TP2 - Gestion des State + météo

HAI912I - Développement embarqué

Groupe n°8 :

COSSU Arnaud *21908322*
POINTEAU Gabrielle *21917975*
SAID Adam *21905365*

Table des matières

1 Choix de conception

1.1 Diagrammes de classes d'UML

L'idée du TP et de faire une application qui réunit différents aspects de gestion d'état pour un quiz et la météo. Pour cela, nous avons choisi de faire une seule et même application qui gère les trois cas.

Donc on a un Menu, qui permet d'accéder à 3 pages : la page de quiz cubit ; la page de quiz provider ; la page de météo.

On peut voir sur la figure 6 à la page 7 l'aspect général de toute l'application. Cependant, étant donné que tout le diagramme est trop grand est illisible on a décidé de le séparer en plusieurs parties. En voici donc une :

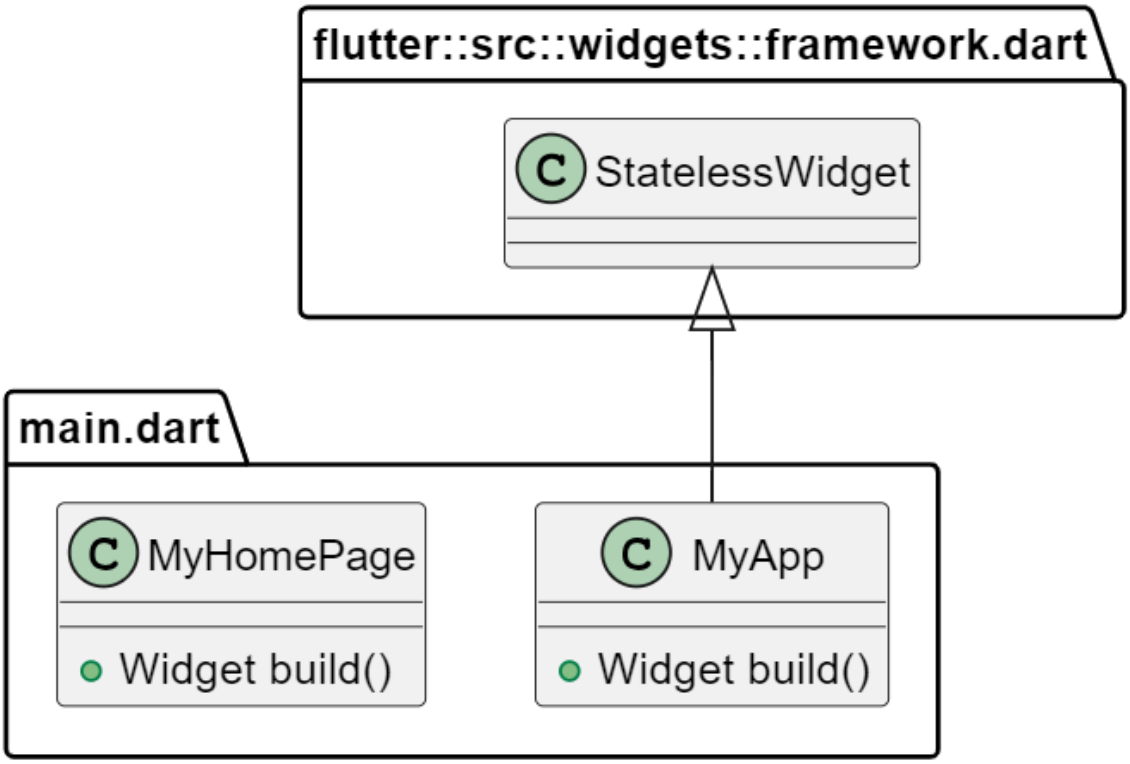

FIGURE 1 – Diagramme UML de dépendances des classes de main.dart

1.2 Architecture

Au niveau de l'architecture, l'application Flutter du TP2 est structurée à travers plusieurs fichiers principaux : `main.dart`, `quizz_cubit.dart`, `quizz_provider.dart`, `question_state.dart`, `resultat_quiz.dart`, `weather.dart`, et `weather_model.dart`.

1.2.1 `main.dart`

Commençons par le fichier `main.dart`. C'est la première page de l'application Flutter réalisé. Il définit la navigation entre les différentes pages. Le fichier inclut différents fichiers

de styles, images, couleurs et textes dans `styles.dart`, `colors.dart`, `text.dart` et `images.dart`.

Ensuite, au niveau des classes présentes dans `main.dart` :

* La classe `MyApp` est une classe `StatelessWidget` qui a une méthode build pour construire l'interface utilisateur. De plus, elle crée une instance de `MyHomePage` comme page d'accueil. Par ailleurs, elle gère la navigation entre les différentes pages de l'application. On y retrouve ainsi les 4 routes possibles :
  - `/` : route principale du menu
  - `/quizz_cubit` : route qui amène à la page sur le quiz utilisant Bloc/Cubit (cf. Exercice 2)
  - `/quizz_provider` : route qui amène à la page sur le quiz utilisant provider (cf. Exercice 1)
  - `/weather` : route pour consulter la météo (cf. Exercice 3)
* La classe `MyHomePage` est une classe `StatelessWidget` et contient la page d'accueil de l'application dont le menu principal. Ce menu est constitué de 3 boutons pour accéder aux pages de quiz réalisé avec Provider (`QuizzProviderPage`), au quiz utilisant Bloc/Cubit (`QuizzCubitPage`) et à la météo (`WeatherPage`).

1.2.2 `quizz_cubit.dart`

Une fois qu'on a appuyé sur le bouton de quiz cubit, on accède à une nouvelle page : `QuizzCubitPage`. Elle correspond à l'exercice 2 du TP et est dédiée à la gestion du quiz en utilisant le concept de Cubit.

Au niveau des classes on aura :

* La classe `QuizEvent` qui est une classe abstraite représentant les événements liés au quiz.
* La classe `NextQuestionEvent` est une classe concrète héritant de `QuizEvent` pour représenter le passage à la question suivante.
* La classe `QuizCubit` qui étend la classe `Cubit` pour gérer l'état du quiz, les questions, et le score.
* La classe `QuizzPageCubit` est une classe `StatelessWidget` pour la page du quiz en elle même avec la photo les boutons Vrai et Faux, utilisant `QuizCubit`.
* La classe `QuizzPageStateCubit` est une classe `StatelessWidget` pour l'état de la page du quiz en fonction de la question à laquelle on est, utilisant `QuizCubit`.

1.2.3 `quizz_provider.dart`

Le fichier `quizz_provider.dart` est dédié à la gestion du quiz en utilisant le concept de Provider. Il comprend des classes responsables de la gestion de l'état et de la logique du quiz.

* La classe `QuizProvider` qui est une classe étendant `ChangeNotifier` pour gérer l'état du quiz, les questions, et le score.
* La classe `QuizzPageProvider` qui est une classe `StatelessWidget` pour la page du quiz, utilisant `QuizProvider`.

* La classe `QuizzPageStateProvider` est une classe `StatelessWidget` pour l'état de la page du quiz, utilisant `QuizProvider`.

1.2.4 `question_state.dart`

Le fichier `question_state.dart` contient la définition de la classe `QuestionState`, représentant l'état d'une question du quiz avec des propriétés telles que le texte de la question, la correction de la réponse, et une décoration d'image pour la question.

1.2.5 `resultat_quizz.dart`

Une fois qu'on a terminé le quiz, on accède à une nouvelle page : `ResultatPage`.
Au niveau des classes on aura une classe :

* La classe `ResultatPage` est une classe StatelessWidget avec une méthode build pour construire l'interface utilisateur de la page de résultats. Elle affiche le score de l'utilisateur et un message en fonction de son score.

Pour les pages de quiz on pourra retrouver pour chaque cas (provider, cubit) les diagrammes UML figure 2 et 3 aux pages 3 et 4.

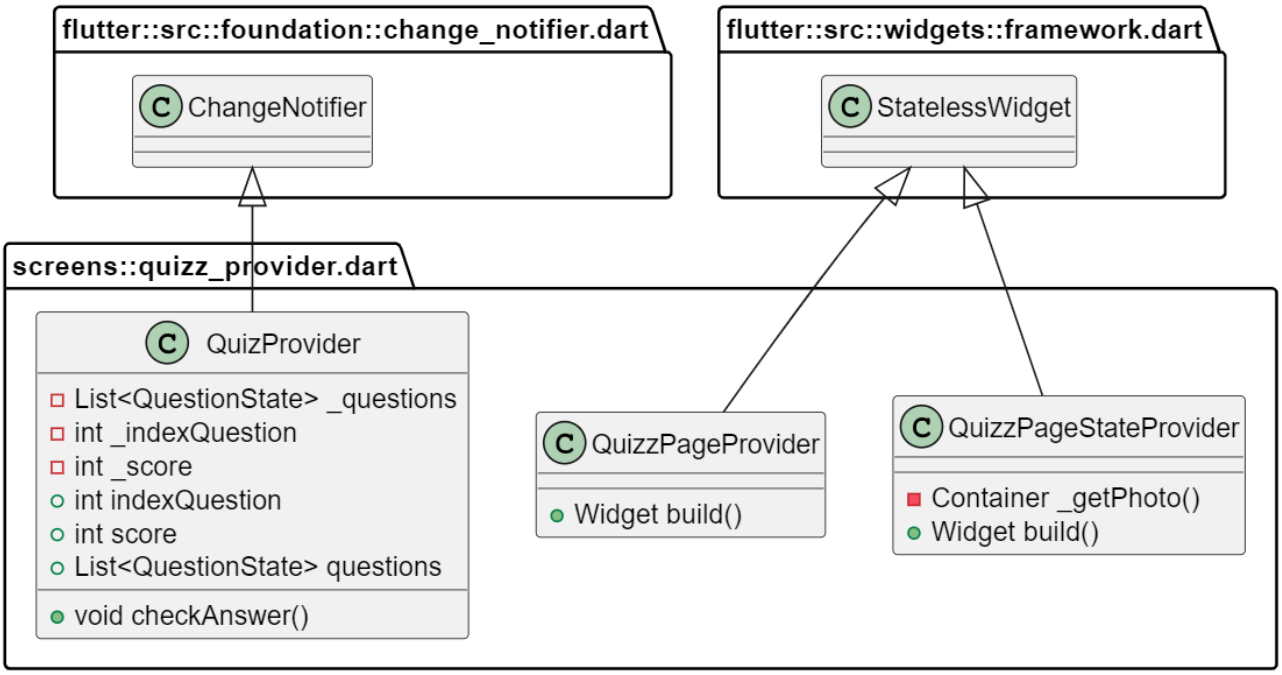

FIGURE 2 – Diagramme UML de dépendances des classes de quizz_provider.dart

1.2.6 `weather.dart`

Le fichier `weather.dart` gère l'affichage des informations météorologiques.

* La classe `Network` qui représente la connexion réseau pour obtenir des données météorologiques.

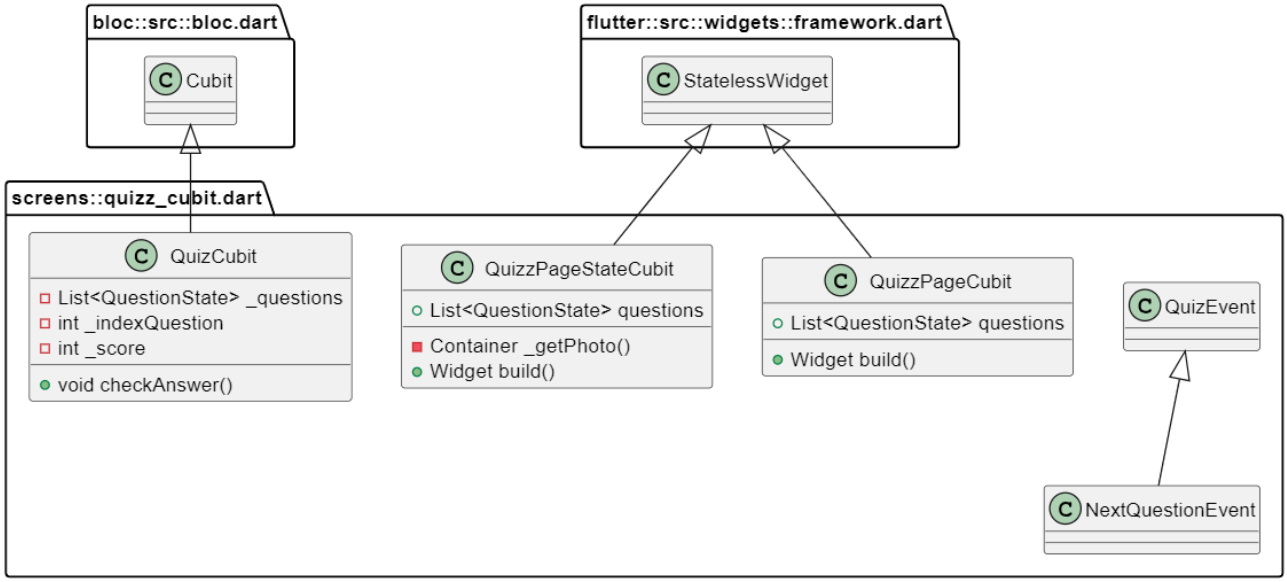

FIGURE 3 – Diagramme UML de dépendances des classes de quizz_cubit.dart

* La classe `WeatherPage` qui est une classe StatefulWidget représentant la page météo.

  On pourra ainsi voir la disposition des éléments de météo. On aura la ville et la date, puis la météo du jour avec l'icon associé. Et pour finir, on aura une prévision météo pour les 4 prochains jours.

La figure 4 à la page 6 représente donc le détail du fichier dans un diagramme UML représentant les dépendances de chaque classe.

Il ne sera pas visible les dépendances entre les fichiers comme `weather.dart` et `weather_model.dart` puisque inon cela fait un trop grand diagramme on a préféré séparer les deux.

1.2.7 `weather_model.dart`

Le fichier weather_model.dart contient les classes représentant la structure des données météorologiques.

On aura :

* La classe `WeatherModel` qui représente les données météorologiques générales.
* La classe `WeatherList` qui représente une liste de données météorologiques.
* Les classes `Main`, `WeatherData`, `Clouds`, `Wind`, `Sys`, `Rain`, `City`, `Coord` qui sont les classes représentant des composants spécifiques des données météorologiques.

On pourra ainsi retrouver le detail des éléments de chaque classe sur la figure 5 à la page 5.

L'architecture suit le modèle Flutter avec des classes `Stateless` et `StatefulWidget` pour gérer l'interface utilisateur et l'état de l'application.

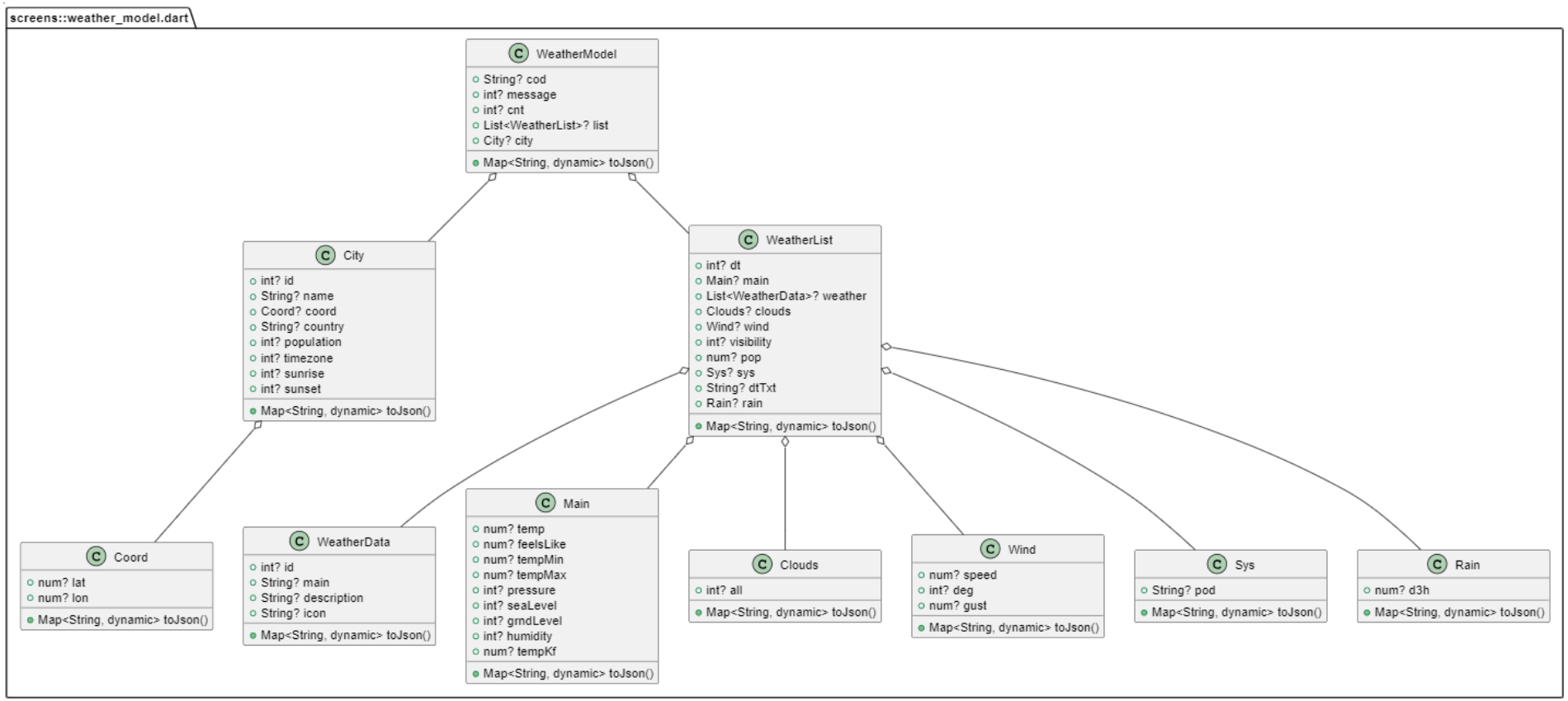

FIGURE 5 – Diagramme UML de dépendances des classes de weather_model.dart

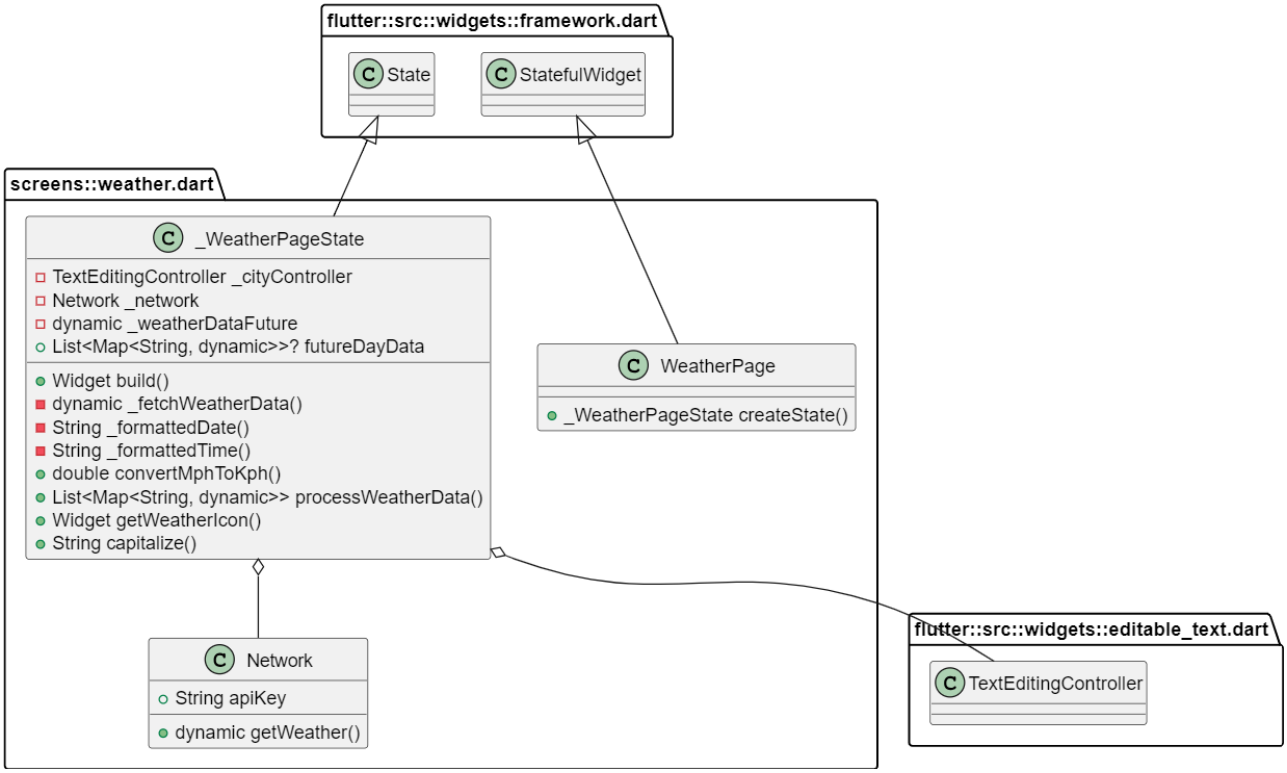

FIGURE 4 – Diagramme UML de dépendances des classes de weather.dart

1.3 Workflow

1. Lancement de l'Application : L'utilisateur lance l'application.
2. Affichage de la Page d'Accueil (`MyHomePage`) : La page d'accueil est affichée, présentant des options telles que le quiz et la météo.
3. Navigation vers la Page de `QuizzProvider` ou `QuizzCubit` : L'utilisateur clique sur l'option de quiz qu'il veut. La page `QuizzProvider` (`QuizzPageProvider`) ou `QuizzCubit` (`QuizzPageCubit`) est affichée.
4. Interaction avec `QuizzPage` (Cubit, Provider) : L'utilisateur répond aux questions du quiz. Les réponses sont gérées par `QuizzCubit` ou `QuizzProvider` pour mettre à jour l'état du quiz.
5. Affichage des Résultats (`ResultatPage`) après le `QuizzProvider` ou `QuizzCubit` : Une fois le quiz terminé, l'utilisateur est redirigé vers la page de résultats (`ResultatPage`). Les résultats sont basés sur les réponses fournies.
6. Navigation vers la Page `Météo` (`WeatherPage`) : L'utilisateur choisit de consulter la météo depuis la page d'accueil. La page `Météo`(`WeatherPage`) est affichée.
7. Interaction avec `WeatherPage` : La page `Météo` affiche les informations météorologiques à partir des données récupérées par `WeatherState`.
8. Récupération des Données Météo : Les données météorologiques sont récupérées et traitées par `WeatherState`, à partir de l'API fourni par le TP.
9. Fin de l'Application : L'utilisateur peut quitter l'application.

1.4 Diagramme de séquences

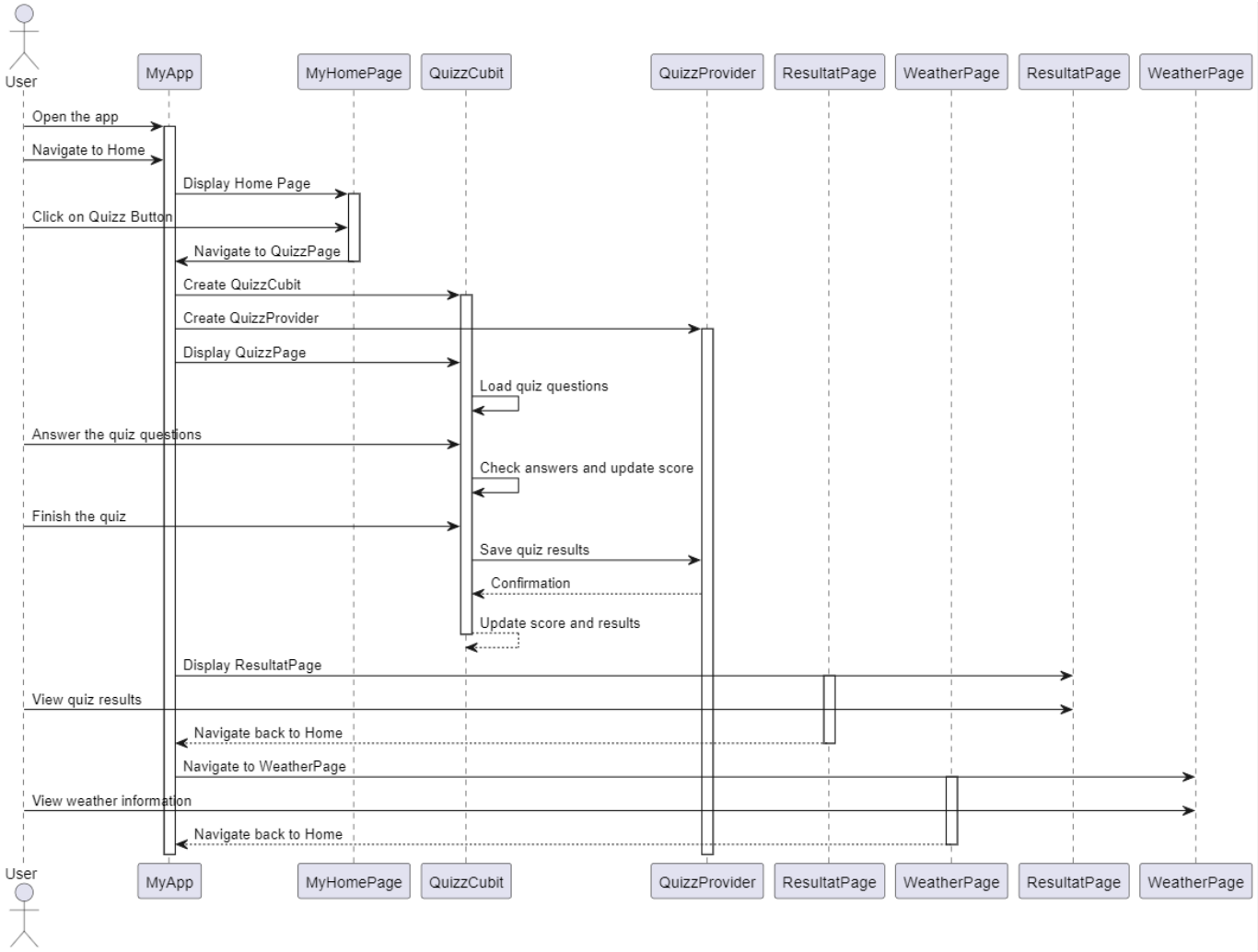

FIGURE 6 – Diagramme de séquence

2 Snapshot de notre application

On va pouvoir montrer le fil directeur de l'application à partir de snapshot de l'application.

On commence par le menu :

FIGURE 7 – Pages d'affichage de menu et de profil

Puis nous pouvons voir les snapshots de `QuizzProvider` et `QuizzCubit` mais ce sont les mêmes ainsi, nous mettrons 2 snapshot qui représenteront les 2 cas. De même pour la page de résultat.

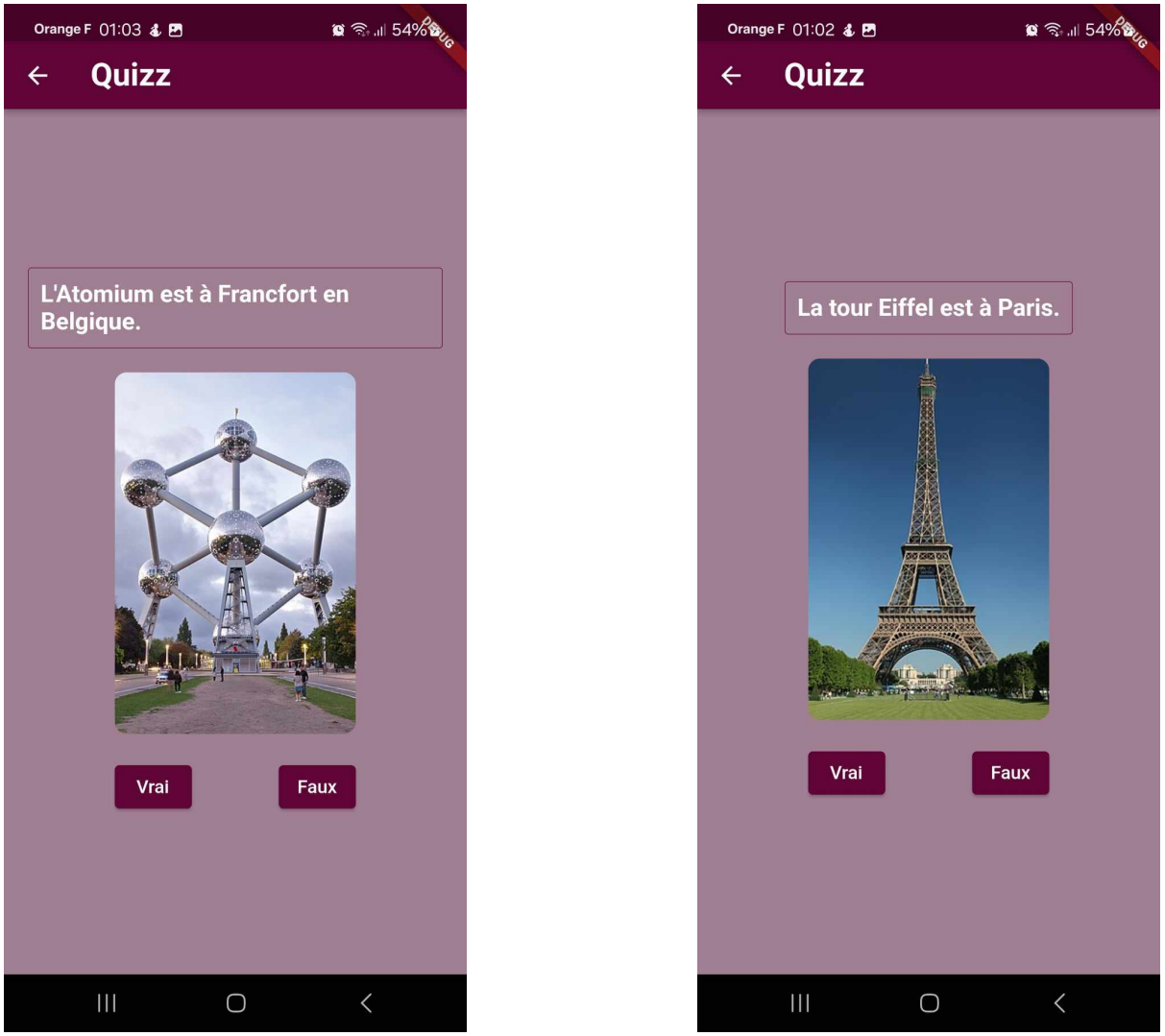

FIGURE 8 – Pages d'affichage du quiz

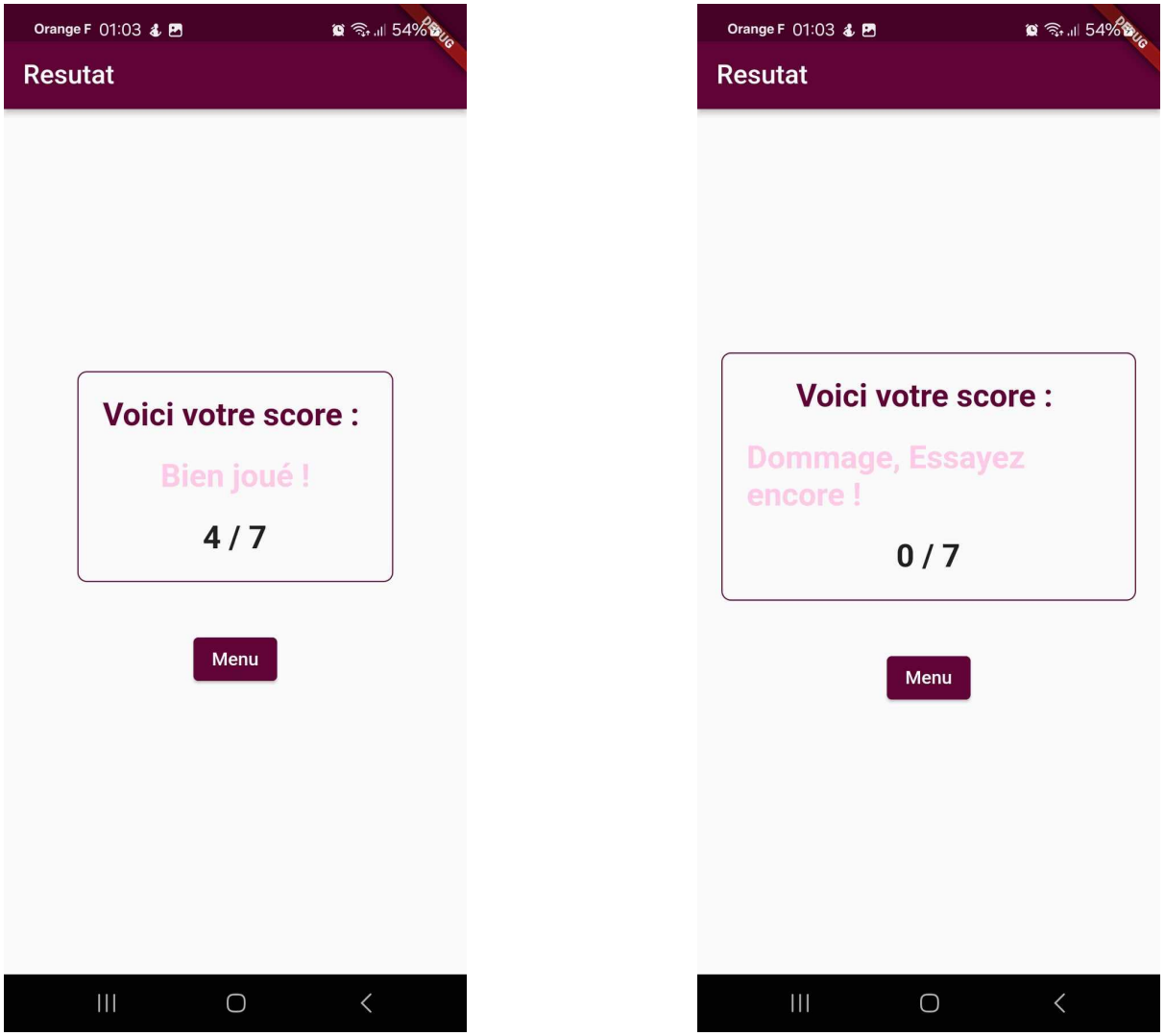

FIGURE 9 – Pages d'affichage des résultats

Ensuite, pour la météo nous avons les deux pages de recherche.

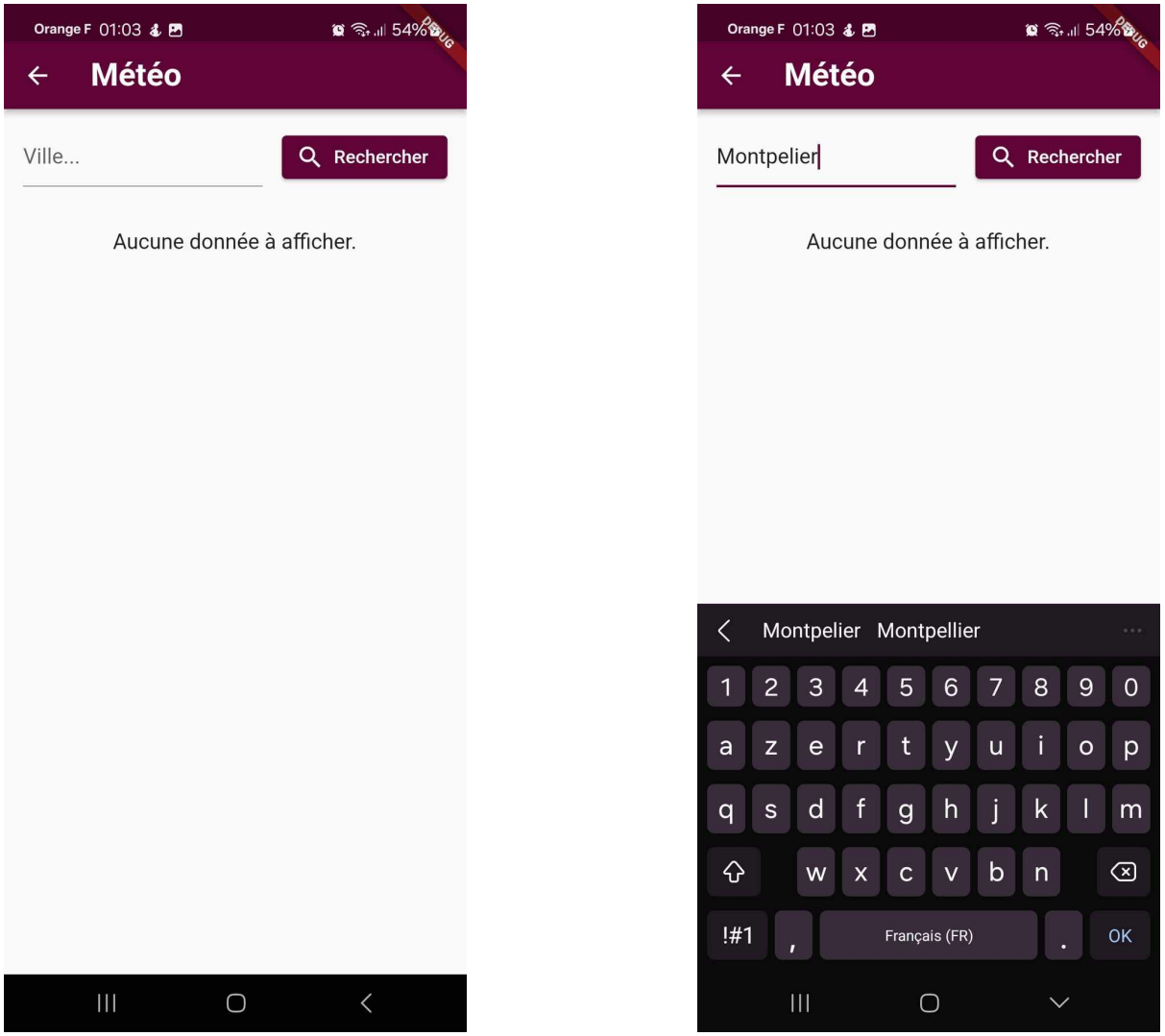

FIGURE 10 – Pages d'affichage de recherche de la météo

Pour finir, nous aurons les deux affichages de la météo de Montpellier un pour le général et l'autre pour montrer le scroll possible en bas de l'écran pour les températures avenir.

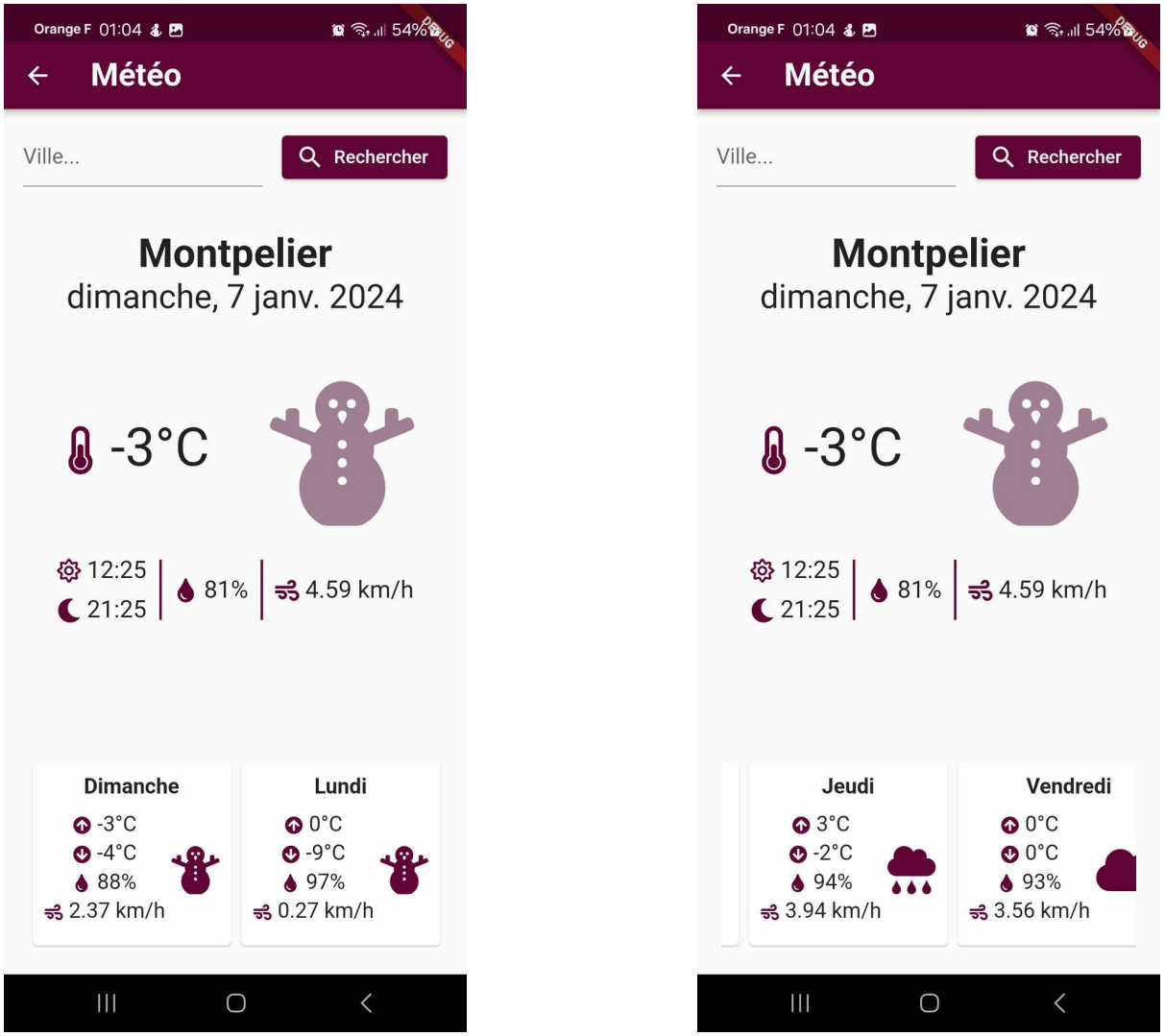

FIGURE 11 – Pages d'affichage de la météo

3 Conclusion

Ainsi, pour ce 2ème TP, nous avons vu les différentes choses pour gérer les états d'une application et notamment ici pour faire un quiz. De plus la gestion de la météo nous a permis de savoir utiliser une API dans notre application Flutter.