AACE Platform

Project diary

This document will contain all the steps and processes involved in building the AACE platform. This is intended to be used as a handbook that will serve as an entry point for future references. More documents will be attached for a complete overview.

In addition to reporting on the steps taken for this particular project, this document also seeks to be a guide for further projects. For AACE we have tried to generalize the process of building an API based backend and frontend for any web or mobile application.

AACE will be built using two main technologies. Flask for server side, and VueJS for client side. APIs will be served using flask, and mobile or web clients will consume that API to produce SPA or mobile applications. For this reason, we need to scaffold the application in two main directions.

The backend (and frontend?) will be containerized with Docker.

# Backend scaffolding

The backend scaffolding will include setting up and configuring a Flask application for use with API.

## 1.1 Basic flask application setup procedure

The first step to take when dealing with a new project that will require Flask programming, is to open a repository in BitBucket or any other code management tool. Since we use BitBucket, we open a new project.



Fig 1.1

AACE project repository in Bitbucket

For this kind of job, it makes sense to open a project repository for two reasons:

1. All team members can have access to the code and therefore can perform operations from their own accounts
2. The project might require different repositories for different aspects of the work (Mobile, Web, Landing Page, etc)

The first repository is named ‘aace-platform’. The repository is cloned locally and the first commit is a folder containing, among other things, this document. It is advisable that every project repository is accompanied by a report of some sort that serves as a guide to the code.

The second addition to the project that is suggested at the time of writing is to include a .gitignore file. This file needs to be added before scaffolding a Flask project to avoid complications with caching and other issues that follow mid-project .gitignore tentatives. This file will be copied from the Flusk repository where we will adapt the general architecture. In addition to the Flusk .gitignore defaults, add the .vscode line.

The third addition is to add a virtual environment for python. Having an isolated environment protects the application from different requirements among projects residing in the local development machines.

The resulting folder should look similar to Fig 1.2.

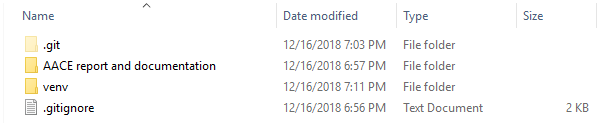


Fig 1.2

Documentation, venv and gitignore

Having prepared a clean environment, we are ready to proceed with the architecture of the application itself. For further reference, most of the steps taken in this report are directly or indirectly influenced by the famous Grinberg tutorial on Flask. This point is also appropriate for a commit with a clean description, to easier track changes and progress.

Now activate the virtual environment and install only the necessary packages, to avoid conflicts and bloated space requirements. So far we only need the flask package, done with ‘pip install flask’. This will install 6 required packages for a basic flask application. These packages need to be saved in a requirements.txt file, so we execute ‘pip freeze > requirements.txt’. The result should look similar to Fig 1.3.

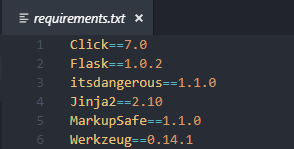


Fig 1.3

Initial pip freeze requirements

Before starting with the Flask setup, we need to create a folder called backend. All the steps taken now on are assumed to be executed within this folder.

Flask applications exist in packages. A package is a sub-directory that includes an ‘\_\_init\_\_.py’ file. A package may contain many modules, which are usually python files. The init file defines what inside the package is exposed to outer uses. Our package directory will be called ‘api’. We create this directory and the init file in it.

*from* flask *import* Flask

app = Flask(\_\_name\_\_)

What needs to be considered here is that putting ‘\_\_name\_\_’ as the argument to Flask() usually configures the application instance right. The created app object now holds our Flask application.

To complete the basic layout, we need to create some environmental variables. We first install the dotenv python package using: pip install python-dotenv. Afterwards update the requirement file with pip freeze.

Create a .flaskenv file where we will store our env variables to configure our application. Create a run.py file in the main directory. This file needs to contain only one line for a basic functioning:

*from* api *import* app

Write down the following on the .flaskenv file:

FLASK\_APP=run.py

You can now execute ‘flask run’ on the terminal. The application is working. This is another good milestone that needs to be clearly committed. The project tree should look similar to the image below.

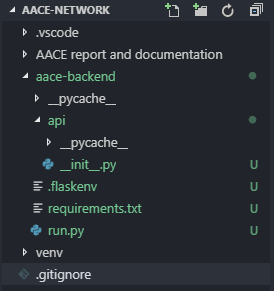


Fig 1.4

Project tree

## 1.2 Merging the basic structure with the Flusk structure

Now that the basic structure is in place, we can proceed to integrate all components found in the Flusk project. The first step in this direction is to install the necessary packages required by the larger flusk layout. After installing the required packages, pip freeze the new requirements.

The second step is to copy all the material from within the flusk ‘api’ folder to our own ‘api’ folder. Then, we have to update our run.py to the following code:

*from* api *import* create\_app

app = create\_app()

While this procedure makes the application start, it is still missing some additional configuration found on the tutorial blog.

The next import will be to copy the migration folder and put it in our backend directly. After making these changes our backend directory should look like this:

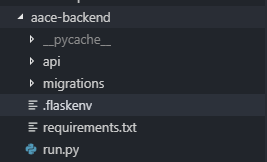


Fig 1.5

Imported api and migrations from flusk

It is important we set the flask environmental variables as needed to be able to run the ‘flask run’ command. To run the server, we also need to configure the database connection URL. This configuration can be changed at api\common\database.py, although it is recommended that these credentials be provided by flask environmental variables. Having given the right credentials of the database, we can now run the flask application. However, making API calls through tools such as POSTMAN will give us responses with a 401 error code, invalid permissions. To mend this for the development process, we have to comment out the code found in the middleware \_\_init\_\_.py

*request.ensure\_public\_unavailability,*

Commenting out this code will make our content publicly available. The full range of features and boilerplate code of flusk are now ready to be modularly adopted.

# Frontend scaffolding

For the frontend we will use VueJS framework. The steps to configuring VueJS are readily available online, but are also replicated here.

## Setting up VueJS

The first step is to create the ‘aace-frontend’ folder where everything related to the frontend client will live. If we already have Vue-CLI installed, we can proceed to create a vue project using the command line in the frontend folder with the following command:

vue create client

This will create a client folder where all the vue files will be scaffolded. Vue CLI will create its own gitignore file, and for safety we should include its contents to the main gitignore file. The full layout so far should be the following:

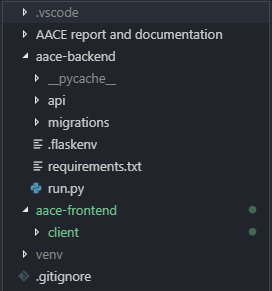


Fig 2.1

The layout including VueJS

The commit directory is also well readable and understandable.

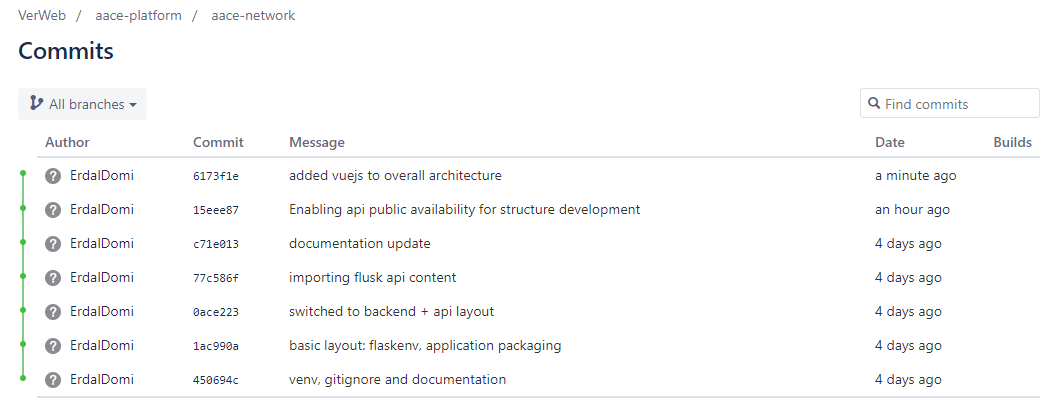


Fig 2.2 Commit directory

The latest commit can serve as a starting point for future projects, although more modifications will be done through the project to the initial template.