ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

Compte Rendu Développement Frontend

TP 1 : Le Web Moderne & React

Réalisé par : DADA Anass

4DDSIG5

Partie 1 — Initialisation

- **Objectif** : Créer le projet React + TypeScript avec Vite et préparer l'environnement.
- **Actions** : initialisation du template Vite, installation des dépendances (npm install), configuration TypeScript (tsconfig.*) et ESLint optionnel.
- **Fichiers clés** : package.json, tsconfig.app.json, index.html, main.tsx.
- **Résultat attendu** : projet démarrable avec npm run dev.

Partie 2 — Backend avec json-server

- **Objectif :** Fournir une fausse API REST pour les données (GET, POST, PUT, DELETE).
- **Actions :**
  - Installer ou utiliser json-server.
  - Créer db.json à la racine (structure : projects, columns).
  - json-server --watch db.json --port 4000
- **Endpoints :**
  - GET http://localhost:4000/projects → liste des projets (JSON)
  - GET http://localhost:4000/columns → colonnes Kanban (JSON)
- **Résultat observé :** API REST locale qui sert les données dynamiquement.

Quelle différence entre des données en dur dans le code et une API REST ?
Réponse :

- **Données en dur :** intégrées directement dans l'application (fichiers ou constantes).
- **API REST :** données servies par un serveur via des endpoints HTTP (GET/POST/PUT/DELETE). Avantages : partage centralisé, persistance, modification dynamique, multi-clients, contrôles d'accès.

Partie 3 — Composants avec Props

- **Objectif :** Créer composants réutilisables et styles via CSS Modules.
- **Composants demandés :**
  - Header (Header.tsx) — props : title, onMenuClick. Style : Header.module.css.
  - Sidebar (src/components/Sidebar.tsx) — props : projects, isOpen. Style : Sidebar.module.css.
  - MainContent (src/components/MainContent.tsx) — props : columns. Style : MainContent.module.css.
- **Concepts expliqués :**
  - className en JSX (car class est mot réservé).
  - key obligatoire dans .map() pour l'identification des éléments (éviter utiliser l'index si l'élément a un id stable).
- **Résultat** : layout statique du Kanban (Header + Sidebar + colonnes de tâches).

Partie 4 — State, useEffect & Fetch

- **Objectif :** Remplacer données en dur par des données chargées depuis l'API (json-server).
- **Implémentation faite :** src/App.tsx utilise useState, useEffect et fetch pour charger projects et columns depuis http://localhost:4000.
- **Comportement :**
  - useEffect(..., []) → fetch exécuté au montage (une seule fois).
  - Affichage d'un écran Chargement... pendant la requête.
  - console.log dans le useEffect pour observer les données reçues.

Questions du TP :
Q1 : Combien de fois le useEffect s'exécute-t-il ? Pourquoi ?
Q2 : Arrêtez json-server (Ctrl+C) et rechargez. Que se passe-t-il ?
Q3 : Ouvrez Network (F12). Voyez-vous les requêtes vers localhost:4000 ? Code HTTP ?
Q4 : Les nouvelles données s'affichent ? Décrivez le cycle complet.
Q5 : Dessinez le flux : json-server → fetch → useState → useEffect → composants → props.

Réponses :

Q1 : useEffect s'exécute 1 fois ici (dépendances []) → fetch au montage.
Q2 : Si on arrête json-server et recharge → fetch échoue, erreurs réseau visibles en console.
Q3 : Dans l'onglet Network (F12) on voit GET vers /projects et /columns (code 200 si OK).
Q4: Modifier db.json, recharger React → les nouvelles données sont servies par json-server ; React charge à nouveau (après reload) et met à jour l'UI via setState.
Q5: Flux : json-server → fetch → setState → re-render → props → composant