**Vocal Weather** est une application permettant d'obtenir la météo par commande vocale. Cette solution a été conçue pour faciliter l'accès à l'information, notamment pour les personnes en situation de handicap.

Pour commencer, j'ai pris le temps d'analyser le sujet et de dresser une liste exhaustive de toutes les fonctionnalités potentielles. J'ai ensuite classé ces fonctionnalités en trois catégories :

- **Must have** : fonctionnalités essentielles
- **Nice to have** : améliorations facultatives
- **Dream feature** : fonctionnalités à implémenter ultérieurement, si les ressources le permettent

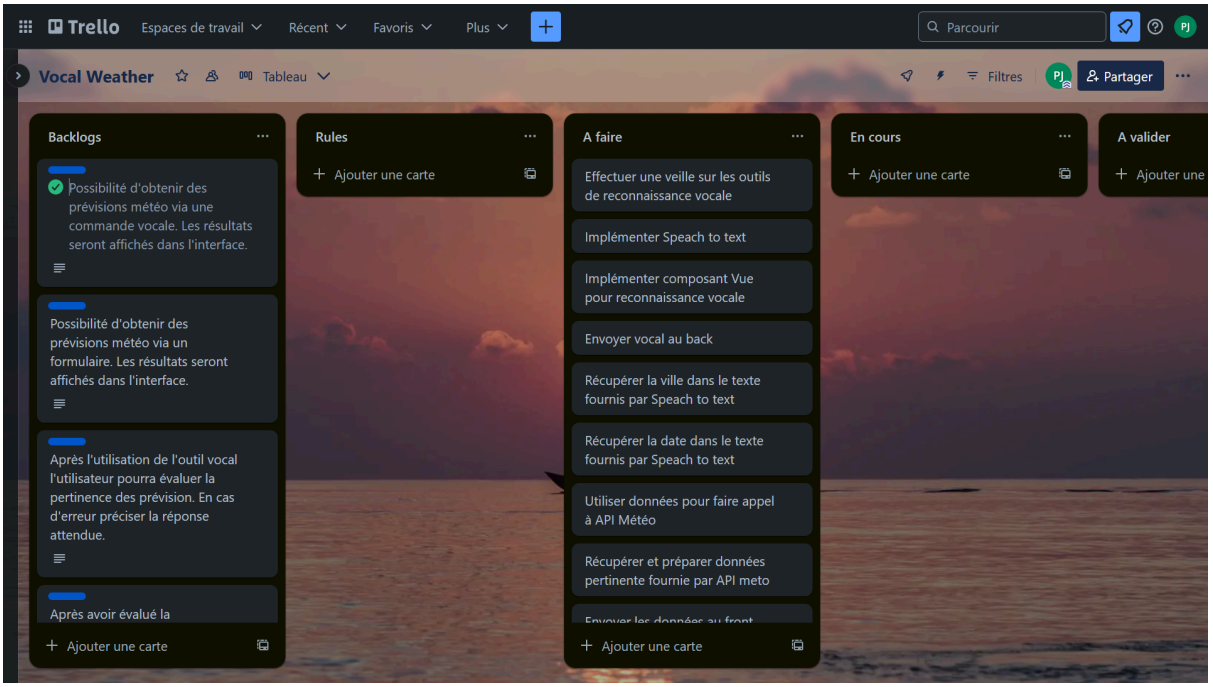

Cette étape m'a permis de déterminer la fonctionnalité principale sur laquelle me concentrer, tout en assurant la faisabilité dans les délais impartis.

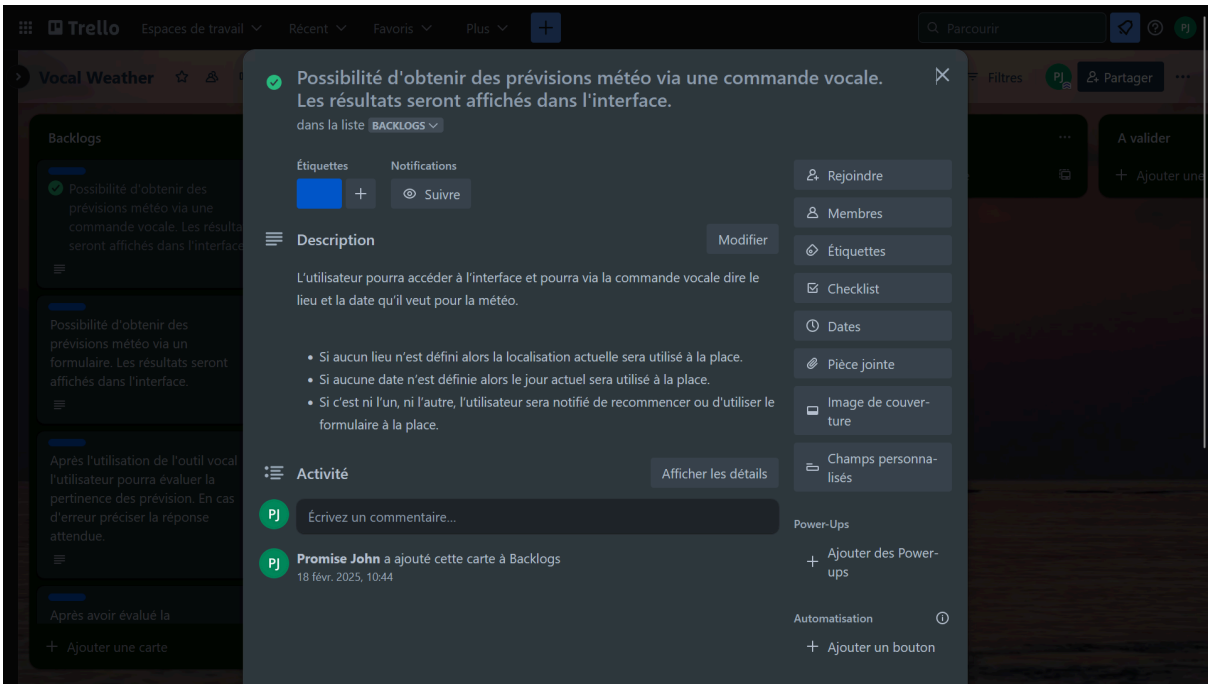
Une fois ce tri effectué, j'ai réalisé des diagrammes UML pour clarifier et documenter le fonctionnement de l'application. Deux types de diagrammes se sont révélés particulièrement utiles :

- **Diagrammes de cas d'utilisation** : Ils m'ont aidé à identifier les principaux acteurs de l'application et à définir les différentes actions qu'ils peuvent effectuer. J'ai ainsi pu avoir une vue d'ensemble des fonctionnalités, de leurs dépendances et de leur pertinence.
- **Diagrammes de séquence** : Ces diagrammes permettent de visualiser l'enchaînement logique des actions. Ils m'ont donné une idée claire de la manière dont les acteurs et le système interagissent et dans quel ordre.

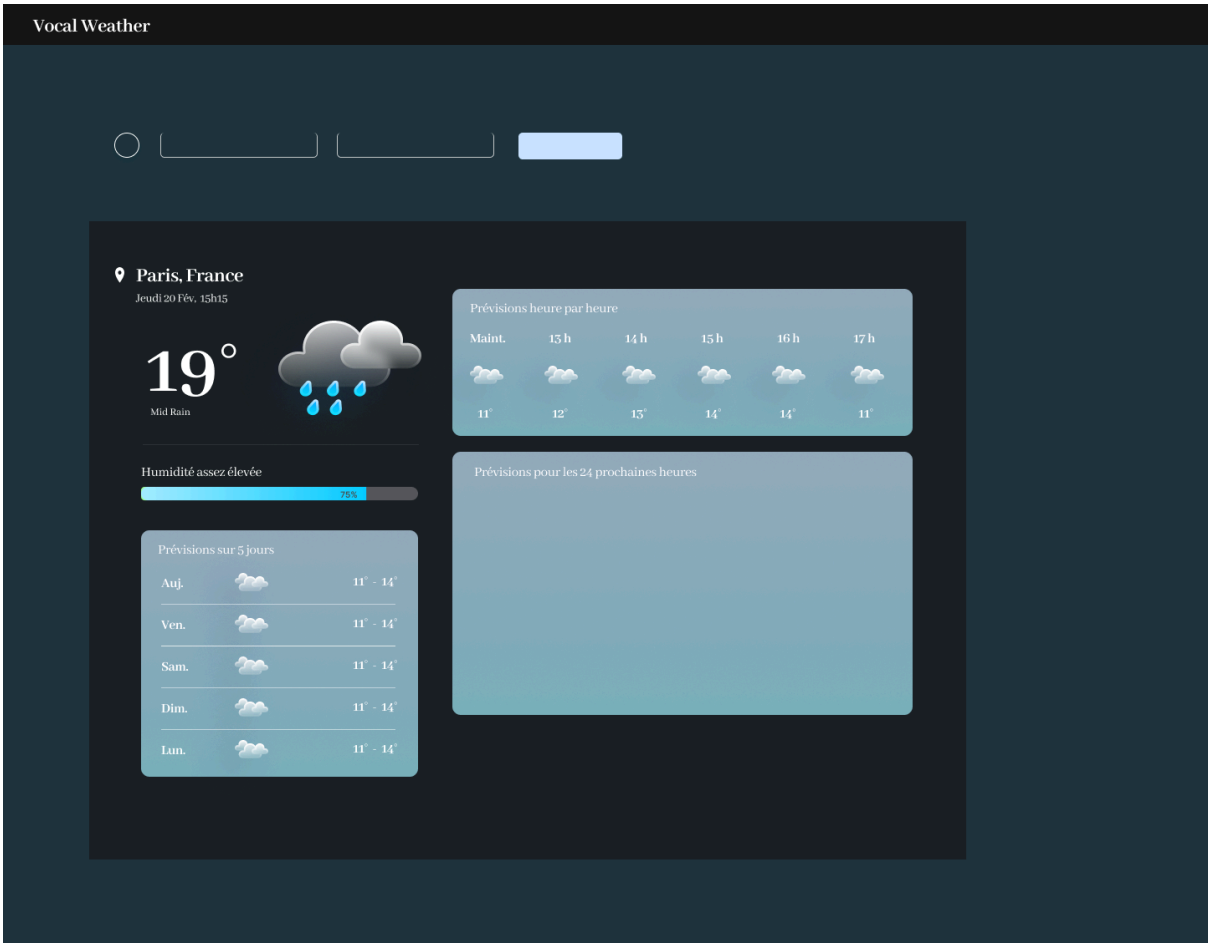
Les diagrammes UML m'ont permis de clarifier mes idées dès le départ, de visualiser le fonctionnement global de l'application et de faciliter la planification. Je savais précisément où j'allais et comment y parvenir.

Après cette étape de modélisation, j'ai utilisé Trello pour organiser et détailler chaque fonctionnalité (par exemple, "si l'événement X se produit, alors l'action Y sera déclenchée").



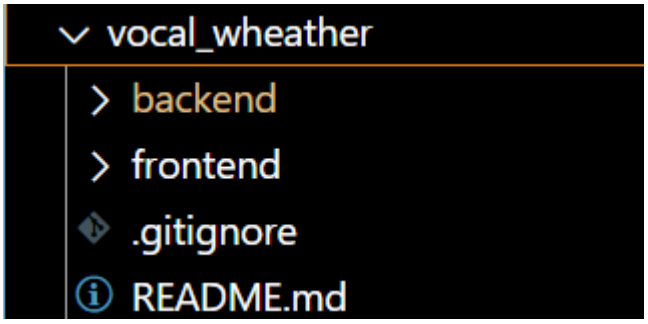

Parallèlement, j'ai commencé à réaliser des maquettes, d'abord sous forme de schémas simples, puis en les affinant.



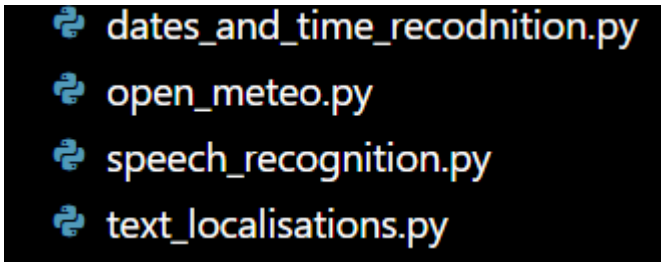

J'ai également mené une veille sur les données météorologiques disponibles via Open Meteo afin de déterminer quelles informations pouvaient être récupérées et comment les afficher. Grâce à ce va-et-vient entre conception et prise en main des API et bibliothèques, le processus a été itératif et fluide. J'ajustais au fur et à mesure mes maquettes et mes cartes Trello en fonction des nouveaux éléments découverts.

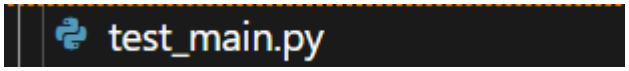
Lors de ma veille sur l'API Open Meteo, j'ai intégré **geopy**, qui permet d'obtenir la latitude et la longitude à partir d'une ville. Cela m'a amené à introduire **spaCy** pour extraire automatiquement le nom d'une ville depuis un texte, et finalement à intégrer la reconnaissance vocale Azure afin de convertir la parole en texte. Le code répondait à mes besoins, c'est pourquoi j'ai conçu des modules dédiés.

Mon projet se compose de deux parties : un backend et un frontend.

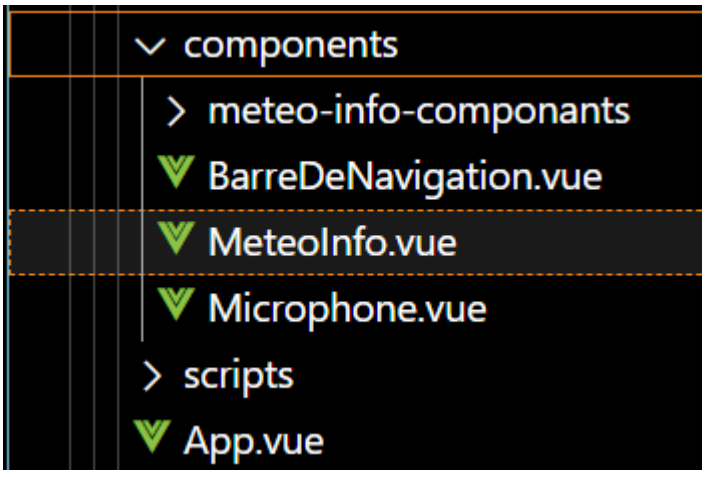


- **Backend**
  Le backend est divisé en deux parties :
  - Mes modules Python, qui gèrent les différentes fonctionnalités (reconnaissance vocale, extraction de données, etc.)
  - Mon serveur Flask (main.py) qui comporte une route pour recevoir le texte vocal et renvoyer les données météo, et une autre pour recevoir la localisation actuelle.



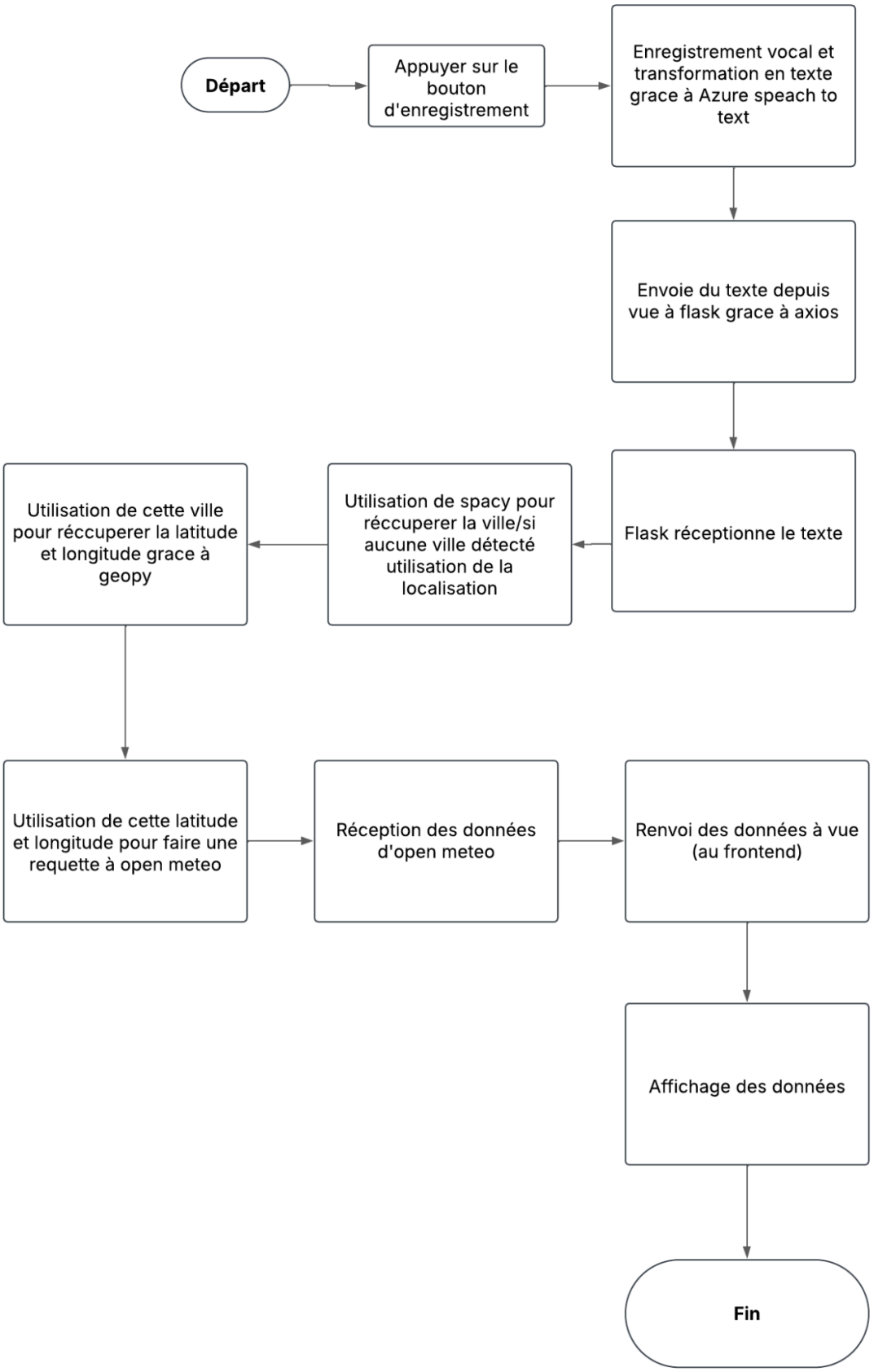




- **Frontend**
  Pour la partie frontend, j'ai choisi d'utiliser Vue.js, ce qui m'a permis de structurer mon interface en plusieurs composants.
  Initialement, la reconnaissance vocale se faisait via Python sur le backend. Cependant, afin de mieux contrôler l'accès au microphone et de faciliter l'autorisation d'utilisation dans le navigateur, j'ai déplacé la capture audio côté frontend en JavaScript. Le texte reconnu est ensuite envoyé au backend avec Axios.
  En cas d'absence de ville dans la commande vocale, la localisation est récupérée grâce à JavaScript. Le navigateur demande alors l'autorisation à l'utilisateur pour obtenir la latitude et la longitude, qui sont ensuite envoyées à Flask.




  ps : On peut retrouver ces composants dans **frontend/vue_front_wheather/src**

Disposant des données nécessaires grâce aux étapes précédentes et ayant un peu de temps supplémentaire, j'ai intégré un graphique en JavaScript pour illustrer l'évolution des températures au fil des heures, rendant ainsi l'application plus intuitive.


Départ
Appuyer sur le bouton d'enregistrement
Enregistrement vocal et transformation en texte grace à Azure speach to text
Envoie du texte depuis vue à flask grace à axios
Flask réceptionne le texte
Utilisation de spacy pour réccuperer la ville/si aucune ville détecté utilisation de la localisation
Utilisation de cette ville pour réccuperer la latitude et longitude grace à geopy
Utilisation de cette latitude et longitude pour faire une requette à open meteo
Réception des données d'open meteo
Renvoi des données à vue (au frontend)
Affichage des données
Fin