

DIAW ANNA
LAARIF ABDELLAH

RAPPORT DE PROJET ANNUEL : WEATHERAPP

INTRODUCTION :	2
1 DEMARCHE DU PROJET :	2
1.2.1_Backend :	2
1.2.2_Frontend :	3
2 CONCEPTION ET DEVELOPPEMENT - ALGORITHMES CLES :	4
2.3.1_Composant Weather :	6
2.3.2_Composant FavoriteCities :	6
2.3.3_Composant Search :	7
2.3.4_Composant HourlyForecast et dailyForecast :	8
2.3.5_Composant MaxMinTemperature :	9
2.3.6_Composants de détails météorologiques :	9
2.4.1_Données d'entraînement :	14
2.4.2_Choix du modèle :	15
2.4.2.1_CNN :	15
2.4.2.2_LSTM :	15
2.4.2.3_Pmdarima :	16
2.4.2.4_SARIMA :	17
2.4.2.5_NeuralProphet :	17
2.4.2.6_GBM :	18
2.4.3_Prédiction de Température avec GBM :	18
3 RÉSULTATS ET FONCTIONNALITÉS DE L'APPLICATION :	19
CONCLUSION :	21

INTRODUCTION :

Dans un monde en constante mutation, où les frontières s'estompent et où la technologie redéfinit les interactions, la nécessité d'une information précise et en temps réel est plus impérative que jamais. Au cœur de ce maelström d'informations, la météorologie émerge comme l'une des données les plus sollicitées, influençant de manière profonde et variée notre quotidien. Que ce soit pour les voyageurs cherchant à optimiser leurs itinéraires, les agriculteurs désireux de protéger leurs récoltes des aléas climatiques, ou tout simplement pour chacun d'entre nous, hésitant devant sa garde-robe le matin, la météo est devenue un élément incontournable de notre processus décisionnel.

Face à cette demande croissante, et reconnaissant l'impact indéniable de la météorologie sur nos vies, nous avons initié un projet audacieux : la conception d'une application météorologique de pointe. Plus qu'une simple plateforme d'affichage, notre vision était de développer une solution innovante capable de fournir des données météorologiques en temps réel pour chaque coin du globe. Et notre ambition ne s'arrêtait pas là. Nous visions également à intégrer des capacités prédictives, permettant à nos utilisateurs d'obtenir des prévisions à court et moyen terme, une véritable prouesse technologique au service de la précision.

Ce rapport est le récit de notre voyage. Il dépeint notre cheminement, des premières esquisses aux choix technologiques cruciaux, en passant par les montagnes russes des défis rencontrés et des solutions élaborées. Plus qu'une simple description, nous y dévoilons les rouages des algorithmes qui sont le cœur battant de notre application, illustrant l'innovation et l'excellence qui distinguent notre travail. Embarquez avec nous pour découvrir les coulisses d'un projet qui, nous l'espérons, redéfinira la manière dont nous interagissons avec la météorologie.

1 DEMARCHE DU PROJET :

1.1 SOURCES D'INFORMATIONS ET BASES DE DONNÉES MÉTÉOROLOGIQUES :

Notre premier défi était d'identifier une source fiable et complète pour les données météorologiques. Après une recherche approfondie, nous avons choisi d'utiliser l'API **WeatherAPI**, réputée pour sa précision et sa couverture mondiale. Nous utilisons aussi Geonames connu pour sa base de donnée complète sur toutes les villes du monde.

1.2 OUTILS ET TECHNOLOGIES ENVISAGÉS :

1.2.1 BACKEND :

Nous avons opté pour Node.js avec Express en raison de sa performance, de sa flexibilité et de sa large adoption dans la communauté des développeurs. Cette combinaison s'est avérée efficace pour gérer des requêtes en temps réel et interagir avec notre base de données.

1.2.2_FRONTEND :

React Native a été notre choix pour le développement de l'interface utilisateur. Connue pour sa portabilité entre les plateformes iOS et Android et son approche basée sur les composants, il nous a permis de créer une interface utilisateur réactive et intuitive.

- **Système de cache :**

Afin d'optimiser l'utilisation de notre API météorologique et de ne pas dépasser les limites d'appels, nous avons intégré un système de cache utilisant MongoDB. Cela nous permet de stocker temporairement les données de météo pour une ville donnée et de ne pas solliciter l'API inutilement dans un délai d'une heure.

- **Recherche de villes :**

Pour faciliter la recherche des villes et proposer des suggestions pertinentes aux utilisateurs en fonction de leurs saisies, nous avons combiné la puissance d'ElasticSearch avec la base de données de Geonames. Geonames, étant une base de données géographique complète, nous a fourni des informations détaillées sur chaque ville, améliorant ainsi la précision de nos suggestions et enrichissant l'expérience utilisateur. Cette synergie entre Elastic et Geonames a rendu notre fonction de recherche robuste, rapide et précise.

- **Tests d'API :**

Postman, un outil de collaboration de plateforme pour le développement d'API, a été utilisé pour tester, documenter et surveiller notre API, assurant ainsi une intégration transparente entre nos services.

- **Hébergement :**

Nous avons choisi AWS pour le déploiement en raison de sa scalabilité, de sa fiabilité et de sa panoplie de services adaptés à nos besoins, notamment pour l'exécution de notre modèle de machine learning dédié aux prévisions météorologiques.

2_ CONCEPTION ET DEVELOPPEMENT - ALGORITHMES CLES :

2.1_ GEONAMES ET ELASTICSEARCH POUR UNE RECHERCHE INTELLIGENTE ET EFFICACE :

La capacité de nos utilisateurs à rechercher efficacement des villes du monde entier est au cœur de l'expérience que nous souhaitons offrir. Pour y parvenir, nous avons allié la richesse de la base de données GeoNames à la puissance de la plateforme de recherche ElasticSearch.

- L'exploitation de GeoNames :**

GeoNames, avec son vaste répertoire géographique, était le candidat idéal pour obtenir des informations détaillées sur presque toutes les localités à travers le globe. Nous avons entamé le processus en extrayant les noms des villes et leurs codes de pays correspondants depuis le fichier allCountries.txt. Plutôt que de se contenter de simples codes, nous avons souhaité améliorer l'intelligibilité de nos données et l'adapter au mieux à nos appels à l'API. Pour ce faire, nous avons utilisé un dictionnaire de correspondance pour remplacer ces codes par le nom intégral des pays, enrichissant ainsi les données présentées à l'utilisateur. Après cette transformation, les données ont été formatées et enregistrées en JSON, un format plus adapté pour la suite de notre intégration avec ElasticSearch.

- L'intégration à ElasticSearch :**

ElasticSearch s'est imposé comme une solution évidente pour nous, étant reconnu pour sa rapidité, sa précision et son adaptabilité. Avec nos données méticuleusement préparées au format JSON, l'étape suivante a été leur indexation dans ElasticSearch. Grâce à la fonction bulk du module helpers, nous avons été en mesure d'optimiser cette indexation, assurant ainsi une mise en ligne rapide et une recherche instantanée pour nos utilisateurs.

En choisissant cette approche combinant GeoNames et ElasticSearch, nous n'avons pas simplement opté pour la facilité ou la popularité de ces outils. Nous avons reconnu que pour offrir une expérience utilisateur inégalée, nous avions besoin de la précision des données de GeoNames, tout en exploitant la vélocité et la flexibilité d'ElasticSearch. Cette décision était aussi tournée vers l'avenir, anticipant une croissance du nombre d'utilisateurs et garantissant que notre système pourrait évoluer et s'adapter sans compromettre la qualité du service.

Ce duo nous offre une grande flexibilité pour implémenter des fonctionnalités supplémentaires à l'avenir, comme des suggestions basées sur la pertinence ou des filtres de recherche avancée.

2.2_ OPTIMISATION DE L'ACCÈS AUX DONNÉES MÉTÉOROLOGIQUES : CACHE, API ET MODÉLISATION :

L'accès en temps réel à des informations météorologiques précises est fondamental pour notre application. Pour y parvenir, nous avons mis en œuvre une solution combinant l'utilisation de l'API WeatherAPI, une optimisation par mise en cache, et une modélisation soignée des données météorologiques.

Intégration de WeatherAPI :

WeatherAPI se présente comme notre principal fournisseur de données. Lorsqu'un utilisateur souhaite connaître la météo d'une ville, notre première démarche est de consulter notre système de cache MongoDB. Cette étape s'inscrit dans une volonté de minimiser la sollicitation de WeatherAPI et d'éviter d'engendrer des coûts additionnels liés à une surutilisation. Si les données du cache sont obsolètes (plus d'une heure) ou inexistantes, c'est à ce moment que nous interrogeons WeatherAPI pour recueillir les informations météorologiques les plus à jour.

Stratégie de mise en cache :

La mise en cache est essentielle pour offrir une expérience utilisateur fluide et réactive. Une fois que nous obtenons des données de WeatherAPI, celles-ci sont immédiatement stockées dans notre système MongoDB. Chaque enregistrement comprend un timestamp, ce qui permet, lors des demandes ultérieures, de vérifier la fraîcheur de ces données.

Modélisation et structuration des données météorologiques :

Pour assurer une présentation claire et pertinente des informations à nos utilisateurs, nous avons défini un modèle dédié, nommé WeatherData. Celui-ci organise les données en trois segments distincts :

- **Localisation** : Comportant des détails comme le nom de la ville, la région, le pays, et d'autres données géographiques.
- **Météo actuelle** : Regroupant des indicateurs en temps réel tels que la température, l'humidité, la qualité de l'air, entre autres.
- **Prévisions horaires** : Fournissant une vue détaillée des conditions météorologiques sur l'ensemble de la journée, avec des mises à jour heure par heure.

Cette modélisation nous permet de garantir que les utilisateurs bénéficient d'une vision complète, mais aussi synthétique, des prévisions météorologiques, optimisant ainsi leur expérience. Nous garantissons une présentation des données météorologiques à la fois complète et épurée, facilitant ainsi la compréhension et l'utilisation par nos utilisateurs.

```
class WeatherData {
  constructor(data) {
    this.data = data;
    this.location = {
      name: data.location.name,
      region: data.location.region,
      country: data.location.country,
      lat: data.location.lat,
      lon: data.location.lon,
      tz_id: data.location.tz_id,
      localtime_epoch: data.location.localtime_epoch,
      localtime: data.location.localtime
    };
    this.current = {
      temperature: data.current.temp_c,
      temperature_f: data.current.temp_f,
      feelslike_c: data.current.feelslike_c,
      feelslike_f: data.current.feelslike_f,
      condition: data.current.condition.text,
      wind_kph: item.wind_kph,
      wind_mph: item.wind_mph,
      wind_dir: data.current.wind_dir,
      precip_mm: data.current.precip_mm,
      humidity: data.current.humidity,
      uv: data.current.uv,
      dewpoint_c: data.current.dewpoint_c,
      dewpoint_f: data.current.dewpoint_f,
      pressure_mb: data.current.pressure_mb,
      pressure_mbar: item.pressure_mb,
      humidity: data.current.humidity,
      cloud: data.current.cloud,
      air_quality: data.current.air_quality,
    };
    this.hourlyForecast = data.forecast.forecastday[0].hour.map(item => {
      return {
        time: item.time,
        temperature: item.temp_c,
        temperature_f: item.temp_f,
        condition: item.condition.text,
        wind_kph: item.wind_kph,
        pressure_mb: item.pressure_mb,
        humidity: item.humidity,
        cloud: item.cloud,
        uv: item.uv,
        dewpoint_c: item.dewpoint_c,
        dewpoint_f: item.dewpoint_f,
        feelslike_c: item.feelslike_c,
        feelslike_f: item.feelslike_f,
      }
    });
  }
}

module.exports = WeatherData;
```

Structure de données

2.3 MISE EN ŒUVRE DU FRONTEND : L'EXPÉRIENCE UTILISATEUR AVEC REACT NATIVE :

React Native, reconnu pour sa capacité à créer des applications mobiles performantes à l'aide de JavaScript, a été notre choix pour développer l'interface utilisateur de notre application. Cette section explorera la manière dont nous avons utilisé React Native pour traduire les fonctionnalités backend précédemment discutées en une interface conviviale et réactive, permettant aux utilisateurs de naviguer facilement et d'obtenir les informations météorologiques souhaitées avec un minimum d'effort.

La conception de notre interface utilisateur repose sur une architecture modulaire, où chaque composant joue un rôle précis dans la présentation des données et l'interaction avec l'utilisateur.

2.3.1 COMPOSANT WEATHER :

Le composant **Weather** est le composant principal de l'application. Il sert de conteneur principal et orchestre les interactions entre les autres sous-composants. Il s'assure que les données météorologiques sont présentées de manière cohérente.

Grâce à lui, l'utilisateur peut consulter en temps réel les données météorologiques pour une ville donnée. Il affiche des informations telles que la température, la sensation thermique, le taux d'humidité, la qualité de l'air, parmi d'autres paramètres essentiels.

À l'intérieur de ce composant, nous avons utilisé ces deux hooks fondamentaux de React pour gérer le cycle de vie de notre composant et le rendu dynamique des données. useState nous a permis de déclarer et gérer les états locaux, tandis que useEffect a été crucial pour les effets secondaires, notamment pour appeler les données météorologiques lorsque la ville sélectionnée change.

Nous utilisons une fonction asynchrone pour récupérer les données météorologiques d'une API locale. En cas de données déjà présentes dans les villes favorites de l'utilisateur, nous utilisons ces données en cache pour réduire les appels inutiles.

Une autre fonction clé est notre fonction de basculement (toggleSection) qui permet à l'utilisateur de montrer ou cacher les différentes sections (comme la qualité de l'air, la prévision horaire, etc.) pour une meilleure expérience utilisateur.

2.3.2 COMPOSANT FAVORITECITIES :

L'objectif principal du composant **FavoriteCities** est de fournir à l'utilisateur une liste visuelle de ses villes favorites avec leurs conditions météorologiques actuelles. Il était crucial pour nous de garder

l'interface utilisateur propre et intuitive, tout en offrant la possibilité de supprimer rapidement une ville de la liste des favoris.

Nous avons cartographié chaque ville favorite pour afficher des informations pertinentes : le nom de la ville, le pays, l'heure locale, la condition météorologique actuelle et la température (avec prise en compte du choix de l'utilisateur entre Celsius et Fahrenheit).

Un petit défi que nous avons rencontré était d'intégrer une fonction de suppression intuitive sans surcharger l'interface. La solution a été d'incorporer une icône "coeur" qui, lorsqu'elle est touchée, supprime la ville de la liste des favoris.

Nous avons envisagé différentes façons d'afficher les villes favorites, y compris l'utilisation de cartes ou de listes déroulantes. Cependant, pour garder la simplicité, nous avons opté pour une mise en page en rangée. L'ajout de l'ombre à chaque ville est une tentative d'améliorer l'esthétique tout en gardant une distinction claire entre les éléments.

Ce composant intègre une combinaison de fonctionnalités essentielles qui améliorent l'expérience utilisateur, lui permettant de rester informé et en contrôle de ses choix.

2.3.3_ COMPOSANT SEARCH :

Le composant **Search** est conçu pour fournir une interface de recherche à l'utilisateur, principalement pour les villes. L'aspect visuel du composant est dominé par une barre de recherche accompagnée d'une icône de loupe, donnant une sensation familière de recherche à l'utilisateur.

Lorsque l'utilisateur commence à taper, une liste de suggestions apparaît, fournissant une interface interactive pour sélectionner une ville.

Pour éviter d'effectuer une requête pour chaque frappe de l'utilisateur, nous utilisons la fonction debounce de la bibliothèque lodash. Cette fonction retarde l'exécution de la fonction de recherche jusqu'à ce que l'utilisateur ait arrêté de taper pendant 300 millisecondes. Cette optimisation permet de réduire le nombre de requêtes inutiles, améliorant ainsi les performances de l'application.

La fonction de recherche elle-même effectue une requête axios à une API Elasticsearch. Cette requête renvoie une liste de villes qui correspondent à la requête de l'utilisateur. Les résultats de la recherche sont ensuite affichés dans une superposition sous la barre de recherche. L'utilisateur peut alors sélectionner une ville à partir de ces résultats, ce qui améliore l'interaction et la précision de la sélection.

La superposition des résultats de recherche est contrôlée par l'état de la requête de l'utilisateur. Elle apparaît dès que l'utilisateur commence à taper et se cache lorsque l'utilisateur clique en dehors de la barre de recherche.

En conclusion, le composant "Search" est un exemple bien pensé d'un composant de recherche optimisé et orienté utilisateur. L'utilisation d'Elasticsearch pour la recherche assure une rapidité et une pertinence des résultats, tandis que l'application de la fonction debounce garantit une interaction fluide et optimisée. Enfin, la superposition des résultats de recherche offre une manière intuitive de présenter des suggestions, améliorant ainsi l'expérience utilisateur dans son ensemble.

2.3.4_ COMPOSANT HOURLYFORECAST ET DAILYFORECAST :

Ces deux composants fournissent respectivement des prévisions météorologiques détaillées pour les heures et les jours à venir.

HourlyForecast affiche des prévisions météorologiques heure par heure pour un jour donné. Les données prévues sont passées au composant en tant que prop hourlyForecast. L'utilisateur a également la possibilité de choisir entre Celsius et Fahrenheit via la prop isCelsius.

Lors du rendu, chaque élément de la liste hourlyForecast est transformé en une représentation visuelle où l'heure et la condition météorologique sont affichées. Pour chaque prévision, l'heure est extraite et formatée en une chaîne à deux chiffres (par exemple, '05h'). Ensuite, la température est affichée, soit en Celsius soit en Fahrenheit, en fonction de la valeur de isCelsius.

Le second composant est **DailyForecast**. Il est conçu pour afficher les prévisions météorologiques quotidiennes pour une semaine, basées sur le nom d'une ville passée en prop cityName.

Ce composant est particulier parce qu'il utilise la bibliothèque Papa pour lire des données depuis des fichiers CSV. Ces fichiers CSV contiennent des prévisions de température maximale, moyenne et minimale obtenues grâce à notre modèle de machine learning conçu pour cette cause.

Dès que le cityName change, le composant déclenche une série d'appels pour lire ces fichiers CSV. Une fois les données lues, elles sont fusionnées pour créer un tableau de prévisions journalières. Ce tableau est ensuite filtré pour ne montrer que les prévisions des 7 prochains jours.

Chaque élément de la prévision affiche la date, ainsi que les températures maximale, moyenne et minimale. Les températures sont arrondies à l'entier le plus proche pour une meilleure lisibilité.

En conclusion, DailyForecast est un composant complexe qui combine plusieurs sources de données pour fournir une vue complète des prévisions météorologiques pour une semaine, en se basant sur le nom de la ville.

2.3.5_ COMPOSANT MAXMINTEMPERATURE :

Le composant **MaxMinTemperature** est conçu pour présenter les températures maximales et minimales prévues pour la journée à partir d'une liste de prévisions météorologiques fournies pour différentes heures.

Ce composant prend en compte deux propriétés : hourlyForecast, qui est un tableau d'objets représentant les prévisions météorologiques pour différentes heures, et isCelsius, un booléen qui détermine si les températures doivent être affichées en Celsius ou en Fahrenheit.

Au début, deux variables maxTemperature et minTemperature sont initialisées avec des valeurs extrêmes (-Infinity pour max et Infinity pour min) pour s'assurer que toute température dans la liste des prévisions sera soit plus grande, soit plus petite. Ensuite, une boucle parcourt chaque objet de prévision dans hourlyForecast. Pour chaque objet, elle extrait la température (en Celsius ou Fahrenheit en fonction de isCelsius) et la compare avec les valeurs actuelles de maxTemperature et minTemperature. Si une température est trouvée qui est plus élevée ou plus basse que les valeurs actuelles, celles-ci sont mises à jour.

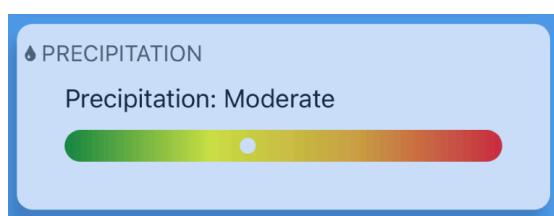
2.3.6_ COMPOSANTS DE DÉTAILS MÉTÉOROLOGIQUES :

• Precipitation :

Le composant **Precipitation** est destiné à représenter visuellement le niveau de précipitation à l'aide d'une barre colorée, tout en affichant également un qualificatif pour la précipitation basé sur un certain seuil.

Au cœur de ce composant, il y a une fonction appelée *interpretPrecipitation* qui évalue le niveau de précipitation en fonction d'une valeur d'entrée, probablement exprimée en millimètres. Quand cette valeur est supérieure à 0,7, la précipitation est qualifiée d'"Extrême". Si elle est supérieure à 0,5, elle est considérée comme "Forte". Une valeur supérieure à 0,3 donne une précipitation "Modérée", et toute valeur en dessous est simplement qualifiée de "Faible".

Le composant utilise cette interprétation pour afficher un message décrivant le niveau de précipitation. De plus, il visualise cette information à l'aide d'une barre de couleur. Cette barre utilise un gradient allant du vert (faibles précipitations) au rouge (précipitations élevées). Le niveau actuel de précipitation est indiqué sur cette barre par un marqueur, qui est un petit cercle blanc positionné en fonction de la valeur des précipitations.



Rendu du composant Precipitation

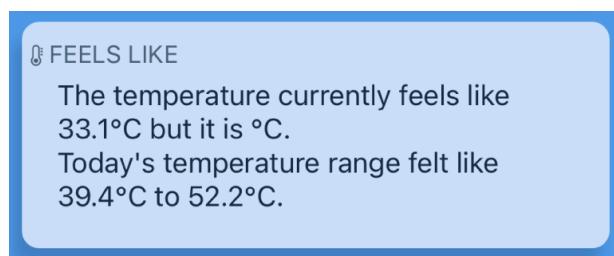
- **FeelsLike :**

Le composant **FeelsLike** est conçu pour afficher à l'utilisateur comment la température est perçue, par rapport à la température réelle mesurée. Cela est souvent différent à cause de facteurs tels que le vent ou l'humidité.

Au cœur de ce composant, nous utilisons les prévisions pour extraire une liste de températures ressenties pour chaque heure. Cette extraction est basée sur le choix de l'utilisateur : s'il préfère les degrés Celsius, alors nous utilisons les valeurs feelslike_c, sinon nous utilisons les valeurs feelslike_f.

De cette liste, nous déterminons la température ressentie la plus élevée (maxFeelsLike) et la température ressentie la plus basse (minFeelsLike) en utilisant les fonctions Math.max et Math.min. Ces valeurs nous donnent une idée de l'amplitude thermique perçue au cours de la journée.

Le rendu visuel du composant informe l'utilisateur de la température actuelle ressentie par rapport à la température réelle mesurée et donne une perspective sur l'amplitude des températures ressenties au cours de la journée.



Rendu du composant feelsLike

- **Wind :**

Le composant **Wind** est un élément graphique conçu pour représenter visuellement la direction et la vitesse du vent. Il se présente sous forme d'une boussole circulaire, avec des indicateurs pour les directions principales et secondaires du vent, ainsi qu'une flèche pointant dans la direction du vent actuel.

Tout d'abord, nous avons un tableau des directions possibles du vent, allant de 'N' pour le nord à 'NNW' pour le nord-nord-ouest. Il y a aussi un tableau pour les principales directions, qui ne contient que 'N', 'E', 'S' et 'W'. En fonction de la direction du vent passée en prop, on détermine l'index de cette direction dans notre tableau. Cet index nous donne ensuite l'angle auquel la flèche doit pointer : chaque direction est espacée de 22,5 degrés.



Rendu du composant wind

- **UVIndex :**

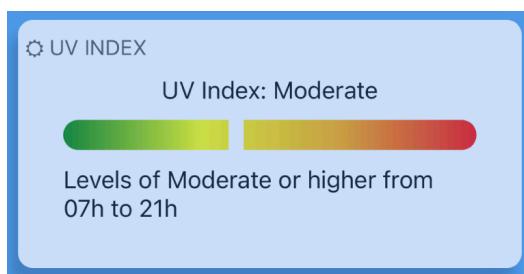
Le composant **UVIndex** est conçu pour afficher visuellement l'indice UV actuel et offrir un aperçu de son évolution au cours de la journée. Cet indice UV, qui mesure la puissance des rayons ultraviolets du soleil, est essentiel pour comprendre la capacité du soleil à causer des dommages à la peau et aux yeux. Pour aider les utilisateurs à interpréter l'indice, une fonction, `interpretUVIndex`, a été créée. Elle prend cet indice comme entrée et retourne une description, qui peut être 'Extreme', 'High', 'Moderate', ou 'Low'.

Par ailleurs, pour afficher l'indice UV des prévisions horaires sous un format facilement lisible, la fonction `formatHour` a été conçue. Elle transforme une chaîne temporelle, par exemple "2023-07-23 16:45:00", en une forme simplifiée comme "16h".

Quant à l'affichage visuel, le composant présente une barre de gradient, colorée du vert au rouge, reflétant la gamme de l'indice UV du plus faible au plus élevé. La position du marqueur sur cette barre est déterminée par la fonction `calculateMarkerPosition`, qui se base sur une valeur maximale présumée pour l'indice UV, fixée à 10 ici.

Le composant fournit également un aperçu des prévisions de l'indice UV. Si, à un moment donné de la journée, l'indice est prévu pour atteindre ou dépasser le niveau 'Moderate', un message indiquera à l'utilisateur l'intervalle de temps pendant lequel ces niveaux élevés sont attendus.

Finalement, le style visuel de ce composant a été soigneusement conçu pour assurer une expérience utilisateur optimale. Les motifs de style courants de React Native, tels que l'utilisation de vues positionnées de manière absolue ou relative, sont exploités pour superposer des éléments, comme le marqueur, sur la barre de gradient. Des tailles de police distinctes et des espacements appropriés garantissent également que l'information est présentée de manière claire et esthétiquement plaisante.



Rendu du composant UVIndex

- **Humidity :**

Le composant **Humidity** est un composant créé pour afficher clairement à l'utilisateur des informations sur l'humidité actuelle ainsi que des prévisions horaires sur l'humidité. L'objectif principal est d'aider l'utilisateur à comprendre l'humidité actuelle et comment elle pourrait évoluer au cours de la journée.

Pour formater l'affichage de l'heure, une fonction `formatHour` est utilisée. Cette fonction extrait l'heure d'une chaîne de caractères représentant le temps et la formate de manière à afficher uniquement l'heure suivie du symbole "h".

Ensuite, la fonction `calculateAverageHumidity` est conçue pour calculer l'humidité moyenne basée sur un ensemble de prévisions horaires. Elle utilise la méthode `reduce` pour sommer toutes les valeurs d'humidité et divise ensuite par le nombre total d'entrées pour obtenir la moyenne. Le résultat est arrondi pour obtenir un chiffre entier.

Le composant utilise principalement les propriétés `currentHumidity`, `hourlyForecast` et `currentHour`. En se basant sur la valeur de `currentHumidity`, le niveau d'humidité est catégorisé en trois niveaux : 'Low', 'Moderate' et 'High'.

De plus, des prévisions horaires sont transformées en niveaux d'humidité pour chaque heure en utilisant la méthode `map`. Ces niveaux d'humidité horaires sont ensuite filtrés pour identifier les périodes où l'humidité est élevée.

Un message est généré pour informer l'utilisateur de l'humidité actuelle. Si l'humidité est prévue comme élevée pendant certaines heures, le message sera mis à jour pour refléter cette information. Par exemple, si l'humidité est prévue comme élevée de 10h à 15h, le message affichera "L'humidité sera élevée de 10h à 15h".

De plus, un autre message informe l'utilisateur de l'humidité moyenne prévue pour la journée.



Rendu du composant Humidity

● **AirQuality:**

Le composant **AirQuality** est un composant React conçu pour présenter de manière claire et informative la qualité de l'air à un utilisateur. Il utilise des données sur la qualité de l'air pour fournir des informations et des conseils sur les précautions à prendre en fonction de la qualité actuelle de l'air.

Pour donner un sens à ces données, une fonction d'interprétation, nommée `interpretAirQuality`, est utilisée. Cette fonction prend un indice de qualité de l'air comme entrée et renvoie des informations

détaillées sur la qualité et les conseils associés. L'indice est basé sur le système d'évaluation de l'US EPA (Environmental Protection Agency des États-Unis), et il catégorise la qualité de l'air en six niveaux : de "Good" (Bon) à "Hazardous" (Dangereux). Chaque niveau est accompagné d'un conseil pour aider les utilisateurs à comprendre les précautions à prendre en fonction de la qualité de l'air actuelle.

Esthétiquement, le composant utilise un ensemble de couleurs définis pour représenter visuellement chaque niveau de qualité de l'air. Ces couleurs vont du vert pour "Good" à des teintes plus sombres de rouge et de violet pour les niveaux "Very Unhealthy" et "Hazardous". La qualité de l'air et le conseil associé sont affichés à l'intérieur d'un dégradé de couleurs, créé à l'aide de LinearGradient, qui est adapté à la qualité de l'air actuelle.

Si aucune donnée sur la qualité de l'air n'est fournie au composant, il retournera simplement null, ce qui signifie qu'aucun rendu ne sera effectué, offrant ainsi une expérience utilisateur sans encombre.



**Rendu du composant
AirQuality**



**Rendu du composant
AirQuality**

● Pressure :

Le composant **Pressure** est dédié à la représentation graphique de la pression atmosphérique sous forme de jauge analogique en utilisant le format SVG. Cette jauge est conçue pour présenter une lecture semi-circulaire, avec une aiguille pivotant autour d'un centre pour indiquer la pression actuelle.

Le rayon de la jauge est défini par la variable radius, tandis que la strokeWidth spécifie la largeur du trait utilisé dans le graphique. Le diameter, qui est simplement le double du rayon, donne la dimension complète de la jauge. Pour la représentation des pressions, le composant utilise une plage définie par minPressure et maxPressure. La variable pressureRange est calculée pour déterminer la différence totale entre ces deux valeurs. Quant à la variable pressureAngle, elle est essentielle pour positionner l'aiguille à un angle spécifique correspondant à la valeur de la pression actuelle.

Pour créer une échelle graduée sur la jauge, le composant génère un tableau allScale. Ce tableau contient toutes les valeurs de pression possibles entre le minimum et le maximum. Chaque valeur de cette échelle est ensuite utilisée pour dessiner des marques sur la jauge.



**Rendu du composant
Pressure**

2.4. UN MODELE DE MACHINE LEARNING POUR DES PREDICTIONS DE LA TEMPÉRATURE OPTIMALE :

2.4.1. DONNÉES D'ENTRAINEMENT :

Nous avons opté pour l'utilisation d'un ensemble de données particulièrement précieux, hébergé sur www.data.gouv.fr. Ce portail est une initiative de l'administration française visant à promouvoir l'ouverture des données publiques, en conformité avec une directive de l'Union européenne de 2003 qui favorise l'accès aux informations détenues par les institutions publiques.

Cet ensemble de données, que nous avons sélectionné, présente plusieurs avantages essentiels pour l'accomplissement de notre projet :

Richesse des données : Cet ensemble de données couvre l'ensemble des départements français sur une durée de plus de cinq ans, de janvier 2018 à nos jours. Cette volumineuse quantité d'information nous permet d'identifier des tendances, d'effectuer des comparaisons entre les départements et de générer des prévisions fiables.

Actualisation constante : Les informations sont régulièrement mises à jour, nous assurant ainsi un accès aux données les plus récentes, essentielles pour une analyse précise et pertinente.

Accessibilité des données : Les données sont présentées dans un format compréhensible et facilement exploitable. Leur intégration dans un notebook Jupyter pour effectuer des analyses et créer des visualisations est donc facilitée.

Fiabilité : Dernier point mais non des moindres, ces données proviennent d'une source gouvernementale, gage de leur fiabilité, précision et authenticité.

	date_obs	code_insee_departement	departement	tmin	tmax	tmoy
189787	2023-05-31	48	Lozère	9.80	19.62	14.71
189788	2023-05-31	49	Maine-et-Loire	11.70	26.52	19.11
189789	2023-05-31	59	Nord	11.95	20.50	16.23
189790	2023-05-31	64	Pyrénées-Atlantiques	16.77	22.02	19.39
189791	2023-05-31	67	Bas-Rhin	12.50	26.60	19.55

Echantillon des données

2.4.2 CHOIX DU MODÈLE :

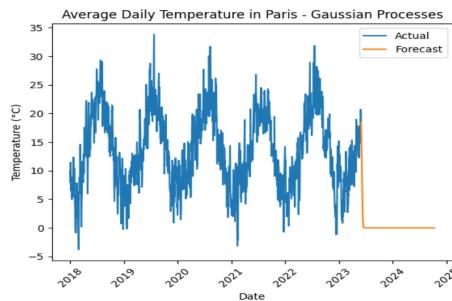
Nous avons testé plusieurs modèles statistiques et d'apprentissage automatique dans notre projet de prédiction météorologique en France. Chaque modèle a ses propres avantages et inconvénients.

2.4.2.1 CNN :

Les CNN sont principalement utilisés dans le traitement des images, où ils excellent à identifier les dépendances spatiales et temporelles grâce à l'application de filtres. Cependant, l'application de ces réseaux à la prédiction météorologique peut s'avérer moins efficace, particulièrement lorsque les données sont centrées uniquement sur les températures quotidiennes d'un emplacement précis. Un inconvénient majeur des CNN est qu'ils nécessitent une grande quantité de données pour l'apprentissage, limitant ainsi leur utilité dans un contexte de données limitées.

```
kernel = GPy.kern.RBF(input_dim=1)
model_gp = GPy.models.GPRegression(X_gp, y_gp.reshape(-1, 1), kernel)
model_gp.optimize()
```

```
forecast_gp = model_gp.predict(np.arange(len(new_column), len(new_column) + 500).reshape(-1, 1))
forecast_df_gp = pd.DataFrame({'ds': pd.date_range(new_column['ds'].iloc[-1], periods=500, freq='D'),
                               'y': forecast_gp[0].flatten()})
```



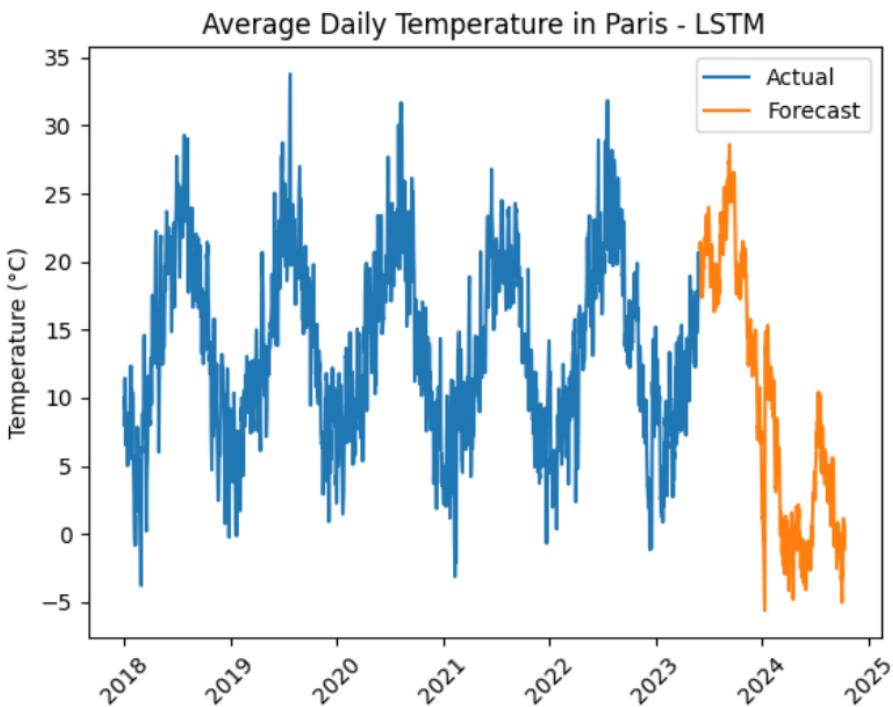
2.4.2.2 LSTM :

Les LSTM sont des réseaux de neurones récurrents capables de se souvenir et d'apprendre à partir d'informations sur de longues périodes. Pour cette raison, ils sont souvent utilisés pour des tâches impliquant des prédictions séquentielles, comme la prédiction météorologique. Néanmoins, leur complexité inhérente et la nécessité d'un temps d'apprentissage plus long avec des séquences plus longues peuvent se révéler problématiques.

```
def train_lstm(X, y):
    model_lstm = Sequential()
    model_lstm.add(LSTM(256, return_sequences=True, input_shape=(1, 1)))
    model_lstm.add(LSTM(128, return_sequences=True))
    model_lstm.add(LSTM(64, return_sequences=False))

    model_lstm.add(Dense(2))
    model_lstm.compile(loss='mean_squared_error', optimizer='adam')
    model_lstm.fit(X, y, epochs=75, batch_size=1, verbose=2)
    return model_lstm
```

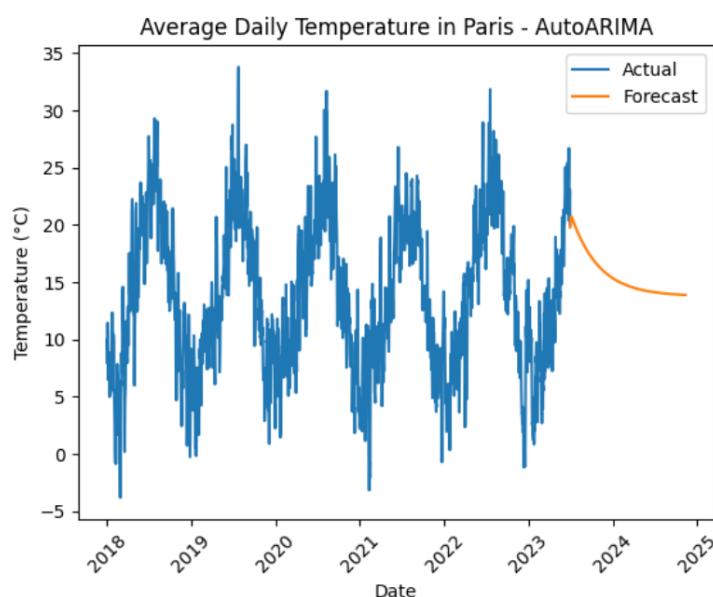
```
results = {}
for city, code in cities.items():
    city_df = df[df['code_insee_departement'] == code]
    new_column = preprocess_data(city_df)
    X_lstm = new_column['y'].values.reshape(-1, 1, 1)
    y_lstm = new_column['y'].values.reshape(-1, 1)
    model_lstm = train_lstm(X_lstm, y_lstm)
    forecast_steps = 500
    forecast_lstm_values = forecast_lstm(model_lstm, X_lstm[-1:], forecast_steps)
    forecast_df_lstm = pd.DataFrame({
        'ds': pd.date_range(new_column['ds'].iloc[-1], periods=forecast_steps, freq='D'),
        'y': forecast_lstm_values[1:] # Removed scaler.inverse_transform
    })
    results[city] = (new_column, forecast_df_lstm)
```



2.4.2.3_PMDARIMA :

pmdarima est une implémentation en Python de la méthode de prévision ARIMA (AutoRegressive Integrated Moving Average). ARIMA est un modèle statistique largement utilisé pour l'analyse de séries temporelles. Un de ses principaux inconvénients est son hypothèse sous-jacente que les séries temporelles sont stationnaires, ce qui n'est pas toujours le cas avec les données météorologiques.

```
model = auto_arima(new_column['y'], seasonal=True, trace=True)
```

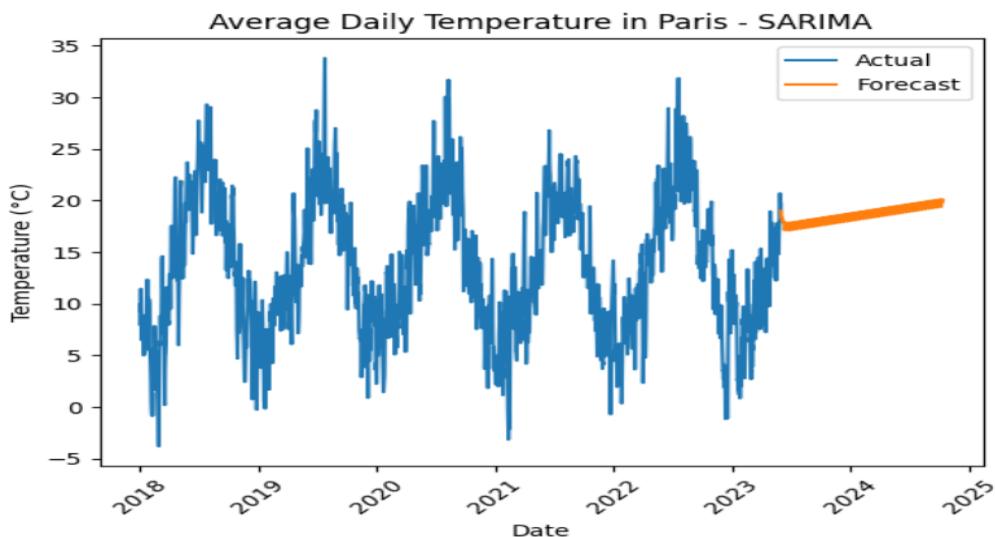


2.4.2.4_SARIMA:

Le modèle SARIMA est une extension du modèle ARIMA qui intègre la saisonnalité des données. Cette caractéristique en fait un choix plus approprié pour notre projet. Cependant, tout comme l'ARIMA, il exige que les séries temporelles soient stationnaires et peut être difficile à paramétriser correctement.

```
model_sarima = SARIMAX(new_column['y'], order=(1, 1, 1), seasonal_order=(1, 1, 1, 12))
model_sarima_fit = model_sarima.fit()

forecast_sarima = model_sarima_fit.predict(start=len(new_column['y']),
                                            end=len(new_column['y']) + 499,
                                            dynamic=True)
forecast_df_sarima = pd.DataFrame({'ds': pd.date_range(new_column['ds'].iloc[-1], periods=500, freq='D'),
                                    'y': forecast_sarima})
```



2.4.2.5_NEURALPROPHET:

NeuralProphet est une version améliorée de la bibliothèque de prévision Prophet de Facebook. Il combine les réseaux neuronaux avec des méthodes statistiques traditionnelles pour la prévision temporelle. Cependant, il peut nécessiter plus de temps pour l'apprentissage et l'ajustement des paramètres que d'autres méthodes.

```
# Loop over each department
for dept_code in department_codes:
    # Filter data for the current department
    df_dept = df[df['code_insee_department'] == dept_code]

    # Loop over each variable
    for var in ['tmax', 'tave', 'tmin']:
        # Aggregate by date and compute the mean of 'tmax', max of 'tmax', and min of 'tmin'
        if var == 'tmax':
            df_var = df_dept.groupby('date_obs')[var].mean().reset_index()
        else:
            df_var = df_dept.groupby('date_obs')[var].max().reset_index()
        elif var == 'tmin':
            df_var = df_dept.groupby('date_obs')[var].min().reset_index()

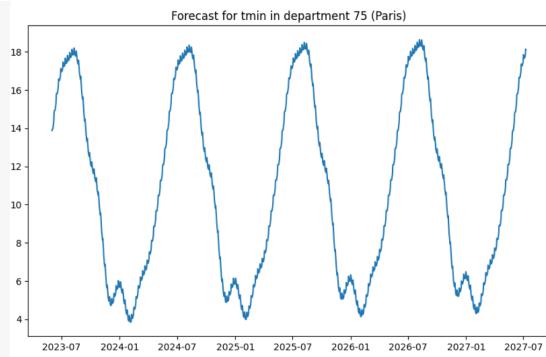
        # Prepare data for model fitting
        new_column = df_var[['date_obs', var]]
        new_column.columns = ['ds', 'y']

        # Initialise and fit the model
        n = NeuralProphet()
        model = n.fit(new_column, freq='D', epochs=800)

        # Store the model
        models[var][dept_code] = model

        # Make Forecast
        future = n.make_future_dataframe(new_column, periods=1500)
        forecast = n.predict(future)

        # Store the forecast
        forecasts[var][dept_code] = forecast
```



2.4.2.6_ GBM :

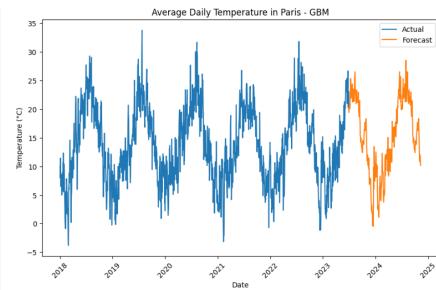
GBM est un algorithme d'augmentation du gradient. Il crée un modèle prédictif en combinant les prédictions de plusieurs modèles plus simples, ce qui réduit les erreurs. Par rapport à XGBoost, le GBM est généralement plus rapide et nécessite moins de mémoire. C'est la raison pour laquelle il a donné de meilleurs résultats dans notre projet.

```
# Définition de la grille des hyperparamètres pour la recherche par validation croisée
param_grid = {'n_estimators': [100, 200, 300], 'learning_rate': [0.01, 0.1, 1], 'max_depth': [3, 4, 5]}

# Recherche par validation croisée pour trouver les meilleurs hyperparamètres
model_gbm = GradientBoostingRegressor(random_state=42)
grid_search = GridSearchCV(model_gbm, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)
best_params_ = grid_search.best_params_

# Entrainement du modèle avec les meilleurs hyperparamètres trouvés
model_gbm_best = GradientBoostingRegressor(**best_params_, random_state=42)
model_gbm_best.fit(X_train, y_train)

# Prédictions pour les dates futures et tracé des résultats
future_dates = pd.date_range(new_column['ds'].iloc[-1] + pd.DateOffset(days=1), periods=500, freq='D')
future_df = pd.DataFrame()
future_df['ds'] = future_dates
future_df['year'] = future_df['ds'].dt.year
future_df['month'] = future_df['ds'].dt.month
future_df['day_of_year'] = future_df['ds'].dt.dayofyear
forecast_gbm = model_gbm_best.predict(future_df[['year', 'month', 'day_of_year']])
forecast_df_gbm = pd.DataFrame({'ds': future_dates, 'y': forecast_gbm})
```



Chaque modèle a été testé en utilisant notre ensemble de données sur la température quotidienne en France afin d'évaluer ses performances. Nous avons finalement retenu le GBM pour sa robustesse, sa rapidité et sa capacité à produire des prédictions de haute précision pour nos données.

2.4.3_ PRÉDICTION DE TEMPÉRATURE AVEC GBM :

Notre script se focalise sur la prédiction des températures pour différentes villes en utilisant la méthode de régression par augmentation de gradient (GBM). Après avoir sélectionné les données pertinentes pour chaque ville à l'aide du code INSEE, nous traitons trois variables de température: moyenne, minimale et maximale. Pour chaque variable, nous calculons la moyenne quotidienne et préparons les données en renommant les colonnes en 'ds' et 'y'.

Des caractéristiques supplémentaires, comme l'année et le jour de la semaine, sont extraites de la date et ajoutées au dataframe. Avec une répartition de 80/20 pour l'apprentissage et le test, nous formons le modèle GBM, optimisant ses paramètres via une recherche sur grille basée sur la RMSE.

Après l'entraînement, le modèle prédit les températures pour des dates futures. Ces prédictions sont sauvegardées dans des fichiers CSV, prêts à être visualisés sur notre application principale.

```
model_gbm = GradientBoostingRegressor(random_state=42)
grid_search = GridSearchCV(model_gbm, param_grid, cv=5, scoring='neg_root_mean_squared_error')
grid_search.fit(X_train, y_train)
best_params_ = grid_search.best_params_
model_gbm_best = GradientBoostingRegressor(**best_params_, random_state=42)
model_gbm_best.fit(X_train, y_train)
```

```

future_dates = pd.date_range(new_column['ds'].iloc[-1] + pd.DateOffset(days=1), periods=500, freq='D')
future_df = pd.DataFrame()
future_df['ds'] = future_dates
future_df['year'] = future_df['ds'].dt.year
future_df['month'] = future_df['ds'].dt.month
future_df['day_of_year'] = future_df['ds'].dt.dayofyear
future_df['day_of_week'] = future_df['ds'].dt.dayofweek
forecast_gbm_best = model_gbm_best.predict(future_df[['year', 'month', 'day_of_year', 'day_of_week']])

```

3_ RÉSULTATS ET FONCTIONNALITÉS DE L'APPLICATION :

Notre application météorologique offre une expérience utilisateur immersive et complète, répondant aux besoins de quiconque souhaite se tenir informé des conditions météorologiques actuelles et futures. Conçue avec une approche centrée sur l'utilisateur, elle fournit des informations pertinentes et en temps réel, tout en étant esthétiquement agréable et facile à naviguer.

Dès l'ouverture de l'application, les utilisateurs sont accueillis par une barre de recherche avancée qui permet une découverte fluide des localités ou des régions dont on veut consulter la météo. Grâce à l'intégration de technologies de pointe, les suggestions apparaissent en temps réel, guidant les utilisateurs vers les résultats les plus pertinents.



Interface dès ouverture de l'application

La barre de recherche avancée permet une découverte fluide des localités ou des régions. Grâce à l'intégration de technologies de pointe, les suggestions apparaissent en temps réel, guidant les utilisateurs vers les résultats les plus pertinents.

Après le choix de la ville, les utilisateurs sont accueillis avec une vue d'ensemble des conditions météorologiques actuelles, y compris la température, l'humidité, les précipitations et bien plus encore, le tout présenté de manière visuellement attrayante. Un switch est également fourni pour alterner entre les degrés Celsius et Fahrenheit selon la préférence de l'utilisateur pour les valeurs de la température actuelle, les prédictions et le ressenti.



Température en Celsius

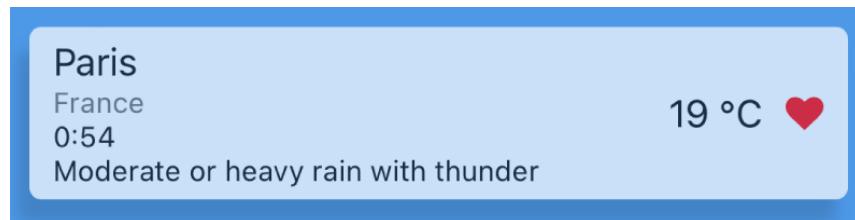


Température en Fahrenheit

L'application ne s'arrête pas simplement à la température. Des éléments tels que la pression atmosphérique, l'humidité et d'autres indicateurs importants sont présentés de manière compréhensible, permettant aux utilisateurs de comprendre pleinement les conditions actuelles.

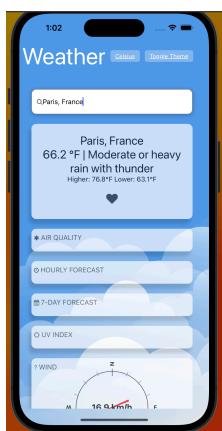
Nous permettons d'obtenir des informations précises sur les températures maximales et minimales prévues pour la journée et même dans les 7 jours à venir, vous permettant de vous habiller ou de planifier vos activités en conséquence.

Comprendre la météo de plusieurs endroits n'a jamais été aussi simple. Les utilisateurs peuvent sauvegarder leurs lieux préférés ou fréquemment consultés, permettant un accès rapide aux prévisions pour ces zones sans avoir à rechercher à chaque fois.

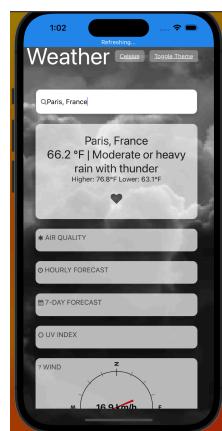


Villes favorites

Pour une expérience personnalisée, l'application permet également aux utilisateurs de basculer entre un thème sombre et un thème clair, en fonction de leurs préférences et des conditions d'éclairage.



Mode clair



Mode sombre

CONCLUSION :

Cette application météorologique est le fruit d'un travail acharné, de recherche approfondie et d'une passion pour offrir la meilleure expérience utilisateur possible. Les fonctionnalités qu'elle propose sont le reflet de cette détermination et de cet engagement envers l'excellence.

L'élaboration de ce projet nous a plongés au cœur de dilemmes technologiques. Face à un éventail de technologies et d'outils disponibles, nous avons été amenés à faire des choix déterminants, souvent au détriment d'autres options tout aussi prometteuses. Ces décisions technologiques, bien que difficiles, ont été essentielles pour façonner l'identité et la performance de notre application.

Chaque étape de développement a été marquée par des choix technologiques critiques. Par exemple, en choisissant d'adopter l'algorithme GBM (Gradient Boosting Machine) pour sa rapidité et son efficacité, nous avons délibérément écarté d'autres méthodes potentielles. De même, notre engagement envers des plateformes spécifiques et des APIs, comme WeatherAPI et Geonames, a nécessité de renoncer à d'autres alternatives peut-être plus conventionnelles.

Les premières étapes du projet, marquées par des divergences d'opinions, ont finalement abouti à une collaboration solide entre deux d'entre nous, renforçant davantage l'importance de ces choix technologiques. Chaque décision était une affirmation de notre vision et de notre engagement envers l'innovation.

Commencer ce projet a nécessité plus que de la simple détermination. Dès les premiers instants, confrontés à des divergences d'opinions et des défis de coordination, nous avons dû faire des choix cruciaux. Chaque décision prise a souvent signifié abandonner d'autres options potentiellement viables. Ces choix, loin d'être faciles, ont façonné la trajectoire de notre projet, et nous ont amenés à réaliser l'importance de la prise de décision éclairée et stratégique.

Aujourd'hui, nous sommes extrêmement fiers de ce que nous avons accompli. Notre application, bien que le fruit de compromis, de passion, d'engagement et de choix délibérés, est le reflet de notre passion, de notre expertise technique et de notre capacité à prendre des décisions éclairées face à la complexité. Elle symbolise notre détermination à réaliser une vision, malgré les obstacles. À tous ceux qui ont soutenu ce projet, votre confiance a été la pierre angulaire de cette réalisation. Avec détermination et discernement technologique, nous avons sculpté une solution qui se démarque par son unicité et sa performance.