Architecture : client-side

Dernière modification : 12 fev 2020

Technologies

Nous avons choisi de developper notre application avec **Flutter**, un framework open source developpé par Google (et par la communauté !).

Voici les raisons qui nous ont menées à chosir **Flutter** :

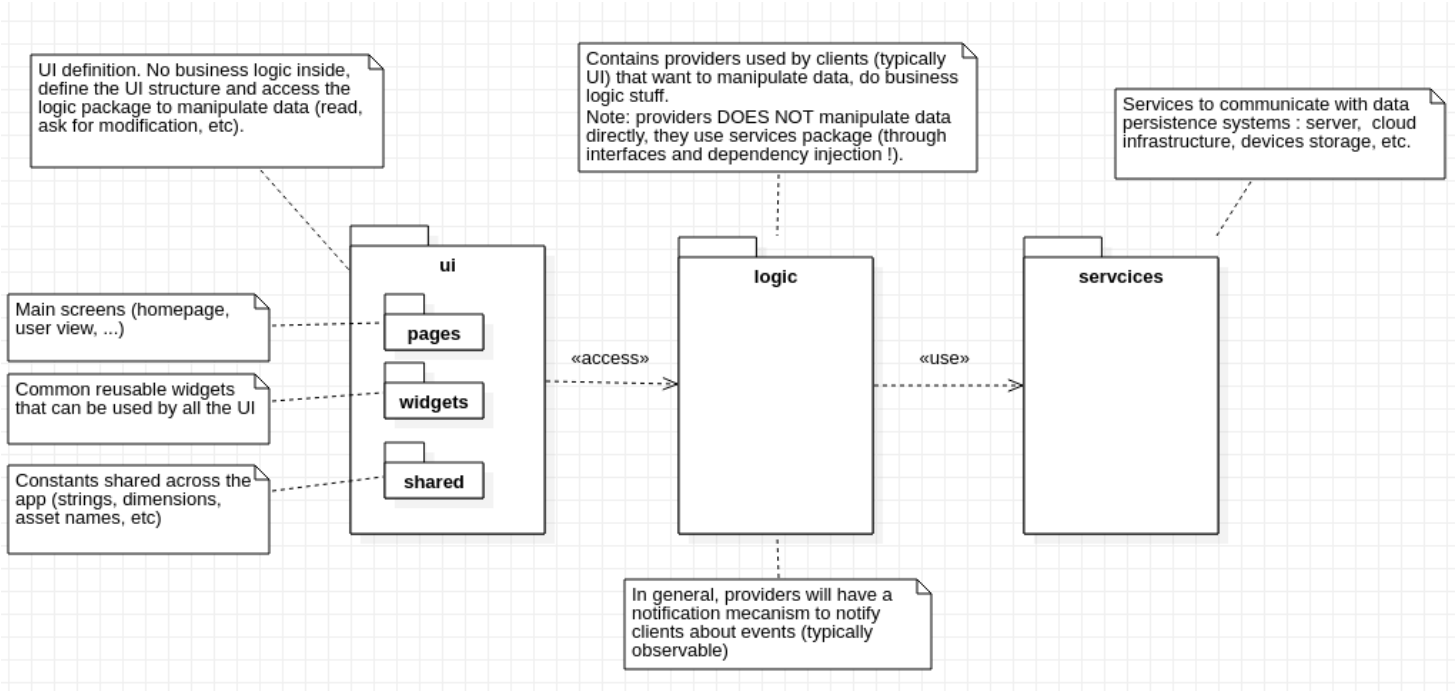
- **Popularité :** Flutter est un framework activement soutenu par sa communauté, avec de nombreuses resources nous permettant de nous documenter (voir à titre d'illustraction la figure ci-dessous)
- **Cross-platform :** permet le developpement sur Android et iOS avec le même *code base* (et aussi Web, Desktop en version bêta)
- **Rapidité de developpement** : Flutter se base sur le langage de programmation *Dart*. C'est un langage qui permet la compilation AOT (Ahead of time) et JIT (Just in time). Lors de developpement l'application est compilé JIT et tourne dans une machine virtuelle pour permettre d'appliquer les changements très rapidement. La version finale, elle, est compilé AOT en langage natif (x64/ARM) pour améliorer les performances et réduire la taille de l'application.
- **La philosophie** : Flutter utilise le *declarative style* pour developper les interface graphique où tout est widget, programmer de cette manière est très appréciable selon nous
- **Framework récent (mais mature)** : Flutter est un framework récent basé sur un langage récent, les architectes / ingénieurs ont donc fait des choix permettant de facilement produire du code clair, bien structuré, suivants des principes de conception élégants

	First release date	Stars (Github)	Contributors	# questions on Stackoverflow
Flutter	Dec 4th, 2018	86.6k	530	34k
React Native	Jan 28, 2015	84.k	2000	67k

Date: Feb 12, 2020

Architecture

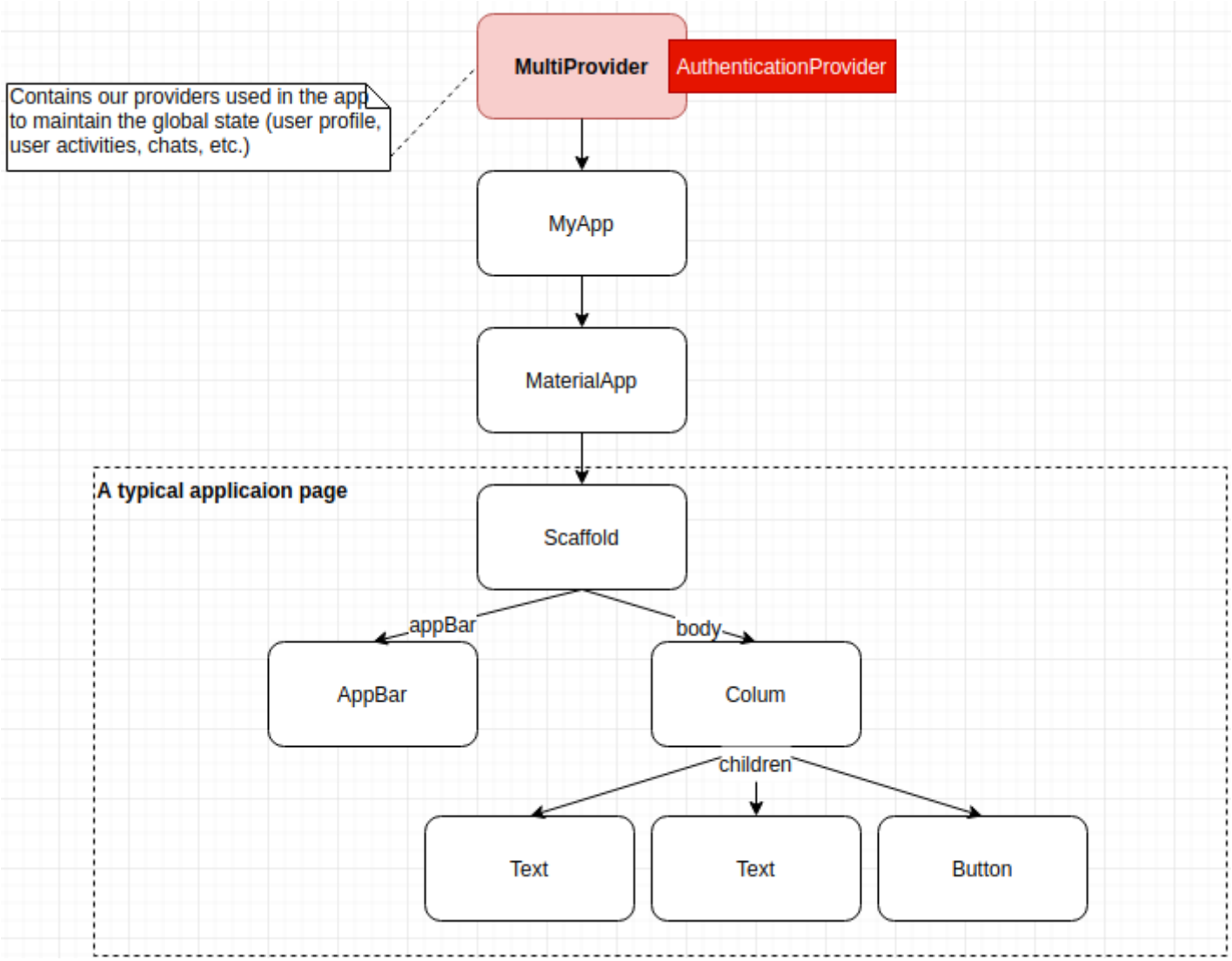
Voici le diagramme de package côté client :



UI

Est contenu dans le dossier `ui/` . Contient les différents Widgets et constants permettant de définir l'interface graphique. Communique avec les providers (contenus dans le package `logic`).

Typiquement voici un arbre représentant une page de notre application :



Les providers sont définis tout en haut de l'arbre, et donc accessible par les enfants.

Logic

Contient nos providers permettant de gérer l'état **global** de notre application, typiquement l'utilisateur. Ils sont définis à la création de l'application (*pas forcément créé à ce moment là, ils seront créés uniquement lorsque quelqu'un en aura besoin)*. Ils seront contenu dans un Widget pouvant les contenir, voir l'abre de widget ci-dessus.

La plupart du temps ils extendront la classe `ChangeNotifier` pour notifier les clients (observer/observable pattern) :

> A class that can be extended or mixed in that provides a change notification API using VoidCallback for notifications.

Services

Permettent la communication avec les serveurs (manipulation de données : écriture, lecture, suppression).

Voici un diagramme de classe montrant l'architecture actuelle avec l' `AuthenticationProvider` et le service permetant l'authentification. Les providers ont connaissance des services via leurs interfaces, et sont injectés dans le constructeur (IoC)