Solution design document

# Technology stack

* Python 3
* Flask as web framework
* Flask-PyMongo as ORM
* Docker as Container
* MongoDB as a database
* Pytest for unit test
* Postman for API test

# Current file structure

| .gitattributes

| .gitignore

| app.py

| Dockerfile

| hello.sh

| LICENSE

| MAKEFILE

| output.doc

| README.md

| requirements.txt

|

+---.vscode

| settings.json

|

+---Database

| | db.py

| |

| \---recipes\_raw

| LICENSE

| recipes\_raw\_nosource\_fn.json

|

+---Design

| Solution design document.docx

|

+---Model

| | user.py

| |

| +---Schema

| | feedbacks\_schema.txt

| | Meal\_plan\_schema.txt

| | Recipes\_schema.txt

| | Users\_schema.txt

| |

| \---\_\_pycache\_\_

| user.cpython-37.pyc

|

+---Postman\_Collection

| Hellofresh.postman\_collection.json

|

+---Settings

| | security.py

| | settings.py

| |

| \---\_\_pycache\_\_

| security.cpython-37.pyc

|

\---Tests

.env.test

BaseCase.py

Test\_Get\_User.py

\_\_init\_\_.py

# Project Setup

## **Initialisation**

You can begin by either downloading a zip file of the project through github, or using a git command to clone the project by:

git clone <https://github.com/Sanchitakr/My_Hellowfresh/tree/finetuned>

## **Virtual Environment Setup**

It is preferred to create a virtual environment per project, rather then installing all dependencies of each of your projects system wide. Once you install virtual env, and move to your projects directory through your terminal, you can set up a virtual env with:

virtualenv venv -p python3.7

This will create a python3.7 based virtual environment (venv) for you within your projects directory.

Note: You need to have Python 3.7.4 installed on your local device.

## **Dependency installations**

To install the necessary packages:

source venv/bin/activate

pip install -r requirements.txt

This will install the required packages within your venv.

## **The Makefile**

When you are in the project directory on your terminal, you can use the make command for various options such as generating a new requirements.txt file, installing requirements on your venv or cleaning old compiled python .pyc files.

# **Database configurations**

These are used for configuring your MongoDB database.

SECRET\_KEY: Flask's secret key which is used for hashing and security purposes. Make sure to keep this secret and don't commit to github.

The configurations are carried to code through the decouple library, which IMO provides a better solution compared to the traditional python-dotenv library.

# Project Structure

### **Main Modules**

Every flask application has a top-level module for creating the app itself, in this case, this module is the app.py. This contains the flask application, and is used by other services such Flask's CLI while serving the application.The app.py module relies on the Setting/security.py and the app/\_\_init\_\_.py modules. The file app.py uses one of the configs specified to create an application through the Flask(\_\_name\_\_) method, which is placed under the app.py module.

The file app.py ties the necessary packages such as your PyMongo or Flask-Restplus libraries to your app, and provides a nice function for generating an application with a pre-specified config.

### **Models**

The models, which are your database objects, are handled through Flask-PyMongo ORM. Flask provides a wrapper around the traditional Flask-PyMongo package, which is used through out this project.The models created are similar to your regular Python classes. They are inherited from pre-specified Flask-PyMongo classes to make database table creation processes easier. These collection schema can be found under the app/models/schema folder.

### **API**

The project creates a simple API which has 4 endpoints namely create, list, read, update, delete data model objects. The API is structured by using Flask's blueprint functionality.

app.py module creates the endpoints, where the init\_\_.py creates the blueprint for API formation.

### **Database Choice & Operations**

Usually for flexible projects, databases such as MongoDB is preferred for the ease of use. For the current project I have used MongoDB atlas to connect to MongoDB clusers.

### **Security Guidelines**

1. Any PI & sensitive data such as a user password will be encrypted using standard encryption techniques in the system.
2. JWT/API Key security mechanism implemented in the current application for securing the application endpoints

# Running the Application

Once you have setup your database, you are ready to run the application.

You can go ahead and run the application with a simple command:

flask run

You can also run your app using simple run button in the VS-Code at the right hand side, which will run the application as same as the above command.

# Docker Setup

The project also has docker functionality, which means if you have docker installed on your computer, you can run the application using Docker as well!

Docker creates containers for you, and basically serves your application using these containers. The necessary setting files for the docker setup can be found in Dockerfile it self.

In this case, docker uses a prebuilt Python3.7.4 image that runs on Windows, and MongoDB database containers to serve the application.

To run Dockerfile:

docker build -t hellofresh .

Once its built, to run the docker image:

docker run hellofresh

If there are any errors or module import errors simply run

docker run hellofresh ls ./

which shows the files present in the docker image.

Docker becomes especially useful while deploying your applications on servers, and makes the DevOps easier. You can read more about docker.

# Tests