

TP – Frontend React basique pour consommer l'API MVC

Objectifs

- Initialiser un **frontend React** moderne (Vite).
- Appeler l'API Express (GET/POST/PUT/DELETE) et **afficher / créer / modifier / supprimer** des données.
- Structurer un mini-frontend propre : **pages, composants, client API, routing**.
- Gérer **loading, errors, formulaires**.
- En autonomie : répliquer la logique pour **Products** et bonus UX.

Prérequis

- Node 18+ / npm
- API locale fonctionnelle (ex : `http://localhost:3000`)
- Connaissances de base JS/React

PARTIE GUIDÉE (users) — 45 à 60 min

1) Création du projet React (Vite)

```
npm create vite@latest frontend-react-api -- --template react
cd frontend-react-api
npm install
npm install react-router-dom
```

Démarrer

```
npm run dev
```

2) Config d'environnement

Crée un fichier `.env` à la racine :

```
VITE_API_URL=http://localhost:3000
```

3) Client API minimal

Crée `src/api/client.js` :

```
const BASE = import.meta.env.VITE_API_URL;

async function request(path, options = {}) {
  const res = await fetch(`${BASE}${path}`, {
    headers: { "Content-Type": "application/json", ...(options.headers || {}) },
    ...options,
  });
  if (!res.ok) {
    const text = await res.text().catch(() => "");
    throw new Error(`HTTP ${res.status} - ${text} || res.statusText`);
  }
  // 204 No Content
  if (res.status === 204) return null;
  return res.json();
}

export const api = {
  get: (p) => request(p),
  post: (p, body) => request(p, { method: "POST", body: JSON.stringify(body) }),
  put: (p, body) => request(p, { method: "PUT", body: JSON.stringify(body) }),
  del: (p) => request(p, { method: "DELETE" }),
};
```

4) Routing + Shell de l'app

Édite `src/main.jsx` :

```
import React from "react";
import ReactDOM from "react-dom/client";
import { BrowserRouter, RouterProvider } from "react-router-dom";
import App from "./App.jsx";
import UsersPage from "./pages/UsersPage.jsx";

const router = createBrowserRouter([
  {
    path: "/",
    element: <App />,
    children: [
      { index: true, element: <UsersPage /> },
      // (autonomie) { path: "products", element: <ProductsPage /> }
    ],
  },
]);

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <RouterProvider router={router} />
  </React.StrictMode>
);
```

```
    </React.StrictMode>
  );
};
```

Crée `src/App.jsx` :

```
import { Outlet, NavLink } from "react-router-dom";

export default function App() {
  return (
    <div style={{ maxWidth: 900, margin: "0 auto", padding: 24 }}>
      <h1>Frontend React ↔ API</h1>
      <nav style={{ display: "flex", gap: 12, marginBottom: 16 }}>
        <NavLink to="/">Users</NavLink>
        <NavLink to="/products">Products</NavLink>
      </nav>
      <Outlet />
    </div>
  );
}
```

5) Page Users — liste + création + suppression

Crée `src/pages/UsersPage.jsx` :

```
import { useEffect, useState } from "react";
import { api } from "../api/client";

export default function UsersPage() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);
  const [err, setErr] = useState("");

  // form
  const [name, setName] = useState("");
  const [age, setAge] = useState("");

  async function load() {
    try {
      setLoading(true);
      setErr("");
      const res = await api.get("/api/users");
      setUsers(res.data ?? res); // selon ton contrôleur (liste ou {total,data})
    } catch (e) {
      setErr(e.message);
    } finally {
      setLoading(false);
    }
  }
}
```

```

useEffect(() => { load(); }, []);

async function createUser(e) {
  e.preventDefault();
  try {
    const created = await api.post("/api/users", { name, age: Number(age) });
    setUsers((u) => [ ...(Array.isArray(u) ? u : u.data), created ]);
    setName(""); setAge("");
  } catch (e) {
    alert(e.message);
  }
}

async function deleteUser(id) {
  if (!confirm("Supprimer cet utilisateur ?")) return;
  const prev = users;
  // Optimistic UI
  setUsers(users.filter((u) => u.id !== id));
  try {
    await api.del(`/api/users/${id}`);
  } catch (e) {
    alert("Suppression échouée, retour état précédent");
    setUsers(prev);
  }
}

if (loading) return <p>Chargement...</p>;
if (err) return <p style={{ color: "crimson" }}>Erreur: {err}</p>;

const list = Array.isArray(users) ? users : users.data;

return (
  <div>
    <h2>Users</h2>

    <form onSubmit={createUser} style={{ display: "flex", gap: 8, marginBottom: 16 }}>
      <input placeholder="name" value={name} onChange={(e) => setName(e.target.value)}
required />
      <input placeholder="age" type="number" value={age} onChange={(e) =>
setAge(e.target.value)} required />
      <button>Ajouter</button>
    </form>

    <table width="100%" cellPadding="8" style={{ borderCollapse: "collapse" }}>
      <thead>
        <tr>
          <th align="left">ID</th>
          <th align="left">Name</th>
          <th align="left">Age</th>
          <th align="left">Actions</th>
        </tr>
      </thead>

```

```

<tbody>
  {list.map((u) => (
    <tr key={u.id} style={{ borderTop: "1px solid #ddd" }}>
      <td>{u.id}</td>
      <td>{u.name}</td>
      <td>{u.age}</td>
      <td>
        { /* (autonomie) bouton Éditer */ }
        <button onClick={() => deleteUser(u.id)}>Supprimer</button>
      </td>
    </tr>
  ))}
  {list.length === 0 && (
    <tr><td colspan="4">Aucun utilisateur pour le moment.</td></tr>
  )}
</tbody>
</table>
</div>
);
}

```

💡 Test :

- Lancer l'API (`npm run dev` côté back)
- Lancer le front (`npm run dev` côté front)
- Ajouter / supprimer des users depuis l'UI et vérifier côté API (GET).

6) (Option guidée courte) — Édition (PUT)

Ajoute un petit **mode édition inline** :

- un bouton **Éditer** qui remplace les cellules par des inputs,
- un bouton **Sauver** qui fait `PUT /api/users/:id`.

Ébauche d'état :

```

const [editingId, setEditingId] = useState(null);
const [editName, setEditName] = useState("");
const [editAge, setEditAge] = useState("");

function startEdit(u) { setEditingId(u.id); setEditName(u.name); setEditAge(u.age); }

async function saveEdit(id) {
  const updated = await api.put(`/api/users/${id}`, { name: editName, age: Number(editAge) });
};

setUsers(list.map(u => (u.id === id ? updated : u)));
setEditingId(null);
}

```

PARTIE EN AUTONOMIE (produits)

Mission

Créer la page **Products** avec la même logique : **liste + création + suppression + édition**, et bonus **recherche / filtres / pagination**.

1) Routing

- Créer `src/pages/ProductsPage.jsx`
- Déclarer la route `{ path: "products", element: <ProductsPage /> }` dans `main.jsx`.

2) API

- Endpoints disponibles (API côté back) :
 - `GET /api/products` (bonus : `?q=&minPrice=&maxPrice=&limit=&offset=`)
 - `GET /api/products/:id`
 - `POST /api/products { name, price }`
 - `PUT /api/products/:id`
 - `DELETE /api/products/:id`

3) Exigences UI

- Tableau des produits : **id, name, price**.
- Formulaire d'ajout en haut.
- Bouton **Supprimer** par ligne.
- **Édition inline** (ou modal simple).
- États **loading / error**.

4) Bonus UX

- Barre de **recherche texte** (`q`) filtrant côté API (ou côté front).
- Filtres **minPrice / maxPrice** (au submit, recharge la liste).
- **Pagination** naïve (`limit`, `offset`) si ton back l'expose.
- **Toast** minimal (alert/console) en cas d'erreur.
- Extraction d'un **hook** `useApiList` réutilisable (loading/err/list/reload).

Scénarios de test

1. **Créer** un produit puis **recharger** la liste.
2. **Supprimer** un produit : vérifier l'**optimistic UI** + rollback en cas d'échec.
3. **Éditer** un produit : le prix doit être un nombre ≥ 0 .

4. **Recherche** `q=cla` → “Clavier” apparaît.
5. **Filtres** min/max → seules les lignes valides restent.

Critères de réussite

Critère	OK
Routing opérationnel (Users ↔ Products)	<input type="checkbox"/>
Users : liste + create + delete + (edit)	<input type="checkbox"/>
Products : liste + create + delete + edit	<input type="checkbox"/>
Erreurs et chargements gérés proprement	<input type="checkbox"/>
Code clair (client API séparé, composants découpés)	<input type="checkbox"/>
Bonus : filtres / recherche / pagination	<input type="checkbox"/>

Structure finale attendue (exemple)

```
frontend-react-api/  
  .env  
  index.html  
  src/  
    api/client.js  
    App.jsx  
    main.jsx  
    pages/  
      UsersPage.jsx  
      ProductsPage.jsx    (autonomie)  
    components/  
      (facultatif : Table, Modal, Form, Toast...)
```

Pour aller plus loin (à la maison)

- Installer **Tailwind** ou **MUI** pour styliser rapidement.
- Créer un **contexte** “ApiProvider” pour partager `BASE_URL`.
- Extraire un **hook** `useCrud(resource)` générique.
- Ajouter une **authentification** (quand l’API proposera `/login` + JWT).
- Écrire des **tests React** (Vitest + Testing Library) pour la page Users.