Architecture

Clean Architecture is a software design principle which has the goal to give an idea of organizing and structuring code to make it maintainable and scalable. The primary goal of *Clean Architecture* is to separate concerns and dependencies, making the base of the code more flexible and adaptable to changes.

Goals

Some of the main goals of this approach can be summarized in these points:
- **Independency from frameworks**: this makes the software more portableand adaptable to changes in the underlying technology;
- **Independency from UI**: this makes the software more reusable and adapt-able to the changes ofUI;
- **Independency from database**: this makes the software more portable andadaptable to changes in the underlying database technology;
- **Independency from external agencies**: this makes the software more portable and adaptable to changes in theunderlying infrastructure.

Implementation

The key idea of this approach is to encourage a clear separation of concerns, having the most important and stable business logic at the center and the details at the side. Directories of the codebase enhance this separation of concerns perfectly, indeed there are data, domain and presentation folders.

Data layer

Data layer is suppose to contains repositories' concrete implementation: that's it, classes in this layer have to fetch datas. Treatments and purchases lists are retrieved from *Firebase*, where there are treatment and purchase collections. In these collections datas are encoded like in the app, so for serialization json library is used. Treatment:
- id: Uuid
- date: DateTime,
- pesticides: Map<Pesticide, Quantity>

Pesticide Purchase:
- id: Uuid
- pesticide: Pesticide,
- quantity: Quantity,
- price: double

Weather informations are fetched from *Open-Meteo* API, which is a free API based on national weather reports. After fetching datas, a list of object is creates, where each object represent the weather in a day since last treatment, therefore it has:
- day: DateTime,
- maxTemperature: double,
- minTemperatire: double,
- precipitationSum: double

Business layer
As said above, this layer is totally independent, it contains the entity used throughout the app, use cases and the interface of repositories.

Repository
Repository folder contains all repository intefaces definitions. In this way all code within this layer depends on that interfaces and not in their concrete implementation.

Entity
Here you can find all entities used throughout the application. It also contains a *generated* directory in which you can locate the auto generated file for serialization purpose.

Use case
Each screen has its own use case class, which contains all use case needed for the screen. Some *Clean Architecture* guide suggest to create one class per use case, nevertheless I think this is a little bit confusing so I prefered to create one class per screen and move into its all use case.

Presentation
Contains screens and widgets.