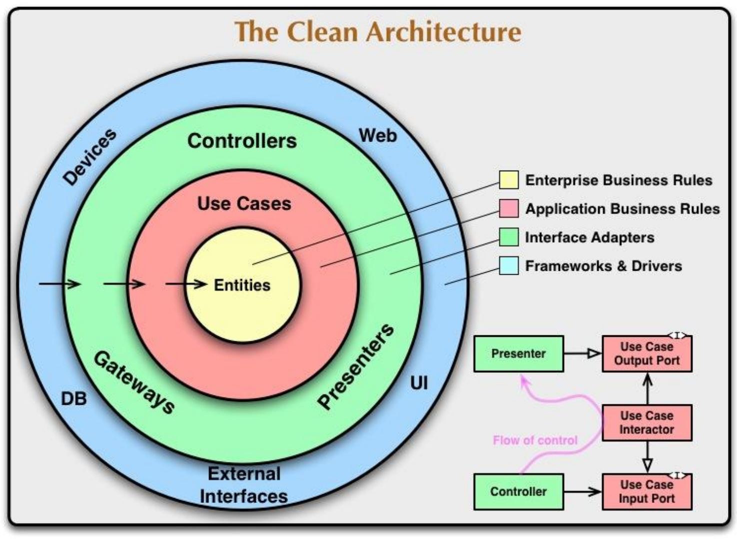
Objective

To get cryptocurrencies' values per day and the forecasting of them was created, this project can enable the next projects to use these values with a simple call in the API.

Clean Architecture



For developing this software it was decided to use the "Clean Architecture" because this model was created with focus on business rulers and the "Clean Architecture" model this software having the business rulers on the core. This makes this kind of architecture better for models' deployment. Below are all the responsibilities of every layer.

Entities (Inside Domain)

The core layer, every layer depends on it and holds the most important business rulers

Responsibilities:

- **Currencies**: guarantee the correct data types on the dataset, guarantee the existence of a cryptocurrency and a market currency and guarantee the existence.

Use Cases (Inside Domain)

This layer holds some business rulers who are less important than the rulers in entities and the use cases of the software that the users are going to need.

Responsibilities:

- **Sarimax (Model)**: training and forecasting with this model.
- **Crypto Transfomer**: transform cryptocurrencies' data to the correct format.
- **Scale Changer**: rescale all features to the model running.

Connectors (Inside Domain)

This layer has the responsibility to connect the business rulers from inner layers with the external factors.

Responsibilities:
- **Data Getter**: get data from the API of the Alpha Vantage.
- **Sarimax Runner**: run sarimax with the data obtained.
- **API Pre-Unifier**: unify data getter and sarimax runner for calling after in the framework.

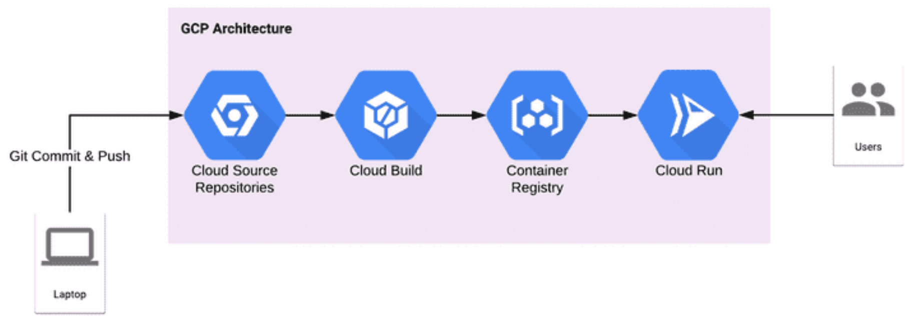
External Factors (Outside Domain)

This layer has the responsibility to hold all external factors who are not in the domain and can need to be changed are very flexible to be changed.

Responsibilities:
- **Framework Flask (main.py)**: create an API to expose the application in an endpoint.
- **Tests**: unitary and manual tests for debugging the software.
- **Docker**: containerization of the application to guarantee the accuracy.

Google Cloud

This application was deployed on Google Cloud using two services from it.



Container Registry

This service was selected to keep the Docker Image of this project which is being used to guarantee the accuracy of this application in relation with the performance on localhost.

Cloud Run

This service runs the image from the container registry to a serverless application, creating the endpoint of this application and possibilities for this application to be requested.

CI/CD

GitHub Actions

All the CI / CD process was done with the github actions, authenticating with a Service Account Key, **creating the docker image who runs all the tests (CI)** and **sending this docker image for the cloud run (CD)**.