

# Atelier 04 Weather : Dashboard Météo avec API

**Objectif:** Apprendre à consommer des API, gérer les effets secondaires et organiser une application React avec des composants réutilisables.

## Prérequis

- Node.js installé sur votre machine.
- Docker et Docker Compose installés.
- Clé API **OpenWeatherMap** (<https://openweathermap.org/api>), vous pouvez utiliser ce code : **96cb0c26b7b823e0ab8be44d133eecc4**

## Étape 1 : Préparer l'environnement React

### Objectif pédagogique :

Savoir créer un projet React fonctionnel et structurer les fichiers.

### Instructions :

1. Dans le dossier Ressources de cet atelier, copier le dossier **WeatherApp** dans le dossier **C:\ReactProjects**
2. Ouvrir le dossier **C:\ReactProjects\WeatherApp** avec votre éditeur de code
3. Démarrer docker et vérifier que les containers ne sont pas en exécution
4. Avec l'invite de commande, entrez dans le dossier **C:\ReactProjects\WeatherApp>** et lancez la commande suivante :  
**docker-compose up -d --build**
5. Créez un projet React avec Vite ou Create React App.
  1. Accéder au **WeatherApp\_container** avec la commande suivante :

1. **docker exec -it WeatherApp\_container sh**
2. **npm create vite@latest . --template**

Deux traits avant  
template

```
/usr/src/app # npm create vite@latest . --template
> npx
> create-vite .

  Select a framework:
    React
  Select a variant:
    JavaScript
  Use rollup-vite (Experimental)?:
  No
  Install with npm and start now?
    Yes / o No
```

3. Si le serveur Vite lancé, arrête-le (CTR + C)

4. Quitter le container

```
/usr/src/app # exit
```

5. Modifier le fichier vite.config.js

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [react()],
  server: {
    host: true,           // équivaut à --host
    watch: {
      usePolling: true, // <== active le mode "polling"
      interval: 1000,   // <== vérifie les changements toutes les 1s
    },
  },
})
```

6. Puis arrêter et démarrer le container, directement avec le docker ou bien exécuter ces deux commandes

```
docker-compose down
docker-compose up -d --build
```

7. Entre dans le container

```
docker exec -it WeatherApp_container sh
```

8. Lancer cette commande :

```
npm run dev -- --host
```

9. Lester l'application sur l'url <http://localhost:5173/>

## Étape 2 : Structurer les dossiers et composants

1. Créer les dossiers suivants :

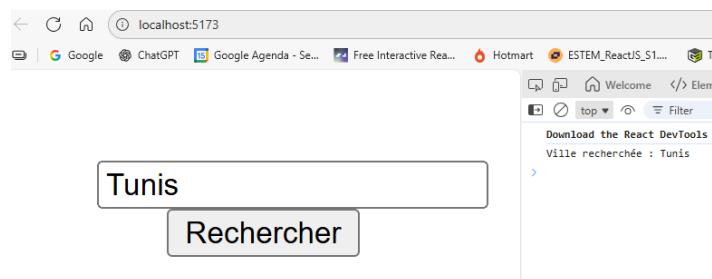
```
src/components
src/services
src/styles
```

2. Créer les composants :

- FormulaireVille.jsx : formulaire pour saisir une ville.
- CardMeteo.jsx : carte pour afficher les infos météo.
- HistoriqueRecherche.jsx : liste des villes recherchées.

### Étape 3 : Créer le formulaire de recherche de ville

1. Créer un composant React nommé FormulaireVille, exporté par défaut, et lui faire recevoir la fonction onSearch via les props.
2. Déclarer un état local nommé ville, avec sa fonction de mise à jour setVille, initialisé à une chaîne vide.
3. Crée une fonction interne nommée handleSubmit qui empêche le rechargement de la page, vérifie si ville n'est pas vide après un trim, et selon le cas :
  - a. appeler **onSearch(ville)** puis réinitialiser ville,
  - b. ou afficher un message demandant de saisir une ville.
4. Ajouter un formulaire HTML utilisant **handleSubmit** lors de la soumission.
5. Ajouter un champ de saisie dont la valeur est liée à ville, et dont les modifications appellent **setVille** via un **onChange** (n'est pas encore définie).
6. Ajouter un bouton de type **submit** permettant de lancer la recherche.
7. Vérifier que la saisie met bien à jour ville, que la soumission appelle **onSearch**, que ville se réinitialise, et qu'un message apparaît si le champ est vide.



```
//src/components/FormulaireVille.jsx
import React, { useState } from "react";

export default function FormulaireVille({ onSearch }) {
  const [ville, setVille] = useState("");

  const handleSubmit = (e) => {
    e.preventDefault();
    if (ville.trim() !== "") {
      //onSearch(ville);
      setVille("");
      console.log("Ville recherchée :", ville);
    } else {
      alert("Veuillez saisir une ville !");
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        placeholder="Entrez une ville"
        value={ville}
        onChange={(e) => setVille(e.target.value)}
      />
      <button type="submit">Rechercher</button>
    </form>
  );
}
```

## Étape 4 : Consommer l'API météo

- On va utiliser le site gratuit : <https://openweathermap.org/api>
- Exemple d'appel API :
   
`https://api.openweathermap.org/data/2.5/weather?lat=44.34&lon=10.99&appid={API key}`
- Clé API **OpenWeatherMap** (<https://openweathermap.org/api>), vous pouvez utiliser ce code : **96cb0c26b7b823e0ab8be44d133eecc4**

1. Créer un fichier de service API : src/services/api.js

```
const API_KEY = "VOTRE_CLE_API_OPENWEATHERMAP";

export const getWeather = async (ville) => {
  try {
    const response = await fetch(
      `https://api.openweathermap.org/data/2.5/weather?q=${ville}&units=metric&lang=fr&appid=${API_KEY}`
    );
    if (!response.ok) throw new Error("Ville non trouvée");
    return await response.json();
  } catch (error) {
    throw error;
  }
};
```

## Étape 5 : Assemblage dans App.jsx

```
//src/App.jsx
import FormulaireVille from "./components/FormulaireVille";
import { getWeather } from "./services/api";

function App() {
  const handleSearch = async (ville) => {
    try {
      const data = await getWeather(ville);

      console.log(data); // Afficher le retour de l'API

    } catch (error) {
      alert(error.message);
    }
  };

  return (
    <div className="App">
      <h1>Dashboard Météo</h1>
      <FormulaireVille onSearch={handleSearch} />
    </div>
  );
}

export default App;
```

```

        </div>
    );
}

export default App;

```

## Étape 6 : Modifier FormulaireVille.jsx

```

//src/components/FormulaireVille.jsx
import React, { useState } from "react";

export default function FormulaireVille({ onSearch }) {
  const [ville, setVille] = useState("");

  const handleSubmit = (e) => {
    e.preventDefault();
    if (ville.trim() !== "") {
      onSearch(ville); // Ajouter cet appel
      setVille("");
      console.log("Ville recherchée :", ville);
    } else {
      alert("Veuillez saisir une ville !");
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        placeholder="Entrez une ville"
        value={ville}
        onChange={(e) => setVille(e.target.value)}
      />
      <button type="submit">Rechercher</button>
    </form>
  );
}

```

## Étape 7 : Tester l'application

**Dashboard Météo**

Tunis Rechercher



## Étape 8 : Créer la carte météo réutilisable

```

//src/components/CardMeteo.jsx

export default function CardMeteo({ data }) {

```

```

if (!data) return null;

return (
  <div className="card-meteo">
    <h2>{data.name}</h2>
    <p>Température : {data.main.temp} °C</p>
    <p>Humidité : {data.main.humidity} %</p>
    <p>{data.weather[0].description}</p>
    <img
      src={`http://openweathermap.org/img/wn/${data.weather[0].icon}.png`}
      alt={data.weather[0].description}
    />
  </div>
);
}

```

## Étape 9 : Appel de composant CardMeteo dans App.jsx

```

//src/App.jsx
import CardMeteo from "./components/CardMeteo";
import FormulaireVille from "./components/FormulaireVille";
import { getWeather } from "./services/api";
import { useState } from "react";

function App() {

  const [meteo, setMeteo] = useState(null);

  const handleSearch = async (ville) => {
    try {
      const data = await getWeather(ville);
      setMeteo(data);
      console.log(data);
    } catch (error) {
      alert(error.message);
    }
  };

  return (
    <div className="App">
      <h1>Dashboard Météo</h1>
      <FormulaireVille onSearch={handleSearch} />
      <CardMeteo data={meteo}/>
    </div>
  );
}

export default App;

```

## Etape 10 : Tester l'application

```

Ville recherchée : Tunis
{
  coord: {...},
  weather: Array(1),
  base: 'stations',
  main: {...},
  visibility: 10000,
  ...
  base: "stations"
  ...
  ...
}
  
```

## Étape 11 : Ajouter l'historique des recherches

```
//src/components/HistoriqueRecherche.jsx

export default function HistoriqueRecherche({ villes }) {
  return (
    <ul>
      {villes.map((ville, index) => (
        <li key={index}>{ville}</li>
      )))
    </ul>
  );
}
```

## Étape 12 : Appel de l'historique dans App.jsx

```
//src/App.jsx
import CardMeteo from "./components/CardMeteo";
import FormulaireVille from "./components/FormulaireVille";
import { getWeather } from "./services/api";
import HistoriqueRecherche from "./components/HistoriqueRecherche";
import { useState } from "react";

function App() {

  const [meteo, setMeteo] = useState(null);
  const [historique, setHistorique] = useState([]);

  const handleSearch = async (ville) => {
    try {
      const data = await getWeather(ville);
      setMeteo(data);
      console.log(data);
      setHistorique([ville, ...historique]);
    } catch (error) {
      alert(error.message);
    }
  };

  return (
    <div className="App">
      <h1>Dashboard Météo</h1>
      <FormulaireVille onSearch={handleSearch} />
    </div>
  );
}

export default App;
```

```
<CardMeteo data={meteo} />
<h3>Historique :</h3>
<HistoriqueRecherche villes={historique} />
</div>
};

export default App;
```

## Étape 13 Tester l'application

### Dashboard Météo

#### Paris

Température : 11.02 °C

Humidité : 92 %

brouillard



#### Historique :

- Paris
- Tunis

## Étape 14 : Ajouter styles CSS (optionnel)

```
/* src/styles/App.css */
.App {
  text-align: center;
  font-family: Arial, sans-serif;
}

.card-meteo {
  border: 1px solid #ccc;
  padding: 1rem;
  display: inline-block;
  margin: 1rem;
  border-radius: 10px;
}
```

## Dashboard Météo

### Sfax

Température : 16.08 °C

Humidité : 55 %

peu nuageux



#### Historique :

- Sfax
- Paris
- Gabes
- tunis

### Étape 10 : Bonus / améliorations possibles

- Sauvegarder l'historique dans localStorage.
- Ajouter un filtre pour supprimer des villes de l'historique.
- Ajouter un loader pendant l'appel API (isLoading).
- Ajouter la météo par coordonnées GPS.