

Projet 2D ESME   
Groupe 3

Une image contenant texte, clipart, grille

Description générée automatiquement

BONNAFOUS Victor BOYER Adrian MIERMON Alice

&

HENRIO Victor

AVA crypto

# Remerciements

Dans un premier temps, nous souhaitons, tout particulièrement, remercier Monsieur HENRIO, ancien élève de l’ESME, qui nous a accompagné, sans faille, tout le long de ce projet. Sa disponibilité, son savoir-faire, et ses propositions nous ont permis de dépasser nos limites et de réaliser un projet dont nous sommes fiers. Mêmement, nous souhaitons le remercier pour nous avoir rassuré sur notre avenir en tant que futur ingénieurs-manageurs dans le domaine de la data.

Nous voulons également remercier Madame KAMOUN, et l’ensemble de l’équipe pédagogique de la majeure Big Data & Digital Marketing, de nous avoir proposer des projets stimulant de 3 mois permettant de mettre en pratique nos connaissances acquise le long de ce semestre.

Enfin, merci aux différents professeurs ainsi qu’à la promotion 2D : Big Data & Digital Marketing, pour les différents moments d’entraides et de conseil.

# Sommaire

Remerciements  
Sommaire

Table des figures  
Intro

Contexte et problématique du projet

-> présentation contexte initial (projet)

-> problématique et objectif (outils utilisés)

-> présentation équipe

Conception et implémentation de la solution

-> traitements des données : VICTOR : features

-> visualisation des données : ALICE

-> BDD Heroku : ADRI

D’un point de vue marketing ? Digital Marketing

-> budget pour une telle app ?

-> marché ?

Conclusion

-> retours / piste d’amélioration

Sitographie

# Table des figures

# Introduction

Chaque année, durant le second semestre de 4ème année, dans le cadre de la majeure Big Data, il nous est proposé un projet. Ce projet nous permet d’utiliser et d’approfondir les notions vues tout le long du semestre, mais également d’acquérir des nouvelles connaissances.

Après présentation des différents projets, nous avons choisi le projet n°4 portants sur la réalisation d’un Dashboard suivi de portefeuille de crypto monnaie. Utilisateurs de ce type de service, ce projet nous intéressé d’autant plus qu’il concerne un domaine nouveau et important du web 3.0 : la notion de blockchain. En tant que futurs ingénieurs-manageurs en Big Data, nous sommes conscients de l’impact du web3.0 dans notre monde connecté, et c’est également pour cette notion que ce projet était important pour nous.

C’est dans le cadre de l’utilisation de la blockchain via des Interfaces de Programmation (API) que notre projet s’inscrit. Dans ce rapport, nous allons vous détailler, dans un premier temps, le contexte initial ainsi que la problématique du projet. Ensuite, la conception et l’implémentation de la solution y seront expliqué en détails. Enfin, étant également des futurs ingénieurs – manageurs en Digital Marketing, nous avons souhaité apporter les connaissances que nous avons acquises en marketing dans ce projet pour lier les deux domaines de notre majeure.

# Contexte et problématique

## Contexte initial

Comme évoqué brièvement dans l’introduction, ce projet est directement relié au web 3.0. De nos jours, le web3.0, et les crypto monnaies qui en découle sont en plein essor avec un marché de plus en plus convoité.

En effet le web3.0 offre des possibilités totalement différentes que les versions du web précédente. Le web3.0 est une version d’Internet permettant de repenser une toute nouvelle gestion des données de manière transparentes, décentraliser et sécurisé, grâce à la blockchain, sans qu’elles soient contrôlées et détenues par les GAMAM (Google, Amazon, Meta, Microsoft) par exemple. La blockchain est une technologie permettant à 2 utilisateurs de réaliser des transactions de manière complétement transparentes ; tous les utilisateurs de la blockchain peuvent voir les différentes transactions réalisées dessus permettant la décentralisation des données : il n’y a plus de grandes sociétés contrôlant toutes les données. Le graphique ci-dessous nous montre bien que le web3.0 deviendra le web principal des prochaines années.

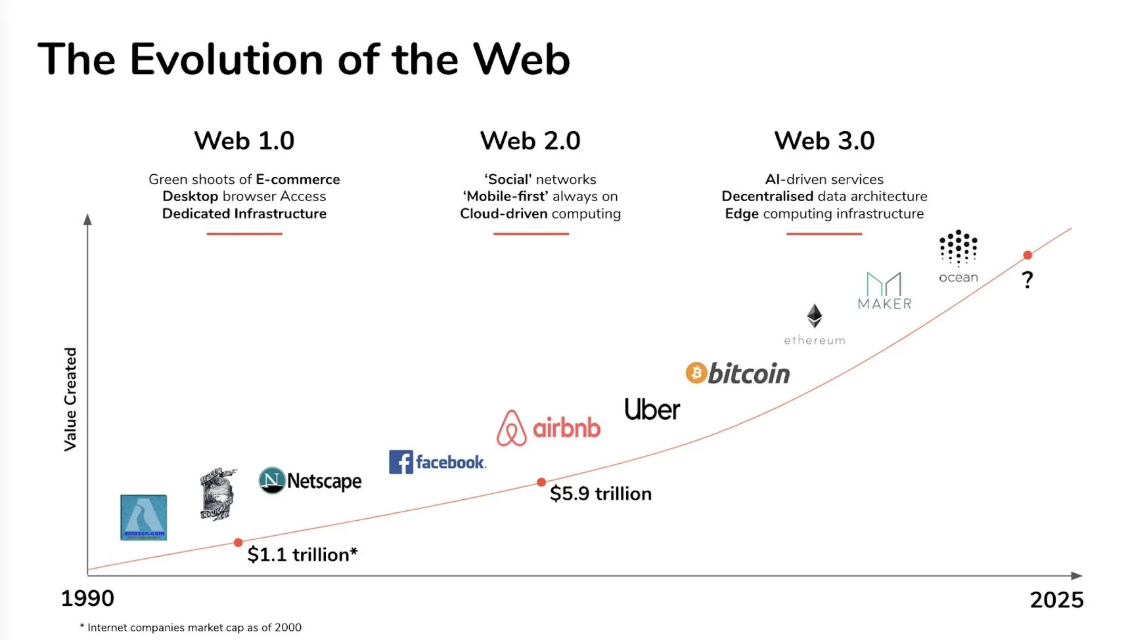


Figure 1 : L’évolution du web

## Problématique et objectif

Avec cette montée des crypto monnaies, les utilisateurs ne demande que des solutions de visualisations simple et efficace pour pouvoir surveillés leurs portefeuilles de crypto monnaies ; grâce à la transparence de la blockchain, il est possible de récupérer facilement ces données. La problématique de ce projet est donc :

Comment rendre facilement accessible ces données brutes à un utilisateur ?

Pour réaliser au mieux ce projet, un objectif a été défini : réussir à mettre en œuvre des algorithmes sous forme de pipeline de données qui permettrons d’exposer les données après les avoirs récoltés, stocker et traiter.

Pour ce faire, nous avons utilisé des Interfaces de Programmation connues sous le nom d’API. L’API est une façade permettant à 2 systèmes informatique, via des requêtes, de se communiquer entre elles, et de faire appel à leurs différentes fonctionnalités et donc par extension de s’échanger des données ou des services (cf. figure2 ci-dessous). Il y a différents types d’API, ici nous avons utilisé des API ouvertes qui sont disponible publiquement sans restriction.

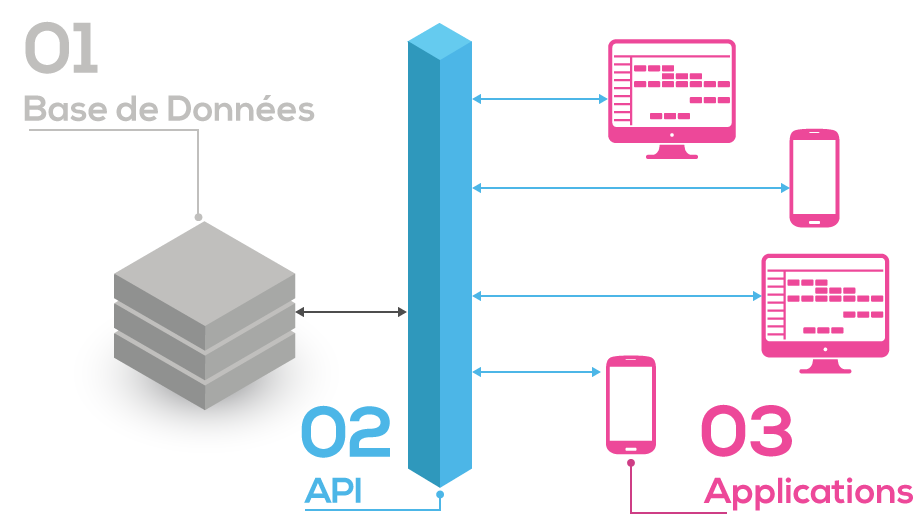


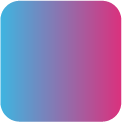
Figure 2 : Fonctionnement API

En plus d’API, nous avons dû trouver une solution de visualisation appropriée permettant aux utilisateurs de suivre en temps réels le contenu de leurs portefeuilles de crypto monnaies.

De plus, l’application devra être hautement disponible, résilient au nombre d’utilisateur, et pouvant être mis à jour facilement grâce à GIT. GIT est un logiciel de gestion de version décentralisé permettant la collaboration, la mise à jour, l’historique etc. de projet informatique.

Pour réaliser ce projet, nous l’avons réalisé sous python en travaillant également sous GIT (lien de repository GIT page 4).

Nous avons également essayé de réaliser ce projet le plus professionnellement possible pour lui donner une dimension managériale. Notre application se nomme **AVA Crypto** (cf. logo ci-dessous) et a été réalisé en une équipe de 3 personnes chacune répartie dans différents aspects de l’application (présentation de l’équipe dans la page suivante), avec l’aide d’un encadrant.



AVA crypto

Figure 2 : logo AVA Crypto

## Présentation de l’équipe



BONNAFOUS  
Victor

BOYER  
Adrian

MIERMON  
Alice

CAO   
- Chief Analytics Officer

Responsable traitement des données

DVC

- Data Visualisation Consultant

Responsable visualisation des données

Responsable fonctionnement de l’application

Déploiement de l’application et création base de données

A DEFINIR CF SITE PLEIN D’OPTION

# Conception et implémentation d’une solution

## Structure du Code

La structure du code est la première réflexion que nous avons eu à avoir. En effet, coder à plusieurs n’est jamais simple, nous avons donc dû définir les différente propriété du projet que nous avons devions réaliser.

Pour cela nous avons utilisé le service web GitHub, qui est un service d’hébergement et de gestions de développement de logiciels, utilisant le logiciel de gestion de versions Git. Nous avons donc créé un repository collaboratif pour que chacun puisse uploader son code et le partager entre tous. A l’aide de différente *« Branch »*, nous avons pu chacun avancer sur le projet sans rencontrer de conflit dans nos codes.



Par la suite nous avons réfléchit à une structure qui suivra notre projet jusqu’à la fin. Pour cela nous avons pensez à chaque *feature* que nous devions créer, à l’application de visualisation *dash* que nous allions créer et les tests que nous devions réaliser pour analyser notre code python.

C’est pour cela, que la structure de notre projet prend cette forme ci-dessous :

project/

|-- .github/workflows

| |-- python-app.yml

|-- src/

| |-- app

| |-- \_\_init\_\_.py

| |-- dash

| |-- \_\_init\_\_.py

| |-- app.py

| |-- database

| |-- \_\_init\_\_.py

| |-- database.py

| |-- feature\_history

| |-- \_\_init\_\_.py

| |-- link\_address\_history.py

| |-- address\_history.py

| |-- feature\_price

| |-- \_\_init\_\_.py

| |-- price.py

| |-- feature\_transaction

| |-- \_\_init\_\_.py

| |-- link\_address\_transaction.py

| |-- transaction.py

| |-- feature\_wallet

| |-- \_\_init\_\_.py

| |-- wallet.py

| |-- link\_address\_wallet.py

| |-- all\_adrdress\_wallet.py

| |-- test\_app/

| |-- \_\_init\_\_.py

| |-- test\_dash

| |-- \_\_init\_\_.py

| |-- test\_dash.py

| |-- …

| |-- … .py

|-- main.py

|-- .gitignore

|-- env\_var\_setup.sh

|-- Procfile

|-- requirements.txt

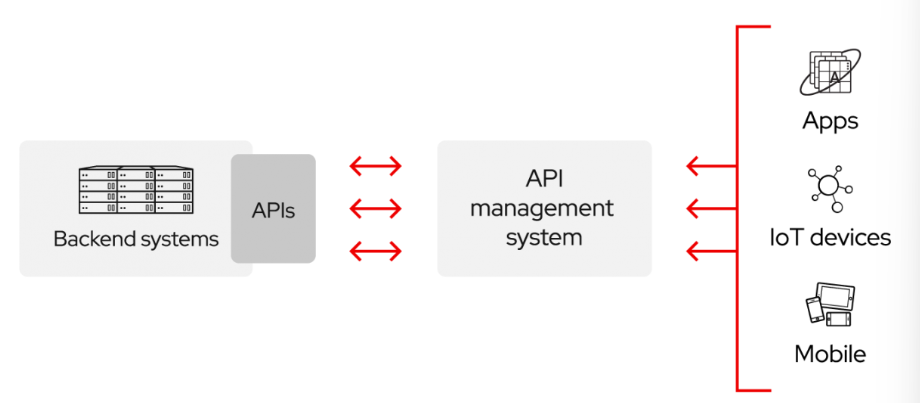
|-- runtime.txt

|-- README.md

Lien Repository GitHub : <https://github.com/Nafbo/AVA-Crypto>

## Traitements des données

Pour réaliser ce projet, et récupérer les informations d’un portefeuille crypto, nous avons utilisé la technologie des API. En effet, une API (Application Programming Interface) est un ensemble de définitions et de protocoles qui facilite la création et l’intégration de logiciels d’applications. L’utilisation d’API dans notre application, nous permet de communiquer avec d’autres services, comme Covalent et CoinGecko, sans connaitre les détails de leur mise en œuvre. Il y a donc un contrat entre les deux parties, avec une documentation qui constitue un accord entre ces parties.



Dans notre cas, comme cité précédemment, ce sont deux API que nous avons utilisée. C’est en grande partie celle de Covalent que nous avons utilisé le plus. En effet, Covalent fournit une API unifiée leader dans son secteur, et qui permet d’apporter une visibilité à des milliards de données du Web3. Cette API nous à partir d’une adresse de portefeuille crypto, de récupérer toutes les informations dont nous avions besoin.



Pour ce faire nous avons dû nous créer une API Key, qui nous à permit d’accéder au données que nous mette à disposition Covalent. De plus, à l’aide de différent *endpoint*, nous avons pu récupérer différente donnée selon un même portefeuille crypto. Les données récupérer, à l’aide de la bibliothèque python *request*, sont sous forme de JSON. Après récupération du JSON, nous avons convertit ce dernier en DataFrame (à l’aide de la bibliothèque python *pandas*), pour pouvoir récupérer les données utiles plus facilement.

Nous avons ensuite analysé ces données, et implémenter dans des fonctions pythons pour pouvoir ensuite exploiter les DataFrame dans notre application *dash*.

Comme vu précédemment dans la structure de notre projet, nous avons réalisé 4 différentes feature.

Feature Wallet :

Dans cette feature, nous avons analysé le solde et se qui compose le portefeuille ou les portefeuilles d’un client. En effet a l’aide du endpoint balances\_v2, nous pouvons récupérer le solde et la composition d’un portefeuille cryptomonnaie. Grâce à cette feature, nous permettrons à nos clients de consulter le solde global de tous leur potrtefeuille crypto, ainsi que la composition de ce dernier sans avoir à se connecter à chaque individuellement.

Feature Transaction :

Cette feature nous permet de récupérer toutes les transaction d’un portefeuille à l’aide du endpoint transfers\_v2. A l’aide d’un lien fait avec les données historique du portefeuille nous pouvons récupérer toutes les transactions depuis la création de ce dernier. Nos clients pourront donc connaître tout de leur transfers grâce à notre analyse.

Feature Price :

Dans cette feature nous n’utilisons pas l’API de Covalent mais l’API de CoinGecko. L’API de CoinGeko permet d’alimenter notre applications avec les données cryptographiques de CoinGecko provenant de sources indépendantes, telles que les prix en direct, le volume d'échange, les volumes d'échange, les paires d'échange, les données historiques, les données d'adresse des contrats, les catégories de crypto, les dérivés de crypto, les images et plus encore.

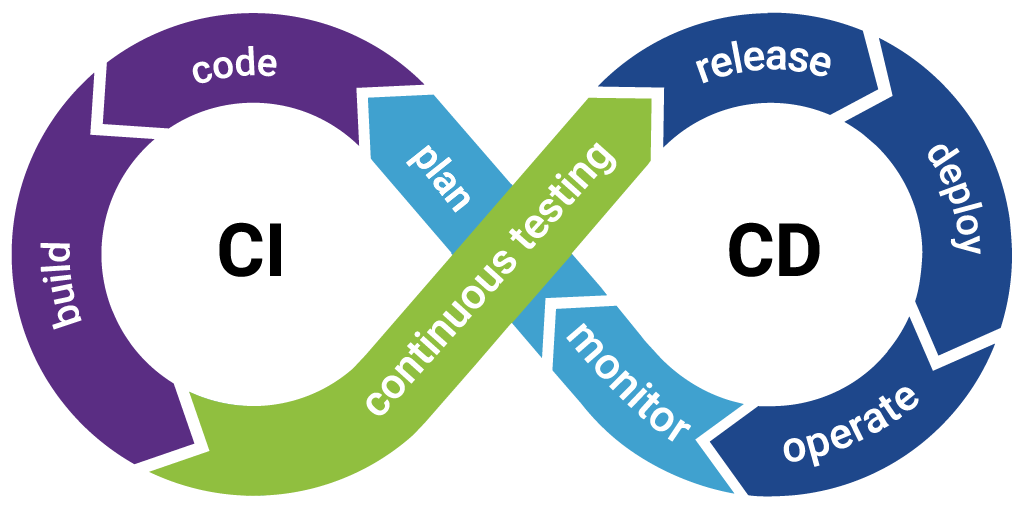
Nous avons donc décidé d’utiliser cette dernière qui nous permettait d’obtenir plus d’information sur le prix de plusieurs cryptomonnaie. C’est pour cela que cette feature, permet à nos clients d’obtenir le prix actuel d’une cryptomonnaie sur notre application, qui est actualisé toutes les 15 secondes.

Feature History :

Pour finir, nous avons voulu afficher l’historique du solde total du portefeuille crypto de nos clients. Pour cela nous avons créer cette feature en analysons les données de l’endpoint portofolio\_v2, qui nous retourne le solde total en USD du portefeuille depuis sa création.

Toutes ces fonctions python features, seront utilisé par la suite dans le dash de notre application pour visualiser les résultats autrement que dans un DataFrame.

Mais à la suite de la création de toute ces fonctions python, nous avons dû les tester pour éviter les erreurs de compilation lors d’un déploiement. Pour cela nous avons utilisé la méthode de CI/CD, cette approche permet d’augmenter la fréquence de distribution des applications grâce à l’automatisation. Lorsque nous envoyant notre code sur github le code python-app.yml se lance et créer une succession de test. Cette approche d’intégration continue et de distribution continue représente une solution aux problèmes de développement et d’exploitation.



C’est pour cela que dans la continuité de notre projet, ainsi que dans l’éventualité d’une implémentation commercial nous avons voulu appliquer cette méthode là à notre code.

Cette implémentation c’est fait directement sur github, dans la catégorie Actions de notre repository. Nous avons donc ajouté une action python qui va venir tester notre code et ressortir les erreurs qui seront gênante lors d’un déploiement. Ces tests sont réalisé à l’aide de *pytest*.

Pour réaliser ces tests nous avons donc dupliqué notre dossier app dans un dossier test\_app, comme vu précédemment dans la structure de notre projet. Tout cela en prenant soin de renommer chaque fichier et chaque fonction, par test\_NonFonction. Car pytest va tester que les fichiers qui son nommer comme cela, et la librairie étant très sensible à la casse cela est très important de suivre le model de la documentation pytest.

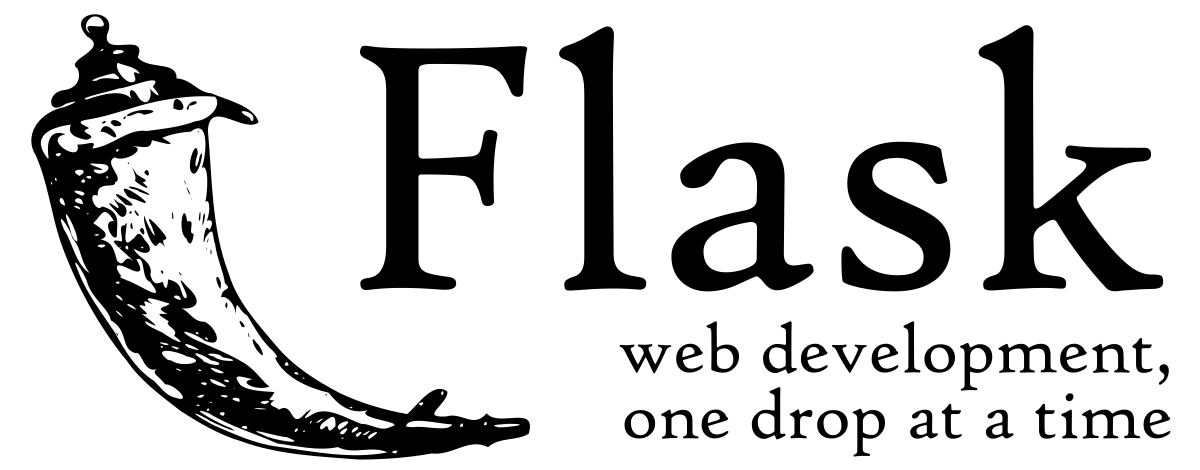
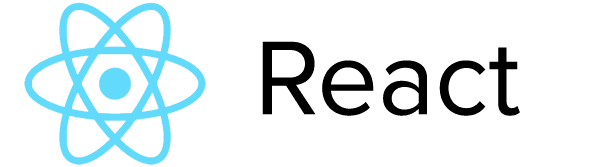


La récupération, l’analyse, l’implémentation ainsi que le test de nos codes étant réalisé nous avons pu réaliser le visuel de notre application.

## Visualisation des données (Localement)



Pour créer notre interface, nous avons décidé d’utiliser Dash. Il s’agit d’un Framework Python qui permet de visualiser aisément des données en développant une ou des applications web. En utilisant Python pour l'analyse des données, l'exploration des données, la visualisation, la modélisation, le contrôle des instruments et la création de rapports, Dash est d’une grande utilité. Il permet de construire simplement une interface de donnée autour d’un code d’analyse de données. Des technologies tel que Flask, React ou encore Plotly sont construits sous ce dernier. Dash permet donc de déployer les applications à travers un navigateur web.

Du côté du front end, React permet de générer les composants HTML qui permet de designer la page web. Pour le côté back end, toutes les propriétés sont configurable car Dash est une instance de Flask.

Pour créer cette l’application Dash, dans un premier temps on importe toutes les dépendances nécessaires au bon fonctionnement de l’application qui ont été installées préalablement à l’aide du requirement.txt .

Dans un second temps, il faut collecter les données dont nous avons besoin, par exemple dans notre cas nous récupérons des fonctions qui proviennent d’un autre fichier et qui ont été réalisées au préalable, tel que les transactions, le wallet et le wallet\_history. Il suffira ensuite d’en extraire les données et les visualiser sur l’application Dash de la manière préférée (Diagramme, Graphique , Diagramme circulaire..).

Une image contenant texte

Description générée automatiquement

Par la suite il faut créer une instance de la classe Dash qui prend en paramètre plusieurs données importantes dont notamment le titre de l’onglet de la page web, le style de l’application en background et le serveur qui est ici définit avec Flask.

Une image contenant texte

Description générée automatiquement

Vient ensuite la partie la plus délicate, on créer toutes les figures destinées à être affichées ainsi que les différentes pages. Une page Dash peut être divisée en colonnes et en lignes afin de correctement séparer les différentes figures et de les afficher de la façon souhaitée. Pour se faire il suffit d’utiliser des ‘dcc.Row’ pour les lignes et des ‘dcc.Col’ , provenant de la bibliothèque Dash Cor Components, pour définir les colonnes et fournir l’élément Graph qui permet d’inclure des graphiques Plotly .

De cette façon les figures sont imbriquées dans plusieurs indentations afin d’être positionnées à l’emplacement souhaité. Des couleurs et du style peuvent être rajoutés pour chaque figure. Dans notre cas, nous avons dû créer des variables afin de créer plusieurs pages, pour afficher la page d’accueil, de login, le formulaire de création de compte et le Dashboard. Tous ces composants regroupe l’affichage des graphiques et sont regroupés dans un layout.

Une image contenant texte

Description générée automatiquement

Pour que le Dashboard ne soit pas statique mais qu’il s’actualise en permanence et que l’utilisateur puisse modifier les graphiques selon ses envies il faut utiliser un système de décorateur qui est définit après le layout des graphiques. Le décorateur dash appelé app.callback contient des paramètres qui font le lien entre les composants HTML du front end qui doivent être lu/modifiés .

En back end, la fonction callback construit la figure en fonction du paramètre passé en argument et la retourne. Ce Callback prend en paramètre un Output, qui correspond au composant HTML que l’application va modifier dynamiquement, et un Input, prend en paramètre une modification de la propriété du widget. La valeur prélevée dans l’Input est déclarée Output et le graphique est mis à jour.

Une image contenant texte

Description générée automatiquement

Ensuite deux lignes sont nécessaires pour lancer l’application. Dans notre cas, nous avons créé un nouveau fichier .py nommé ‘main’ qui récupère la fonction ‘def launch app()‘ comme écrit sur l’image au-dessus et lance l’application après avoir lancé le code python dans un terminal.

Voici quelques images du dashboard et des pages qui s’affichent avec le code dash réalisé en important les bonnes data et en organisant bien le code dans l’ordre :

**PHOTO DAHSBOARD 1**

**PHOTO DASHBOARD 2   
  
PHOTO DASHBOARD 3**

Visualisation des données (Déploiement sur Server)

Pour que notre application soit accessible à tous et depuis n’importe quel appareil, nous avons décidé de la déployer sur la plateforme Heroku.

 Heroku est une plateforme basé sur le Cloud, qui apporte une solution aux développeurs pour convertir leurs idées en URL de manière simplifier et rapide. Heroku rend les processus de déploiement, de configuration, de mise à l’échelle, de réglage et de gestion des applications aussi simples et directs que possible. En plus de cela Heroku permet la création et l’hébergement de Base de Donnée, que nous développerons plus tard.

A l’aide de Git et Github nous avons pu déployer facilement notre application qui est accessible à l’adresse : A DEFINIR.

Base de Donnée

Pour rendre notre application encore plus fonctionnel, et plus simple pour tous nos utilisateurs. Nous avons créé une base de données Heroku, que nous avons lié à notre application. En effet, nous avons voulu que nos utilisateurs, lors que leur connexion à notre site, récupéré directement toutes les informations de leurs portefeuille conservé par notre base de données.

Pour cela nous avons utilisé encore une fois heroku. En effet, cette plateforme offre une haute disponibilité et un basculement automatique avec son service Heroku PostgreSQL. Ce système permet aux développeurs d’étendre leurs applications à l’aide de modules complémentaires.



La base de données PostgreSQL est une base de données orientée colonne. En effet, c’est un système qui stock les tableaux de données par colonne et non par ligne. Cette orientation permet un accès plus efficace aux données pour interroger un sous-ensemble de colonnes. De plus la compression par colonne est plus efficace lorsque les données de la colonne se ressemblent.

Nous avons donc implémenté une base de données PostgreSQL, que nous gérons à l’aide du dossier database dans notre projet. En effet, avec l’utilisation de SQLAlchemy, l’implémentation d’une base de données SQL est simplifié.

Pour gérer les données de nos clients nous avons créé deux tables : Une table Users et une table Potofolio. Les deux tables sont reliées par l’id du user. A l’aide de cette base de données l’utilisateur aura juste à se connecter à son compte à l’aide de son adresse mail et de son mot de passe définit lors de la création de son compte.

Une fois son compte créer l’utilisateur pourra ajouter son portefeuille crypto à son compte, qui sera stoker dans la table Portofolio et relié à son compte User à l’aide de son id définit en tant que Primary Key et implémenter automatique.

Nous donnons la possibilité à nos utilisateurs d’avoir plusieurs portefeuille crypto relié à leur compte, cela se fait donc en créant une nouvelle ligne à la table Portofolio, et un ajout de lien entre la ligne du user à celle de ces portefeuille dans la table Portofolio.

La base de donnée est donc une caractéristique importante pour notre projet, car dans un domaine aussi rapide que celui de la cryptomonaie, chaque minute est importante c’est pour cela que nous avons voulu éviter à nos utilisateurs de devoir rajouter manuellement à notre application leurs portefeuilles.

## Marketing

## Differenciation

# Sitographie

* <https://www.redhat.com/fr/topics/api/what-are-application-programming-interfaces#:~:text=Une%20API%2C%20ou%20interface%20de,int%C3%A9gration%20de%20logiciels%20d'applications>.
* <https://www.covalenthq.com/docs/>
* <https://fr.wikipedia.org/wiki/GitHub>
* <https://www.coingecko.com/fr/api/documentation>
* <https://www.redhat.com/fr/topics/devops/what-is-ci-cd?sc_cid=7013a000002pwNsAAI&gclid=CjwKCAjw7cGUBhA9EiwArBAvorI4jLoeSIBztn6IafGxIjTUJgtG5IOtXRtG4m4nRo0v4b8uh5na4BoCaXAQAvD_BwE&gclsrc=aw.ds>
* <https://docs.pytest.org/en/7.1.x/>
* <https://www.objetconnecte.com/heroku-tout-savoir/>
* <https://www.objetconnecte.com/heroku-tout-savoir/#Principales_caracteristiques_de_Heroku>
* <https://flask.palletsprojects.com/en/2.1.x/>
* <https://dash.plotly.com/>
* <https://getbootstrap.com/>