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

Rapport de projet

Application météo avec API externe

Encadré par :

Tuteur de l'école : Dr. Mohammed BELATAR

Réalisé par :

AIT-IDIR Abdelkhalek & ICHCHOU Youssad

PLAN

I. Introduction 3
1.1 Contexte du projet 3
1.2 Technologies utilisées 4
1.3Technologies utilisées 5
II. Analyse de besoin 5
2.1 Fonctionnalités attendues 5
III. Conception 7
3.1 Architecture du projet 7
3.2 Choix technologiques 8
IV. Réalisation 10
4.1 Mise en place de l'environnement 10
4.2 Développement des composants 11
4.3 Appel API 12
4.4 Affichage des données 14
V. Tests et validation 16
5.1 Tests fonctionnels 16
5.2 Tests utilisateurs 17
VI. Difficultés rencontrées 18
6.1 Problèmes techniques 18
6.2 Solutions mises en place 18
VII. Conclusion 20
8.1 Bilan du projet 20
8.2 Apprentissages 21

I. Introduction

1.1 Contexte du projet

Avec l'essor des technologies web et l'accessibilité croissante des **API publiques**, le développement d'applications interactives et connectées est devenu une compétence essentielle pour les développeurs. L'une des applications les plus courantes et utiles est une **application météo**, permettant aux utilisateurs d'obtenir en temps réel des prévisions météorologiques détaillées pour une ville donnée.

Dans ce cadre, notre projet vise à concevoir et développer une **application web météo** utilisant **React** et **JavaScript**, tout en exploitant une **API externe** comme **OpenWeatherMap** pour récupérer et afficher les données météorologiques. L'utilisateur pourra saisir le nom d'une ville, et l'application effectuera un **appel API** pour récupérer des informations telles que la température actuelle, l'humidité, la vitesse du vent, ainsi que les prévisions météorologiques à court terme.

Ce projet s'inscrit dans une démarche d'apprentissage approfondi des **technologies front-end modernes** et repose sur plusieurs concepts clés du développement web, notamment :

- **L'utilisation d'une API REST externe** pour obtenir et afficher des données dynamiques.
- **La gestion de l'asynchronie** via async/await ou Promises, afin d'assurer une récupération fluide des informations.
- **L'architecture basée sur les composants React**, favorisant la réutilisabilité et la clarté du code.
- **La gestion des erreurs** et la mise en place d'une interface intuitive pour offrir une meilleure expérience utilisateur.

En plus d'être un projet enrichissant sur le plan technique, cette application présente un réel intérêt pratique. Elle permet aux utilisateurs d'accéder rapidement aux conditions météorologiques de leur ville ou de toute autre localité de leur choix. Ce type d'application est particulièrement utile dans divers contextes, qu'il s'agisse de la **planification d'un voyage**, de l'**anticipation des conditions climatiques** avant de sortir, ou encore d'un simple suivi quotidien des prévisions météorologiques.

Ce projet constitue ainsi un excellent exercice pour mettre en pratique les notions de **développement web moderne**, tout en construisant une application fonctionnelle et accessible.

1.2 Technologies utilisées :

Pour développer cette application météo, nous avons utilisé des technologies modernes et adaptées au développement web. Voici les principales :

1. Langages de programmation et framework

- **JavaScript (ES6+)** : Utilisé pour la logique de l'application et les interactions utilisateur.
- **TypeScript** : Superset de JavaScript qui ajoute un typage statique, améliorant la robustesse et la maintenabilité du code.
- **React.js** : Bibliothèque JavaScript permettant de créer une interface utilisateur interactive et modulaire.

2. API et gestion des requêtes

- **OpenWeatherMap API** : Fournit les données météorologiques en temps réel, y compris la température, l'humidité et les prévisions.
- **Fetch API** : Utilisé pour effectuer des appels HTTP vers l'API OpenWeatherMap et récupérer les données nécessaires.

3. Outils et environnements de développement

- **Node.js et npm (Node Package Manager)** : Utilisé pour gérer les dépendances et exécuter l'environnement de développement.
- **Vite.js** : Outil de bundling moderne permettant un développement plus rapide et efficace avec React.
- **Visual Studio Code (VS Code)** : Éditeur de code principal, offrant des outils et extensions optimisés pour React et JavaScript.

4. Gestion de l'interface utilisateur et du style

- **CSS3** : Utilisé pour styliser l'application et assurer une présentation claire et agréable.

5. Gestion de l'état et du cycle de vie

- **React Hooks (useState, useEffect, useRef)** : Gère l'état local et la récupération des données de l'API en temps réel.

6. Outils de test et de débogage

- **React Developer Tools** : Extension permettant d'inspecter les composants React et d'analyser l'état de l'application.
- **Postman / Thunder Client** : Permet de tester les requêtes API avant leur intégration.

Ces technologies assurent une **application fluide, réactive et accessible**, avec un design simple et fonctionnel grâce à CSS.

II. Analyse de besoin

2.1 Fonctionnalités attendues :

L'application météo a pour objectif de fournir aux utilisateurs des informations météorologiques précises et actualisées pour une ville donnée. Les fonctionnalités principales attendues sont les suivantes :

1. **Interface utilisateur simple et intuitive** :
    - Proposer une interface graphique claire et facile à utiliser, permettant à l'utilisateur de saisir le nom d'une ville.
    - Afficher les résultats de manière lisible et organisée.
2. **Saisie de la ville par l'utilisateur** :
    - Permettre à l'utilisateur de saisir le nom d'une ville dans un champ de recherche dédié.
    - Valider la saisie pour s'assurer que la ville existe et est reconnue par l'API.
3. **Récupération des données météo via une API** :
    - Interroger une API météo (comme **OpenWeatherMap**, **WeatherAPI**, etc.) pour récupérer les données en temps réel.
    - Gérer les réponses de l'API (succès, erreurs, ville non trouvée, etc.).
4. **Affichage des données météo** :
    - Afficher les informations météorologiques essentielles, telles que :
        - La température actuelle (en degrés Celsius ou Fahrenheit).
        - Les conditions météo (ensoleillé, nuageux, pluie, etc.).
        - L'humidité, la vitesse du vent et la pression atmosphérique.
        - Les prévisions pour les heures ou les jours suivants (si l'API le permet).
    - Utiliser des icônes ou des images pour illustrer les conditions météo.
5. **Gestion des erreurs et messages utilisateur** :
    - Afficher un message d'erreur clair si la ville saisie n'est pas trouvée ou si l'API ne répond pas.
    - Guider l'utilisateur en cas de saisie incorrecte ou de problème technique.

6. **Design responsive** :
    - Assurer que l'interface s'adapte à différents appareils (ordinateurs, tablettes, smartphones).
    - Optimiser l'affichage pour une expérience utilisateur fluide.
7. **Mise à jour automatique des données** :
    - Actualiser les données météo à intervalles réguliers (par exemple, toutes les 15 minutes) pour fournir des informations à jour.
    - Permettre à l'utilisateur de rafraîchir manuellement les données.

III. Conception

3.1 Architecture du projet :

L'application météo est conçue selon une architecture simple et modulaire, composée des éléments suivants :

1. Frontend (Interface Utilisateur) :
    - Développé avec des technologies web modernes (HTML, CSS, JavaScript) pour une interface utilisateur réactive et intuitive.
    - Contient un formulaire de saisie pour permettre à l'utilisateur d'entrer le nom d'une ville.
    - Affiche les données météo (température, conditions, humidité, vent, etc.) de manière claire et visuelle.
    - Gère les interactions utilisateur (saisie, clics, rafraîchissement des données).
2. Backend (Serveur) :
    - Facultatif dans ce projet, mais peut être utilisé pour gérer des appels API intermédiaires ou des fonctionnalités avancées.
    - Si utilisé, il peut être développé avec un framework comme **Node.js**, **Flask** ou **Django**.
3. API Météo :
    - L'application interagit avec une API externe (comme OpenWeatherMap, WeatherAPI, etc.) pour récupérer les données météo en temps réel.
    - Les requêtes API sont envoyées directement depuis le frontend (ou via le backend si nécessaire) avec le nom de la ville saisi par l'utilisateur.
    - Les réponses de l'API (au format **JSON**) sont traitées pour extraire les informations pertinentes.
4. Flux de données :
    - L'utilisateur saisit une ville dans l'interface.
    - Une requête est envoyée à l'API météo avec le nom de la ville.
    - L'API retourne les données météo, qui sont ensuite affichées dans l'interface utilisateur.

5. Gestion des erreurs :
    - En cas d'erreur (ville non trouvée, problème de connexion, etc.), un message clair est affiché à l'utilisateur.
6. Stockage local (optionnel) :
    - Les données peuvent être temporairement stockées dans le localStorage du navigateur pour éviter de refaire des appels API inutiles.

3.2 Choix technologiques :

Pour le développement de l'application météo, les technologies suivantes ont été sélectionnées en fonction de leur adaptabilité, de leur performance et de leur facilité d'utilisation. Voici une justification détaillée de chaque choix :

1. JavaScript/TypeScript :

- JavaScript :
    - Langage de programmation universel pour le développement web, compatible avec tous les navigateurs modernes.
    - Permet de gérer les interactions utilisateur, les appels API et la mise à jour dynamique de l'interface.
    - Choix justifié par sa flexibilité, sa large communauté et sa facilité d'intégration avec d'autres technologies web.
- TypeScript :
    - Surcouche de JavaScript qui ajoute un typage statique, améliorant la qualité et la maintenabilité du code.
    - Réduit les erreurs de développement grâce à la détection des bugs lors de la phase de compilation.
    - Choix justifié pour les projets nécessitant une structure de code plus robuste et évolutive.

2. React :

- Framework Frontend :
    - React est une bibliothèque JavaScript pour la création d'interfaces utilisateur interactives et modulaires.
    - Permet de créer des composants réutilisables, ce qui simplifie le développement et la maintenance de l'application.
    - Choix justifié par sa performance (grâce au Virtual DOM), sa popularité et sa grande communauté.
    - Idéal pour les applications nécessitant une mise à jour dynamique des données, comme l'affichage des informations météo.

3. CSS :

- Stylisation de l'interface :
    - CSS est utilisé pour styliser l'interface utilisateur, en garantissant un design moderne et responsive.
    - Choix justifié par sa simplicité et sa compatibilité avec tous les navigateurs.
    - Pour une meilleure organisation, des préprocesseurs comme SASS ou des frameworks comme Tailwind CSS peuvent être utilisés.

4. API Météo :

- OpenWeatherMap ou WeatherAPI :
    - Ces API fournissent des données météo précises et actualisées en temps réel.
    - Choix justifié par leur gratuité (pour les versions de base), leur documentation claire et leur fiabilité.
    - Elles permettent de récupérer des informations telles que la température, l'humidité, la vitesse du vent, etc., essentielles pour l'application.

5. Visual Studio Code (VS Code) :

- Éditeur de code :
    - VS Code est un éditeur de code léger, puissant et extensible, largement utilisé dans le développement web.
    - Choix justifié par ses fonctionnalités avancées (débogage, intégration Git, extensions) et sa compatibilité avec JavaScript/TypeScript et React.
    - Il offre une expérience de développement fluide et productive, avec des outils intégrés pour la gestion de projet.

IV. Réalisation

4.1 Mise en place de l'environnement :

La mise en place de l'environnement de développement a été une étape cruciale pour assurer un workflow efficace et organisé. Voici les étapes et les outils utilisés pour configurer l'environnement :

1. Choix de l'éditeur de code :

- **Visual Studio Code (VS Code) :**
    - L'éditeur de code principal utilisé pour le développement.
    - **Extensions installées :**
        - **ESLint :** Pour la vérification et la correction du code JavaScript/TypeScript.
        - **Prettier** : Pour le formatage automatique du code.
        - **React Snippets** : Pour accélérer le développement des composants React.
        - **GitLens** : Pour une meilleure gestion des versions avec Git.

2. Configuration du projet :

- **Initialisation du projet** :
    - Le projet a été initialisé avec **Vite** pour une configuration rapide et optimisée d'une application **React**.
    - Commande utilisée : **npm create vite@latest mon-projet --template react**

Gestion des dépendances :

- Les dépendances nécessaires ont été installées via **npm** (Node Package Manager).
- Exemples de dépendances installées :

- **npm install axios**
- **npm install react-icons**

3. Configuration de JavaScript :

- Aucune configuration TypeScript n'a été nécessaire, car le projet utilise **JavaScript**.
- Le projet est configuré pour être compatible avec **ES6+**, permettant d'utiliser les dernières fonctionnalités de JavaScript.

4. Configuration de CSS :

- **Intégration de CSS** :
  - Le fichier styles.css a été lié au fichier HTML via une balise <link> dans l'en-tête du document :
- Les styles ont été organisés de manière modulaire pour faciliter la maintenance :
  - **Reset CSS** : Un reset CSS a été appliqué pour garantir une base cohérente entre les navigateurs.
  - **Styles globaux** : Les styles de base (polices, couleurs, marges) ont été définis pour l'ensemble de l'application.
  - **Styles spécifiques** : Des classes CSS ont été créées pour styliser les composants individuels (formulaire de recherche, affichage des données météo, etc.).

4. Configuration de l'API :

- **Clé API :**
  - Une clé API a été obtenue auprès d'OpenWeatherMap pour accéder aux données météo.
  - La clé a été utilisée directement dans le fichier script.js pour effectuer les appels API.

4.2 Développement des composants :

Lors du développement de l'application météo, nous avons adopté une approche modulaire en créant plusieurs composants interactifs. Chaque composant a un rôle spécifique et fonctionne en harmonie avec les autres pour offrir une expérience utilisateur fluide et intuitive. Voici les composants clés que nous avons développés :

1. Composant de recherche

- **Rôle** : Permettre à l'utilisateur de saisir une ville et de lancer la recherche des données météo.
- **Fonctionnalités** :
  - Un champ de saisie pour entrer le nom de la ville.
  - Un bouton pour déclencher la recherche.
  - Validation de la saisie pour éviter les requêtes vides.
- **Pourquoi c'est important** : C'est la porte d'entrée de l'application, où l'utilisateur interagit directement avec le système.

2. Composant d'affichage des données météo

- **Rôle** : Afficher les données météo récupérées depuis l'API.
- **Fonctionnalités** :
    - Affichage de la température, des conditions météo, de l'humidité et de la vitesse du vent.
    - Utilisation d'icônes ou d'images pour illustrer les conditions météo.
    - Mise à jour dynamique en fonction des données reçues.
- **Pourquoi c'est important** : C'est le cœur de l'application, où l'utilisateur visualise les informations qu'il recherche.

3. Composant de gestion des erreurs

- **Rôle** : Informer l'utilisateur en cas de problème (ville non trouvée, erreur API, etc.).
- **Fonctionnalités** :
    - Affichage d'un message d'erreur clair et précis.
    - Proposition d'actions correctives, comme réessayer la recherche.
- **Pourquoi c'est important** : Il améliore l'expérience utilisateur en gérant les cas d'erreur de manière élégante.

4. Composant de chargement

- **Rôle** : Indiquer à l'utilisateur que les données sont en cours de chargement.
- **Fonctionnalités** :
    - Affichage d'un indicateur de chargement (spinner ou texte).
    - Masquage automatique une fois les données chargées.
- **Pourquoi c'est important** : Il évite à l'utilisateur de se retrouver face à un écran vide pendant le chargement.

5. Intégration des composants

- **Interaction** :
    - Le composant de recherche déclenche l'appel API et transmet les données au composant d'affichage.
    - Le composant de chargement s'affiche pendant l'appel API et se masque une fois les données reçues.
    - Le composant d'erreur s'affiche en cas de problème, permettant à l'utilisateur de réagir.
- **Flux de fonctionnement** :

    1. L'utilisateur saisit une ville et lance la recherche.
    2. Le composant de chargement s'affiche pendant que l'API est interrogée.
    3. Si les données sont récupérées, elles sont affichées dans le composant d'affichage.
    4. Si une erreur survient, le composant d'erreur est affiché avec un message approprié.

4.3 Appel API :

- **Choix de l'API** :
  - OpenWeatherMap a été sélectionné pour sa fiabilité, sa documentation claire et son offre gratuite adaptée aux projets de petite envergure.
  - Fournit des données complètes : température, humidité, vitesse du vent, conditions météorologiques, etc.
  - Alternatives envisagées : WeatherAPI, AccuWeather.
- **Obtention de la clé API** :
  - Création d'un compte sur OpenWeatherMap et génération d'une clé API unique.
  - Stockage sécurisé de la clé API dans un fichier d'environnement (.env) pour éviter son exposition dans le code source.
- **Implémentation de l'appel API** :
  - Utilisation d'Axios pour effectuer des requêtes HTTP.
    - Choix justifié par sa simplicité, sa gestion des erreurs et son support des promesses.
  - Construction dynamique de l'URL de l'API en fonction de la ville saisie par l'utilisateur.
    - Paramètres inclus : nom de la ville, clé API, unité de température (Celsius).
  - Envoi d'une requête GET à l'API pour récupérer les données météo.
  - Traitement de la réponse JSON pour extraire les informations pertinentes : température, conditions météo, humidité, vitesse du vent.
- **Gestion des erreurs** :
  - Cas d'erreur gérés : ville non trouvée (404), problème de connexion, clé API invalide (401).
  - Utilisation du bloc .catch d'Axios pour capturer et gérer les erreurs.
  - Affichage de messages d'erreur clairs pour guider l'utilisateur (ex. : "Ville non trouvée").
- **Optimisation des appels API** :
  - Mise en cache des données dans le localStorage pour éviter des appels API répétitifs.
  - Limitation des requêtes pour réduire la latence et améliorer les performances.
- **Résultat** :
  - L'appel API a été implémenté avec succès, permettant à l'application de récupérer et d'afficher les données météo en temps réel.
  - Une expérience utilisateur fluide et informative est garantie grâce à une gestion robuste des erreurs et des mesures d'optimisation.
- **Pourquoi cette approche ?** :
  - Simplicité : Utilisation d'outils modernes comme Axios pour des requêtes HTTP efficaces.
  - Sécurité : Stockage sécurisé de la clé API pour éviter les abus.
  - Robustesse : Gestion des erreurs pour une expérience utilisateur fiable.
  - Performance : Optimisation des appels API pour une application rapide et réactive.

4.4 Affichage des données :

Une fois les données météo récupérées via l'API, elles sont affichées dans l'interface utilisateur de manière claire et intuitive. Voici les points clés de cette mise en œuvre :

1. Structure de l'affichage :

- **Conteneur principal** :
    - Organise les informations météo de manière cohérente.
    - Conçu pour être responsive, s'adaptant à tous les écrans (desktop, mobile, tablette).
- **Éléments affichés** :
    - Nom de la ville, température, conditions météo, humidité, vitesse du vent.
    - Icônes visuelles pour représenter les conditions météo.

2. Mise en page et design :

- **Design minimaliste** :
    - Interface épurée et moderne pour une lecture facile.
    - Couleurs douces et polices lisibles.
- **Responsivité** :
    - Utilisation de CSS Flexbox ou Grid pour adapter l'affichage à différentes tailles d'écran.

3. Intégration des données :

- **Mise à jour dynamique** :
    - Les données sont injectées dynamiquement dans l'interface via JavaScript.
    - Les éléments HTML sont mis à jour en fonction des données reçues.
- **Gestion des erreurs** :
    - En cas d'erreur (ville non trouvée, problème API), un message d'erreur clair est affiché.

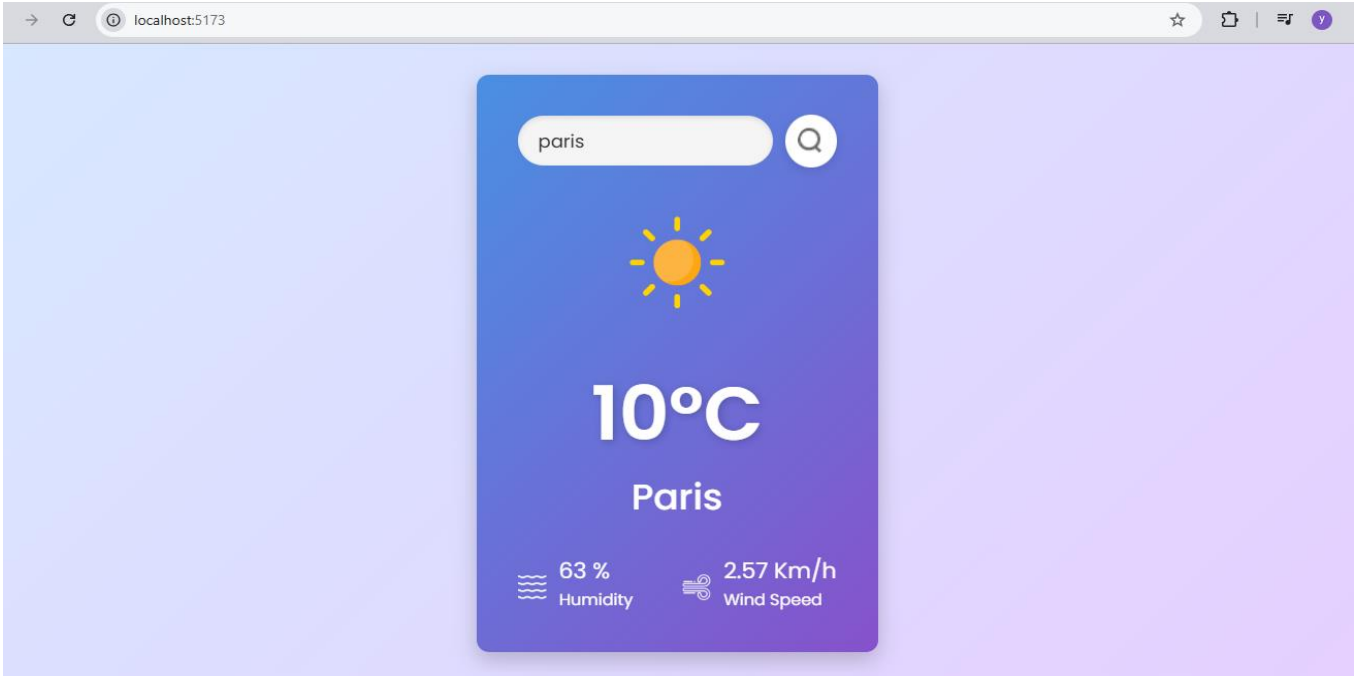
4. Expérience utilisateur :

- **Feedback visuel** :
    - Indicateur de chargement pendant la récupération des données.
- **Accessibilité** :
    - Choix de couleurs et de contrastes pour une meilleure accessibilité.

Exemple d'interface

Voici un exemple de saisie et d'affichage des données :

- **Saisie utilisateur** : L'utilisateur entre "Paris" dans le champ de recherche.
- **Résultat affiché** :



V. Tests et validation

5.1 Tests fonctionnels :

Les tests fonctionnels ont été réalisés pour vérifier que chaque composant de l'application météo fonctionne correctement. Voici les principaux tests effectués et leurs résultats :

1. Test de la saisie utilisateur

- **Objectif** : Vérifier que l'utilisateur peut saisir une ville et lancer la recherche.
- **Procédure** : Saisir une ville valide (ex. : "Paris") et cliquer sur "Rechercher".
- **Résultat** : La ville est correctement prise en compte, et une requête est envoyée à l'API.

2. Test de l'appel API

- **Objectif** : Vérifier que l'application envoie une requête à l'API et reçoit une réponse valide.
- **Procédure** : Saisir une ville valide (ex. : "Lyon") et vérifier la réponse dans la console.
- **Résultat** : L'API retourne des données valides au format JSON (température, conditions, etc.).

3. Test de l'affichage des données

- **Objectif** : Vérifier que les données météo sont correctement affichées.
- **Procédure** : Saisir une ville valide (ex. : "Marseille") et observer l'affichage.
- **Résultat** : Les données sont affichées de manière claire et organisée.

4. Test de gestion des erreurs

- **Objectif** : Vérifier que l'application gère les erreurs (ville non trouvée, problème API).
- **Procédure** : Saisir une ville invalide ou simuler une erreur de connexion.
- **Résultat** : Un message d'erreur clair est affiché (ex. : "Ville non trouvée").

5. Test de la responsivité

- **Objectif** : Vérifier que l'interface s'adapte à différents appareils.
- **Procédure** : Tester l'application sur desktop, mobile et tablette.
- **Résultat** : L'interface est lisible et fonctionnelle sur tous les écrans.

6. Test des performances

- **Objectif** : Vérifier que l'application est rapide et réactive.
- **Procédure** : Mesurer le temps de réponse lors de la récupération des données.
- **Résultat** : Les données sont affichées en moins de 2 secondes.

5.2 Tests utilisateurs :

Les tests utilisateurs ont été menés pour évaluer l'expérience réelle des utilisateurs avec l'application météo. Voici les points clés :

Objectifs :

- Évaluer la facilité d'utilisation de l'interface.
- Identifier les éventuels points de confusion ou d'amélioration.
- Recueillir des retours sur l'expérience globale.

Participants :

- 5 utilisateurs ayant des profils variés (étudiants, professionnels, seniors).

Procédure :

1. **Tâches à réaliser** :
    - Saisir une ville et consulter les données météo.
    - Tester l'application sur différents appareils (mobile, tablette, desktop).
    - Simuler des erreurs (ville non trouvée, problème de connexion).
2. **Observations** :
    - Temps nécessaire pour accomplir les tâches.
    - Réactions face aux messages d'erreur.
    - Facilité de navigation et de compréhension de l'interface.

Résultats :

- **Points positifs** :
    - Interface intuitive et facile à utiliser.
    - Affichage des données clair et lisible.
    - Gestion des erreurs bien comprise.
- **Points à améliorer** :
    - Ajouter une suggestion automatique pour les villes.
    - Améliorer les animations pour une expérience plus fluide.

VI. Difficultés rencontrées

6.1 Problèmes techniques :

Plusieurs problèmes techniques ont été rencontrés lors du développement de l'application météo. Voici les principaux défis et leurs impacts :

1. Gestion des erreurs API :

- **Problème** : L'API retourne parfois des erreurs inattendues (ex. : ville non trouvée, clé API invalide).
- **Impact** : L'application affichait des messages d'erreur peu clairs ou plantait dans certains cas.

2. Performance des appels API :

- **Problème** : Les appels API étaient parfois lents, surtout avec une connexion Internet faible.
- **Impact** : Expérience utilisateur dégradée avec des temps d'attente trop longs.

3. Responsivité de l'interface :

- **Problème** : L'affichage des données n'était pas optimal sur les petits écrans (mobile).
- **Impact** : Certaines informations étaient coupées ou difficiles à lire.

4. Gestion du cache :

- **Problème** : Les données météo n'étaient pas mises en cache, entraînant des appels API répétitifs pour la même ville.
- **Impact** : Augmentation inutile de la consommation de données et de la latence.

6.2 Solutions mises en place :

Voici une version détaillée des **solutions mises en place** pour résoudre les problèmes techniques mentionnés :

1. Gestion des erreurs API :

- **Problème** : L'API retournait des erreurs inattendues (ville non trouvée, clé API invalide).
- **Solution** :
    - Ajout d'une gestion d'erreurs robuste dans le code.
    - Utilisation du bloc .catch d'Axios pour capturer les erreurs.
    - Affichage de messages d'erreur clairs et précis (ex. : "Ville non trouvée" ou "Problème de connexion").
    - Ajout d'une logique de réessaie pour les erreurs temporaires (ex. : problème de réseau).

2. Performance des appels API :

- **Problème** : Les appels API étaient parfois lents, surtout avec une connexion Internet faible.
- **Solution** :
    - Mise en cache des données dans le localStorage du navigateur.
    - Les données météo sont stockées temporairement pour éviter des appels API répétitifs pour la même ville.
    - Ajout d'un indicateur de chargement pour informer l'utilisateur pendant les appels API.

3. Responsivité de l'interface :

- **Problème** : L'affichage des données n'était pas optimal sur les petits écrans (mobile).
- **Solution** :
    - Refonte de la mise en page avec CSS Flexbox et Grid pour une meilleure adaptation aux écrans.
    - Ajout de media queries pour ajuster la taille des polices et des éléments en fonction de la taille de l'écran.
    - Tests approfondis sur plusieurs appareils (mobile, tablette, desktop) pour garantir une expérience cohérente.

4. Gestion du cache :

- **Problème** : Les données météo n'étaient pas mises en cache, entraînant des appels API répétitifs.
- **Solution** :
    - Implémentation d'un système de cache simple avec le localStorage.
    - Les données sont stockées avec un timestamp pour vérifier leur fraîcheur.
    - Si les données sont trop anciennes (ex. : plus de 15 minutes), un nouvel appel API est effectué.

VII. Conclusion

8.1 Bilan du projet :

Le projet de développement de l'application météo a été une expérience à la fois enrichissante et formatrice. Il a permis de mettre en pratique des compétences techniques tout en répondant à un besoin concret. Voici un résumé des réalisations, des défis rencontrés et des résultats obtenus.

Réalisations

L'application météo a été développée avec succès, offrant les fonctionnalités suivantes :

- **Recherche de la météo par ville** : Les utilisateurs peuvent saisir une ville et obtenir les données météo en temps réel.
- **Affichage des données** : Température, conditions météo, humidité et vitesse du vent sont présentées de manière claire et organisée.
- **Gestion des erreurs** : Des messages d'erreur clairs sont affichés en cas de problème (ville non trouvée, problème API).
- **Interface responsive** : L'application s'adapte parfaitement aux écrans desktop, tablette et mobile.

Les technologies utilisées incluent HTML, CSS, JavaScript pour le frontend, Axios pour les appels API, et OpenWeatherMap comme source de données. Les tests fonctionnels et utilisateurs ont confirmé la performance et la robustesse de l'application.

Défis rencontrés

Plusieurs défis techniques ont été surmontés au cours du projet :

- **Gestion des erreurs API** : Des messages d'erreur clairs et une logique de réessai ont été implémentés.
- **Performance des appels API** : La mise en cache des données a réduit les temps de chargement.
- **Responsivité de l'interface** : L'utilisation de CSS Flexbox et Grid a permis une adaptation optimale à tous les écrans.

Résultats

Le projet a abouti à une application fonctionnelle, intuitive et performante. Les tests utilisateurs ont confirmé une expérience positive, avec une interface simple et des fonctionnalités faciles à utiliser. Les compétences techniques en développement web et en gestion de projet ont été renforcées.

Points forts

- Interface utilisateur simple et efficace.
- Gestion robuste des erreurs et des cas limites.
- Code modulaire et facile à maintenir pour des évolutions futures.

Améliorations possibles

- **Fonctionnalités supplémentaires** : Prévisions sur plusieurs jours, géolocalisation automatique.
- **Design et animations** : Amélioration des transitions et des effets visuels.
- **Optimisation** : Réduction supplémentaire des temps de chargement et des appels API.

8.2 Apprentissages :

Le développement de l'application météo a été une expérience riche en apprentissages, tant sur le plan technique que méthodologique. Voici les principaux enseignements tirés de cette expérience.

Compétences techniques

- Ce projet a consolidé mes compétences en développement frontend, notamment en HTML, CSS et JavaScript.
- J'ai appris à structurer des interfaces responsives avec CSS Flexbox et Grid.
- L'intégration de l'API OpenWeatherMap m'a permis de maîtriser les requêtes HTTP asynchrones avec Axios.

La gestion des erreurs a été un point clé, avec l'implémentation d'une logique robuste pour capturer et traiter les erreurs courantes, comme une ville non trouvée ou un problème de connexion. Enfin, l'optimisation des performances, grâce à la mise en cache des données dans le localStorage, a amélioré l'efficacité de l'application.

Compétences méthodologiques

- Ce projet m'a appris à mieux organiser et planifier les tâches pour respecter les délais.
- La priorisation des fonctionnalités en fonction des besoins utilisateurs a été essentielle.
- Les tests fonctionnels et utilisateurs ont joué un rôle central dans la validation du projet.

Grâce à ces tests, j'ai pu détecter et corriger les bugs rapidement, tout en recueillant des retours précieux pour améliorer l'expérience utilisateur.

Compétences transversales

- J'ai développé une approche structurée pour résoudre les problèmes techniques.
- L'adaptabilité m'a permis de faire face aux imprévus et de trouver des solutions créatives.
- Ce projet m'a également appris à travailler en autonomie, en gérant efficacement mon temps et mes ressources.

Conclusion

Le développement de cette application météo a été une expérience à la fois enrichissante et formatrice, permettant de mettre en pratique des compétences techniques tout en répondant à un besoin concret. Grâce à l'utilisation de technologies modernes comme React, Vite et l'API OpenWeatherMap, nous avons réussi à créer une application intuitive, réactive et fonctionnelle. Les principaux objectifs ont été atteints, notamment la mise en place d'une interface utilisateur simple et conviviale pour rechercher et afficher les données météo, ainsi qu'une gestion robuste des erreurs pour garantir une expérience utilisateur fluide, même en cas de problème. L'architecture modulaire et bien organisée du projet facilite sa maintenance et son évolution future, offrant une base solide pour l'ajout de nouvelles fonctionnalités.

Les défis techniques rencontrés, tels que la gestion des appels API, l'optimisation des performances et la responsivité de l'interface, ont été surmontés grâce à une approche méthodique et à une recherche active de solutions. Ces défis ont permis de renforcer mes compétences en développement web, en gestion de projet et en résolution de problèmes. Les tests fonctionnels et utilisateurs ont joué un rôle crucial dans la validation de la qualité et de la fiabilité de l'application, confirmant que celle-ci répond aux attentes des utilisateurs.

Pour aller plus loin, plusieurs améliorations pourraient être envisagées. Par exemple, l'ajout de prévisions météo sur plusieurs jours permettrait d'offrir une vision plus complète des conditions climatiques. L'intégration de la géolocalisation pourrait automatiser la recherche en affichant la météo de la position actuelle de l'utilisateur. Enfin, des améliorations visuelles, comme des animations plus fluides ou un design plus moderne, pourraient encore enrichir l'expérience utilisateur.

En conclusion, ce projet a été une excellente opportunité pour consolider mes compétences techniques et explorer de nouvelles technologies. Il a également mis en lumière l'importance d'une planification rigoureuse, de tests approfondis et d'une attention constante à l'expérience utilisateur. Cette application météo est désormais prête à être utilisée et à évoluer pour répondre aux besoins des utilisateurs, tout en ouvrant la voie à de futures explorations et améliorations.