# Cahier des Charges - Application de Météo

## 1. Contexte et Objectif du Projet

Développer une application de météo qui permet aux utilisateurs de rechercher et d'afficher les conditions météorologiques actuelles et les prévisions pour différentes villes, le tout sur une interface unique.

## 2. Architecture de l'Application

### 2.1. Front-end

* **Technologies** :
  + **React** : Pour construire une interface utilisateur dynamique.
  + **Axios** : Pour les requêtes HTTP vers l'API de météo.
  + **React Router** : Pour la gestion de la navigation.
  + **CSS Modules ou Styled Components** : Pour le style et la mise en page.
  + **Context API ou Redux** : Pour la gestion de l'état (si nécessaire).

### 2.2. Back-end

* **Technologies** :
  + **Laravel** : Pour gérer la logique serveur et interagir avec l'API de météo.
  + **PHP** : Langage utilisé par Laravel.
  + **MySQL** : Pour le stockage des données utilisateur et de l'historique des recherches.

### 2.3. API de Météo

* **Services** : Utilisation d'OpenWeatherMap ou WeatherAPI pour récupérer les données météorologiques.

## 3. Structure de l'Application

### 3.1. Composants de l'Application

* **App.js** : Composant principal.
* **SearchBar.js** : Champ de recherche pour entrer le nom de la ville.
* **WeatherDisplay.js** : Affiche les conditions météorologiques actuelles.
* **ForecastDisplay.js** : Affiche les prévisions météorologiques sur plusieurs jours.
* **ErrorMessage.js** : Affiche les messages d'erreur si la ville n'est pas trouvée ou si l'API échoue.

### 3.2. Routing avec React Router

* Gérer la navigation au sein de l'application, même avec une interface à page unique.

## 4. Flux de Fonctionnement de l'Application

1. **Recherche de Ville** :
   * L'utilisateur entre le nom de la ville.
   * Une requête est envoyée à l'API météo via le back-end Laravel.
2. **Récupération des Données** :
   * Laravel interroge l'API météo et renvoie les données au front-end.
3. **Affichage des Données** :
   * Le front-end met à jour l'affichage avec les données météorologiques récupérées.

## 5. Données à Stocker

### 5.1. Informations Utilisateur

* **Nom** : Le nom de l'utilisateur.
* **Email** : Pour l'authentification ou notifications.
* **Préférences de localisation** : Ville ou région préférée.

### 5.2. Historique de Recherches

* **Recherches récentes** : Historique des villes recherchées avec date et heure.

### 5.3. Données Météorologiques

* **Conditions météo actuelles** : Température, humidité, conditions.
* **Données de réponse de l'API** : Pour éviter les requêtes répétées.

### 5.4. Préférences de Notification (si applicable)

* **Alertes Météo** : Préférences concernant les alertes (ex. alertes de tempête).
* **Fréquence des mises à jour**.

### 5.5. Feedback Utilisateur (si applicable)

* **Commentaires ou évaluations** : Pour recueillir des retours sur l'application.

## 6. Structure de la Base de Données

### 6.1. Tables

* **Utilisateurs**

| **Colonne** | **Type** |
| --- | --- |
| id | Integer (PK) |
| nom | String |
| email | String |
| preferences | JSON |
| created\_at | Timestamp |
| updated\_at | Timestamp |

* **Historique des Recherches**

| **Colonne** | **Type** |
| --- | --- |
| id | Integer (PK) |
| user\_id | Integer (FK) |
| ville | String |
| date\_recherche | Timestamp |

* **Données Météorologiques**

| **Colonne** | **Type** |
| --- | --- |
| id | Integer (PK) |
| ville | String |
| temperature | Float |
| humidity | Integer |
| wind\_speed | Float |
| condition | String |
| forecast | JSON |
| created\_at | Timestamp |

## 7. Conclusion

L'application vise à offrir une expérience personnalisée en utilisant les données d'historique des recherches et les préférences des utilisateurs, tout en s'appuyant sur une API de météo pour la récupération dynamique des données.