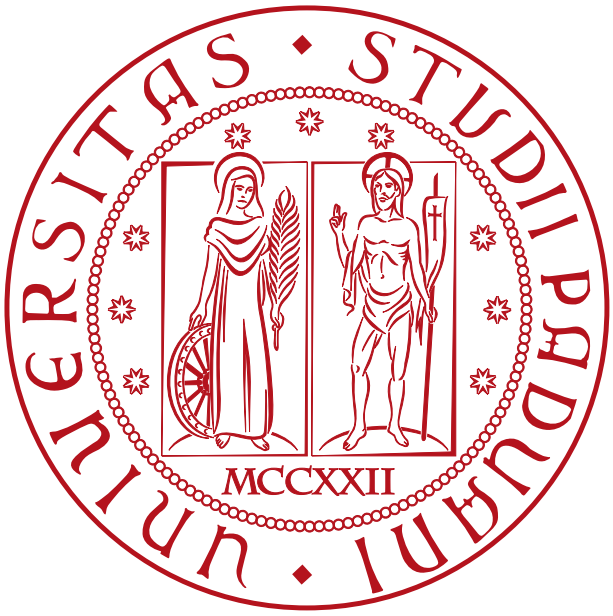
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Computer science



Vineyard Guard

Technical report about the development of the mobile application Vineyard Guard as project for the Mobile Programming and Multimedia course, Master Degree in Computer Science at University of Padua.

Luca Brugnera

2122421

---

academicYear 2023-2024

Contents

1 Introduction

1.1 Project overview

Tracking phytosanitary treatments in vineyards is essential to mitigate the risks of vine diseases and to promote sustainable wine production. This approach helps reduce waste and fosters greater respect for nature.

The lack of an easy and direct way to track these treatments leds to the creation of *Vineyard Guard*, an application designed to manage pesticides used in farming. *Vineyard Guard* supports farmers from the start to the end, from the purchases of pesticide to its consumptions.

More in details, *Vineyard Guard* enables you to:
- Maintain a list of completed treatments, which can be shared with your agronomist.
- Track pesticide purchases to continuously monitor costs.
- Monitor warehouse stocks to ensure timely replenishment.
- Monitor weather conditions since the last treatment to schedule the next intervention.

The app is tailored for family businesses and does not require user management. Nevertheless data are stored in *Firestore*, therefore the application can be used in multiple devices.

1.2 Flutter

Flutter is an open-source UI software development kit (SDK) created by Google. It allows developers to build natively compiled applications for mobile (iOS and Android), web, and desktop from a single codebase. Essentially, this means you can write one codebase and deploy it across multiple platforms, saving time and resources.

Pros of Flutter development:

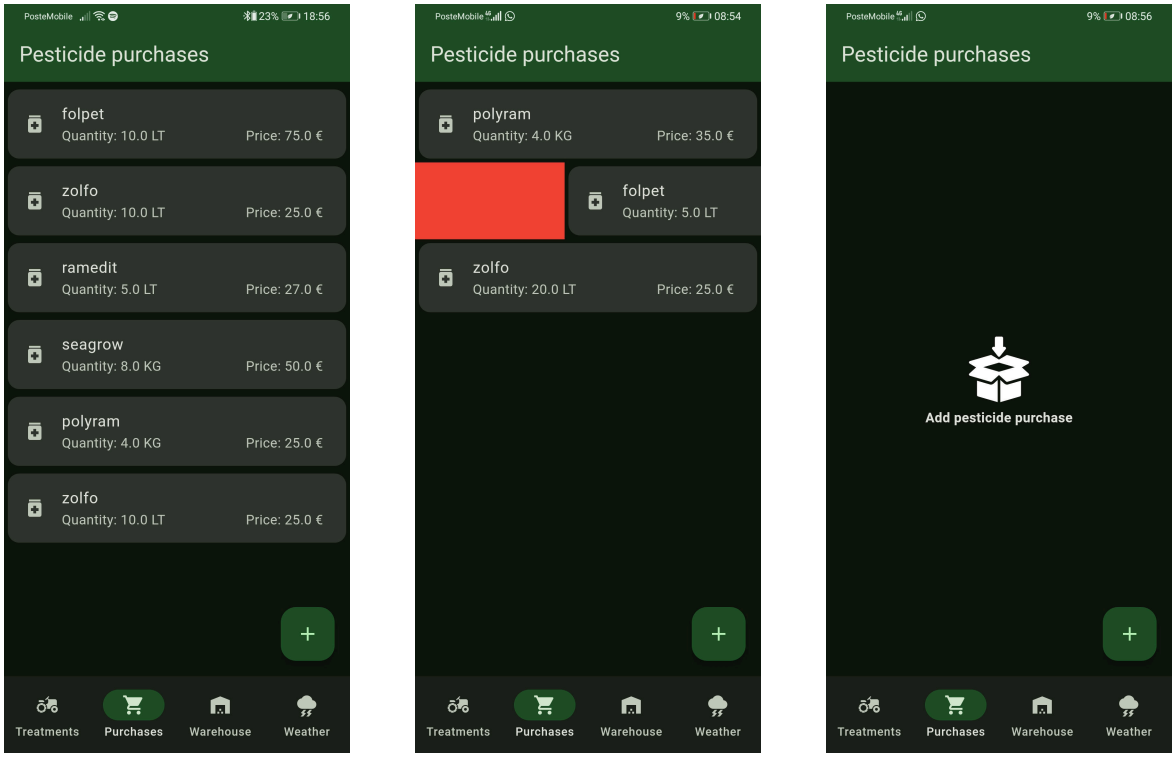
- **Rapid development**: Flutter's hot reload feature allows for quick UI updates without restarting the app, significantly accelerating development cycles.
- **Beautiful and customizable UI**: Flutter provides a rich set of customizable widgets, enabling developers to create stunning and visually appealing apps.
- **Cost-effective**: Developing a single codebase for multiple platforms can lead to substantial cost savings.
- **Large and growing community**: Flutter has a thriving community, offering extensive support, resources, and third-party packages.
- **Open-source**: Being open-source, Flutter is free to use and contributes to a collaborative development environment.

Cons:
- **Large app size**: Flutter apps tend to be larger than native apps, which can be a concern for users with limited storage.

2 Screen and functionalities

Vineyard Guard is made of 4 different screens: treatments, purchases, warehouse and weather. Bottom navigation bar is provided for navigation among screens, even in android, since nowadays is common despite the possible presence of operating system navigation bar. Moreover the implementation of and hamburger menu was not on my schedule during development due to time constraints.

2.1 Pesticide purchases screen



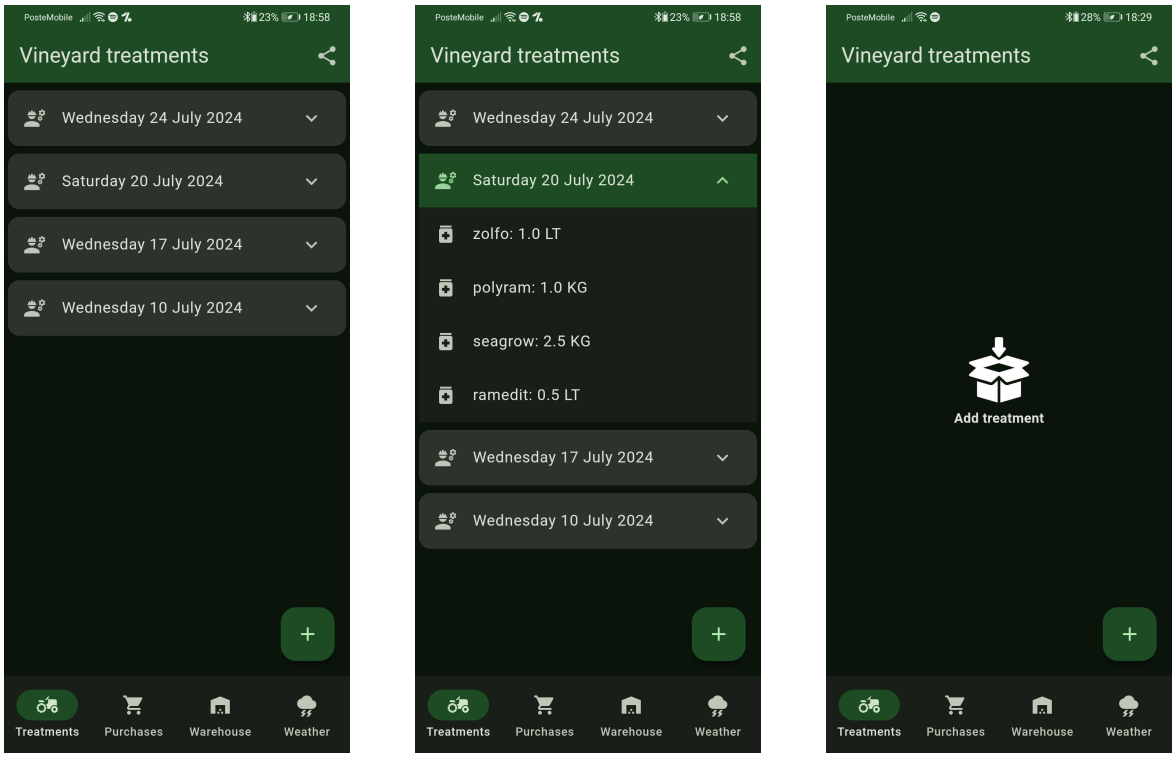
This screen displays all pesticide purchases as a list of cards, with each card representing a purchase and showing the pesticide name, quantity, and price. In the case where there aren't purchases an empty box with an error message is showed.

Purchases of the same pesticide are treated independently to keep management simple. Pesticide names are case-sensitive to avoid user confusion and ensure precise tracking: it means that names with different upper or lower case characters identify different pesticides.

Purchases can be added using the floating action button, with a form that requires the pesticide name, price, amount, and unit of measure (kilograms or liters). Unit of measure selection is a perfect candidate for radio button since there are only 2 possibilities and it removes unnecessary clicks speeding up the whole process.

To remove a purchase, simply swipe it away. This swipe-to-remove functionality is consistent throughout the application improving ease of use (it removes unnecessary tap or click) and accessibility (it requires less precision).

2.2 Treatments screen

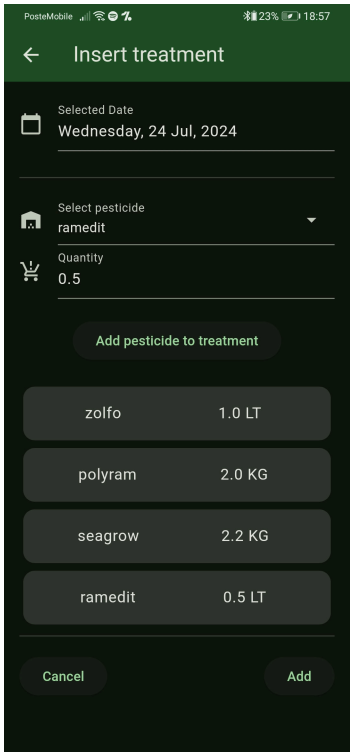


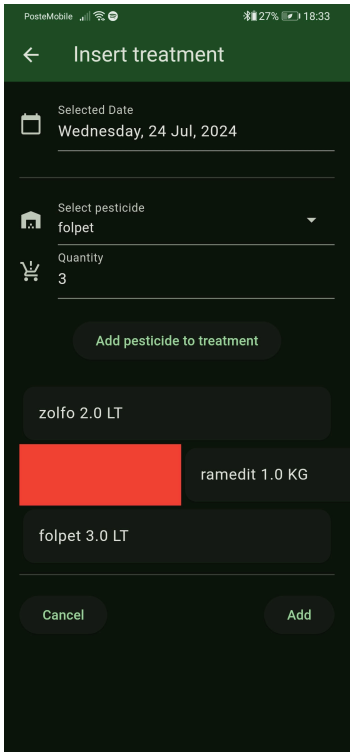
This screen displays all treatments as a list of cards, each uniquely identified by its date. Tapping a card reveals more information about the treatment, including all pesticides used and their quantities. Expanded treatment has different colors to highlight itself among the others. During development expanded card was choosed over a more datailed screen since the informations to show per each treatment are limited and in this way no additional layer of depth is added, gaining in easy of useness.

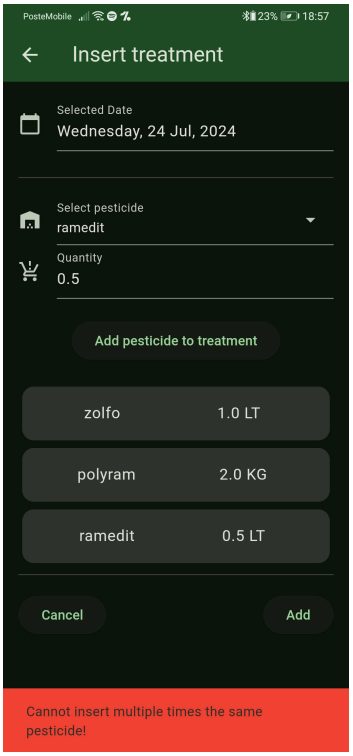
As mentioned above, to delete a treatment, just swipe it away.

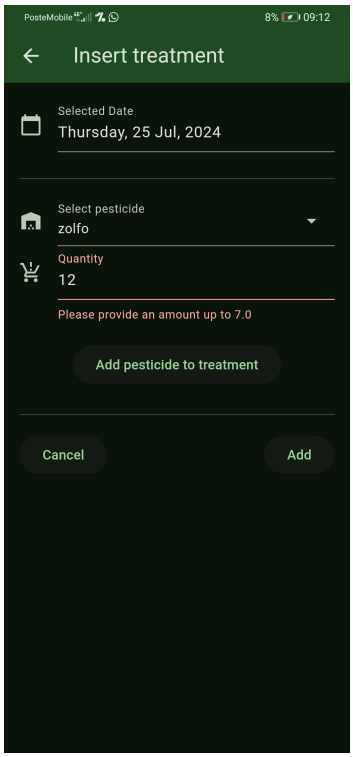
Treatments can be shared as pdf file using the share button in the header. The generated pdf contains a table where each treatment is a row with date as first column and pesticides used with relative quantities as second column. In this way the agronomist that follows the farm throughout the year can suggest the right pesticides and timing for next treatments to minimize the risk of diseases and reduce waste.

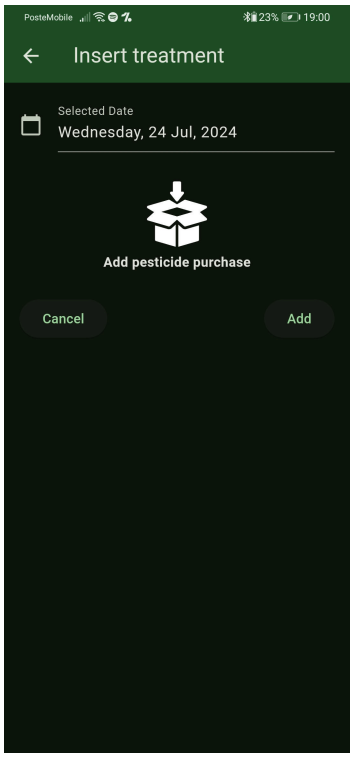
2.2.1 Insert treatment form











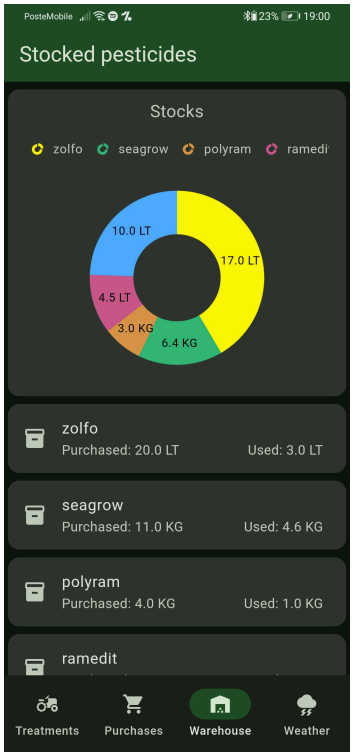
You can add a treatment using the floating button by specifying the date and the pesticides used. Date selection is made by the date picker and the current date is provided as default, since the use case scenario supposes that the operator adds the treatment right after its executions. Pesticides used in treatments must be selected among the ones that are available in the warehouse. This operation is implemented via drop down search menu: the search functionality is added due to the possible hight number of pesticides available. Indeed the user doesn't have to scroll the entire list to find the right

pesticide, it sufficient to search its. While adding pesticides to treatment a list of pesticides used is build under the add button to keep track and manage them. Pesticides available in the warehouse correspond to those purchased and not yet fully used in previous treatments. The creation of a dynamic list using a form is not a simple design task, anyway the final result trys to take down complexity and be as simple and direct as possible. The section of the form for selecting pesticides used is divided by lines to highlight the fact that it is a sub-part of the main form. Other possible design solution for this form (like moving it one step further in depth) were discarded during development since they arise the overlall complexity and are less intuitive than the final one.

If you try to add more than stocked or add a treatment without selecting any pesticide, an error message is displayed. Obviously if you try to add a treatment with no pesticide purchased, so with no pesticide available in the warehouse the app shows you an empty box and suggest you to purchase some pesticides.

Multiple instances of the same pesticide are not allowed to keep the process simple.

2.3 Warehouse screen

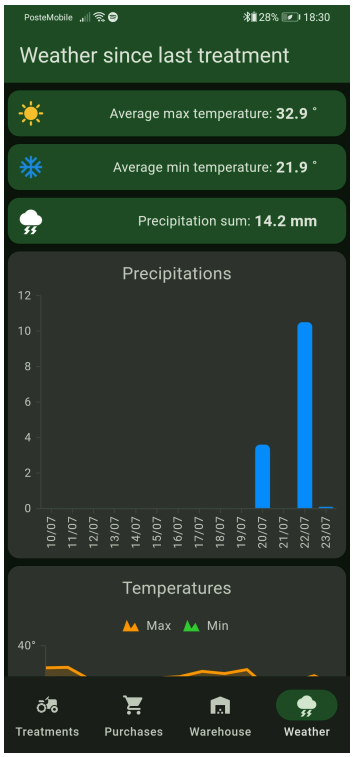


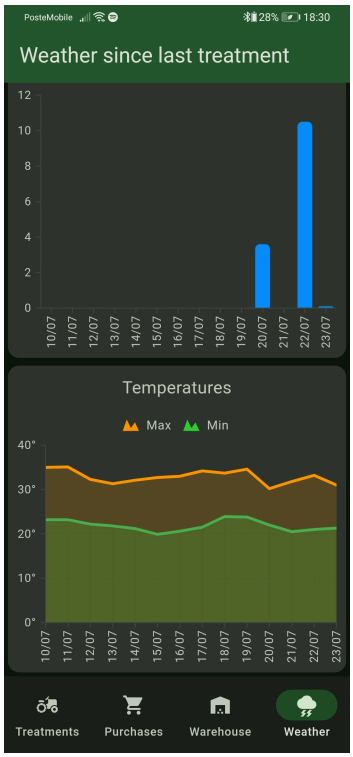
Vineyard Guard doesn't only track treatments, it also tracks purchases. This screen lists the pesticides available in the warehouse, showing the amount purchased and the amount used. The goal of this screen is to keep track of the warehouse status, ensuring timely notifications to suppliers to avoid delays.

To do so in the most efficient way a pie chart is provided, where each slide represent a stocked pestice in the warehouse. Pie chart only creates a visual representation of the list of stocked pesticides that is below, so the talkback gives to the user only a brief description rather than the entire information.

This information is read-only, as it depends on purchases and treatments, so users cannot discard any cards from this screen.

2.4 Weather screen



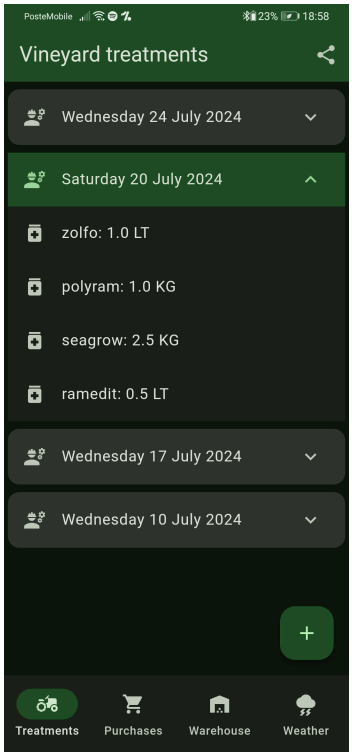


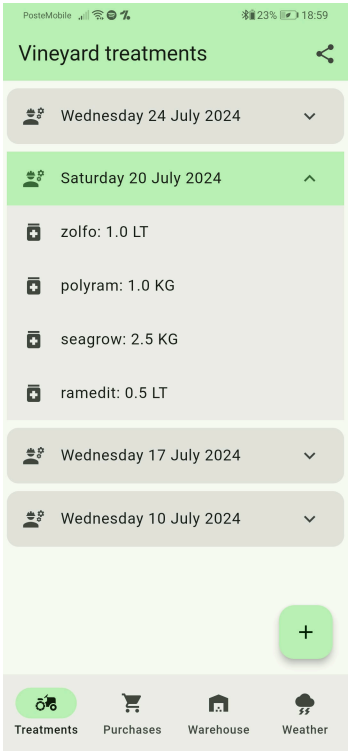
The frequency of treatments depends on precipitation and temperature since diseases are influenced by these factors. This screen shows weather information since the last treatment (with an upper limit of 14 days), if the difference between such date and the current is less than one day, then an error message is displayed.

As for the whole app, this screen display information on cards, one for each average statistics and one for each charts. Charts are accessible, therefore talk back explains the entire content after a short initial description.

This screen helps the farm better schedule the next treatment and select appropriate pesticides based on the weather conditions.

2.5 Theme





Nowadays is not reasonable that an application misses light or dark theme. *Vineyard Guard* use theme selected from the device.

3 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.

3.1 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.

3.2 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.

3.2.1 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

3.2.2 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.

3.2.2.1 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.

3.2.2.2 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.

3.2.2.3 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.

3.2.3 Presentation

Contains screens and widgets.

4 Conclusion

4.1 Test

Vineyard Guard was developed using my personal smarthphone as debug device, nevertheless it was tested also in the following devices:

- Google Pixel 8
- Google Pixel Tablet
- iPhone 15
- MacBook

4.2 Personal opinions

In the end I am prod of the end result. This was my first experience of mobile development. Use of a state management tool like *Bloc* would have been useful indeed it would have make the entire development process easier, nevertheless *Flutter* is a fun tool to use, with a lot of points in favor and its popularity is well deserved.