  --font-size-title: 1.8rem;

  --radius: 8px;
  --spacing: 1rem;
  --shadow: 0 2px 8px rgba(0, 0, 0, 0.1);
}

body {
  background-color: var(--bg-color);
  color: var(--text-color);
  font-family: var(--font-main);
  font-size: var(--font-size-base);
  line-height: 1.6;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
}

/* ============================================== */
/* Main content                                   */
/* ============================================== */
main {
```

```
  flex: 1;
  display: flex;
  flex-direction: row;
  gap: var(--spacing);
  padding: var(--spacing);
}

/* Sections */
section {
  background-color: var(--secondary-color);
  padding: var(--spacing);
  border-radius: var(--radius);
  box-shadow: var(--shadow);
  border: 1px solid var(--border-color);
  flex: 2;
}

h2 {
  margin-bottom: 0.5rem;
  color: var(--primary-color);
}

/* Aside (formulaire création tâche) */
aside {
  background-color: var(--secondary-color);
  padding: var(--spacing);
  border-radius: var(--radius);
  box-shadow: var(--shadow);
  border: 1px solid var(--border-color);
  flex: 1;
  display: flex;
  flex-direction: column;
  gap: 0.5rem;
}

/* ============================================== */
/* Formulaires                                    */
/* ============================================== */
form label {
  display: block;
  margin-top: 0.5rem;
  font-weight: 500;
}

input, textarea, select {
  width: 100%;
  padding: 0.5rem;
  border: 1px solid var(--border-color);
```

```
  border-radius: var(--radius);
}

input:focus, textarea:focus, select:focus {
  outline: none;
  border-color: var(--primary-color);
}

button {
  background-color: var(--accent-color);
  color: white;
  border: none;
  border-radius: var(--radius);
  padding: 0.6rem 1rem;
  margin-top: 0.5rem;
  cursor: pointer;
  transition: 0.3s;
}

button:hover {
  background-color: #e69900;
}

button.success {
  background-color: var(--success-color);
}

button.danger {
  background-color: var(--danger-color);
}

/* ============================================= */
/* Tableaux (liste des tâches)                   */
/* ============================================= */
table {
  width: 100%;
  border-collapse: collapse;
  margin-top: var(--spacing);
}

thead {
  background-color: var(--primary-color);
  color: white;
}

th, td {
  padding: 0.5rem;
  border: 1px solid var(--border-color);
```

```
  text-align: left;
}

tr:nth-child(even) {
  background-color: var(--secondary-color);
}

/* ============================================== */
/* Listes                                         */
/* ============================================== */
ul, ol {
  margin-left: 1.5rem;
  margin-top: 0.5rem;
  margin-bottom: 0.5rem;
}

/* Figures */
figure {
  margin: var(--spacing) 0;
  text-align: center;
}

figcaption {
  font-size: 0.9rem;
  margin-top: 0.25rem;
  color: var(--text-color);
}

/* ============================================== */
/* Mode Sombre                                    */
/* ============================================== */
body.dark-mode {
  background-color: #1a1a1a;
  color: #e0e0e0;
}

body.dark-mode section,
body.dark-mode aside {
  background-color: #2a2a2a;
  color: #e0e0e0;
  border: 1px solid #444;
}

body.dark-mode input,
body.dark-mode textarea,
body.dark-mode select {
  background-color: #333;
  color: #e0e0e0;
```

```
  border-color: #555;
}

body.dark-mode table tr:nth-child(even) {
  background-color: #333;
}

body.dark-mode th,
body.dark-mode td {
  border-color: #555;
}

/* ============================================== */
/* Responsive                                     */
/* ============================================== */
@media (max-width: 1024px) {
  main {
    flex-direction: column;
  }

  aside, section {
    width: 100%;
  }
}

@media (max-width: 768px) {
  button {
    width: 100%;
  }

  figure img {
    width: 100%;
    height: auto;
  }
}
```

Partie 3 – Création des pages et Configuration du routage

3.1 Créer les composants des pages

Sous le dossier `src`, créer le dossier `pages` dans lequel vous créez les deux pages `HomePage.tsx` (pour la page d'accueil) et `ProfilePage.tsx`. Compléter le contenu de ces deux pages adéquatement. Créer aussi la page `TaskPage.tsx` qui, pour le moment, va contenir juste *un message de bienvenue*.

3.2 Routage des pages

- Utiliser `BrowserRouter, Routes, Route` pour router les pages. e.g `<Route path='' element={< HomePage />}/>`
- Tester le routage en naviguant entre les pages.

3.3 Page TaskPage

- Au sein du dossier `pages`, on créera un autre dossier `TaskPage` qui va contenir deux composants `TaskForm.tsx` et `TaskTable.tsx.`
- Modifier la page `TaskPage.tsx` pour afficher le formulaire d'ajout et le tableau d'affichage des tâches.

Partie 4 – Implémentation de la logique du thème et de sidebar

4.1 Logique du thème

Dans App :

- Déclarer un state **theme** avec valeur initiale **null**.
- Charger le thème sauvegardé depuis localStorage ***au montage*** et mettre à jour theme.
- Lorsque theme **change** :
    - ajouter la classe dark-mode au body si **theme** est "dark"
    - la retirer si **theme** est "light"
    - sauvegarder le thème dans localStorage
- Implémenter toggleTheme qui alterne entre "light" et "dark".
- Passer { theme, toggleTheme } au composant Header.

Dans Header :

- Définir les props :

```
interface AppProps {
        theme: "light" | "dark" | null;
        toggleTheme: () => void;
        }
```

- Recevoir les props dans les arguments du composant.
- Ajuster le bouton :
    - son texte dépend du thème (Mode Sombre / Mode Clair)
    - au clic, appeler toggleTheme

4.2 Logique du Sidebar

Dans App:

- Déclarer le state `open` avec un state initial false.
- Dans un `useEffect`, sélectionner le sidebar et y:
    - ajouter la classe `open` si `open` est `true`,
    - retirer la classe `open` si `open` est `false`.
- Implémenter `toggleSidebar()` qui inverse `open`.
- Implémenter `closeSidebar()` qui ferme le sidebar *en cliquant dans la zone principal main.*
- Passer `toggleSidebar` au Header.

Dans Header:

- Ajouter `toggleSidebar` au propriétés AppProps.
- Utiliser convenablement sur le bouton pour ouvrir/fermer le sidebar.

Partie 5 – Api de gestion des tâches

5.1 Installer JSON Server

JSON Server est un outil qui permet de créer **une API REST complète** en quelques secondes à partir d'un simple fichier **.json**. On va l'utiliser pour simuler le backend.

npm install -g json-server

5.2 Création de db.json

Sous le dossier du projet, créer un dossier `api` qui va contenir le fichier `db.json.`

Copier dedans l'exemple suivant:


```
{
  "tasks": [
    {
      "id": "bd21aa76-01fb-4b6f-9eae-4e0f8232f200",
      "taskName": "Livraison de colis",
      "description": "Livrer un petit colis à l'adresse indiquée dans le centre-ville.",
      "dueDate": "2025-12-10",
      "remuneration": "15",
      "vehicle": true,
      "mode": "manuel",
      "taskStatus": "ouverte"
    },
    {
      "id": "e1fbcf97-9519-4876-b123-1c30e292a621",
      "taskName": "Montage de meuble",
      "description": "Monter une petite étagère IKEA dans un appartement.",
      "dueDate": "2025-12-05",
      "remuneration": "25",
```

```
      "vehicle": false,
      "mode": "manuel",
      "taskStatus": "ouverte"
    },
    {
      "id": "78c4aec1-37f8-4dd2-bb9b-a3d3b844372a",
      "taskName": "Aide aux courses",
      "description": "Accompagner une personne âgée pour faire ses courses
hebdomadaires.",
      "dueDate": "2025-12-02",
      "remuneration": "18",
      "vehicle": true,
      "mode": "manuel",
      "taskStatus": "ouverte"
    }
  ]
}
```


5.3 Démarrage du serveur JSON Server

Il suffit d'exécuter la commande suivante pour avoir l'api http://localhost:4000/tasks prete pour utilisation.

```
json-server --watch api/db.json --port 4000
```

Partie 6 – CRUD de la gestion des tâches

6.1 Affichage des tâches

- Dans le composant TaskPage.tsx, créer la fonction asynchrone getTasks permettant :
    - de récupérer toutes les tâches depuis l'API JSON Server,
    - d'attendre la réponse avec await,
    - de convertir la réponse en JSON,
    - d'enregistrer la liste obtenue dans le state tasks,
    - et de gérer les erreurs avec try...catch.

Indice: utiliser async**,** fetch, await

- Ajouter un useEffect permettant d'appeler automatiquement la fonction getTasks au moment du chargement initial du composant.
- Ajouter la propriété tasks={tasks} à TaskTable. Dans ce dernier, recevoir la propriété tasks et utiliser la pour afficher la liste des tâches.

Indice: Pour parcourir le tableau tasks utiliser { tasks.map(task => (<tr key={task.id}> <td>{task.taskName}</td> .. </tr>)) }

6.2 Suppression d'une tâche

- Créer la fonction handleDeleteTask(task: Task) permettant de supprimer la tâche passée en paramètre de la liste des tâches stockée dans le state tasks. Utiliser filter pour retourner tous les tasks à part celui passé en paramètre.
- Passer onDelete={handleDeleteTask} au composant TaskTable
- Tester la suppression d'un élément du tableau des tâches en appliquant onClick={() => onDelete(task)} sur le bouton de suppression.
- Pour s'assurer de la suppression permanente, créer une fonction asynchrone deleteTask permettant de supprimer la tâche passée en paramètre et qui:
    - affiche le message de confirmation en utilisant window.confirm.
    - utilise fetch(... ,{ method: "DELETE" }) pour la suppression.
    - Si la reponse est OK, appelle handleDeleteTask pour la mise à jour du state.
- Passer onDelete={deleteTask} au composant TaskTable au lieu de handleDeleteTask

6.3 Restauration d'une tâche pour édition

- Créer la fonction handleEditClick(task: Task) permettant de mettre à jour le state editingTask par la tâche à éditer ; task.
- Passer onEdit={handleEditClick} au composant TaskTable.
- Tester la restauration d'une tâche en cliquant sur le bouton de l'édition onClick={() => onEdit(task)}
- Passer également editingTask={editingTask} et setEditingTask={setEditingTask} au composant TaskForm.
- Dans TaskForm:
    - Créer les ***states*** du formulaire comme suit.

```
const [taskName, setTaskName] = useState("");
const [description, setDescription] = useState("");
const [dueDate, setDueDate] = useState("");
const [remuneration, setRemuneration] = useState<number | "">("");
const [vehicle, setVehicle] = useState(false);
const [mode, setMode] = useState("manuel");
const [taskStatus, setTaskStatus] = useState("ouverte");
```

    - Dans un useEffect, et selon la propriété editingTask, mettre à jour convenablement les ***states*** du formulaire.
    - Ajouter l'attribut value et l'écouteur onChange convenablement au différents champs du formulaire.
    - Ajuster le titre du formulaire "Créer une tâche" ou "Editer une tâche" dépendemment de editingTask. De même pour le texte du bouton "Ajouter la tâche" ou "Editer la tâche".

6.4 Edition et création des tâches

- Attacher l'événement onSubmit={handleSubmit} au formulaire.

- Créer la fonction asynchrone `handleSubmit` permettant d'ajouter ou de mettre à jour une tâche.
- Tester la création d'une tâche.

Indice:

```
{ method: "PUT" pour update et "POST" pour ajout,

  headers: {"Content-Type":"application/json"},

  body: JSON.stringify(le task à envoyer)

 }
```