Contents

[First web-based approach – Python Flask and Bootstrap: 2](#_Toc99048903)

[Why I decided to change from a GUI to a Web Application: 2](#_Toc99048904)

[Creating a virtual environment using pip: 2](#_Toc99048905)

[Installing all modules and dependencies for the web application: 2](#_Toc99048906)

[Creating the directories and files for the program: 3](#_Toc99048907)

[Setting up the website directory: 3](#_Toc99048908)

[Creating the files that will be needed for the application: 3](#_Toc99048909)

[Setting up the application: 4](#_Toc99048910)

[Creating the views: 4](#_Toc99048911)

[Creating the base template: 4](#_Toc99048912)

[Base.html: 4](#_Toc99048913)

[Home.html: 5](#_Toc99048914)

[Registering the blueprints to the application: 5](#_Toc99048915)

[Updated version of “\_\_init\_\_.py”: 5](#_Toc99048916)

[Running main.py to check everything is working and initialised properly: 5](#_Toc99048917)

[Creating the login page (front-end): 5](#_Toc99048918)

[Adding some extra pieces to “base.html”: 5](#_Toc99048919)

[Second web-based approach – Python Flask and TailwindCSS: 6](#_Toc99048920)

[Why I decided to change from bootstrap to tailwind: 6](#_Toc99048921)

[Preparing the workspace: 6](#_Toc99048922)

[Creating the virtual environment: 6](#_Toc99048923)

[Entering the virtual environment: 6](#_Toc99048924)

[Installing the python packages needed: 6](#_Toc99048925)

[Creating the files and directories I need to begin this application: 6](#_Toc99048926)

[Installation and Setup of TailwindCSS: 7](#_Toc99048927)

[Setting up the main CSS file to inherit Tailwind’s classes: 8](#_Toc99048928)

[Building the application: 8](#_Toc99048929)

[Initialising the Flask Application: 8](#_Toc99048930)

[Creating a test blueprint: 9](#_Toc99048931)

[Building the home page: 10](#_Toc99048932)

[Setting up the route: 10](#_Toc99048933)

[Start of development based on designs: 10](#_Toc99048934)

# First web-based approach – Python Flask and Bootstrap:

This section documents my first initial approach to a web-based application using python’s flask and the CSS framework called bootstrap. This method didn’t make it past the initialisation stage, the reasoning behind this will be explained later on in the document.

## Why I decided to change from a GUI to a Web Application:

I took the decision to change my project from a GUI-based application to a Web-based application because with a web-based app you are offered a lot more freedom in terms of design and features in my opinion. For example, the HTML and CSS aspects alone of the web application allow for great design opportunities compared to that of a pythonic GUI. Furthermore having the ability to merge HTML, CSS and JS with Python allows for many more front-end features for the application which benefits the overall user experience whilst using the application and also provides support for all devices, since it will be accessible via a browser rather than a python script and/or executable.

## Creating a virtual environment using pip:

Text

Description automatically generated with medium confidenceFirst of all, I create a virtual environment by opening command prompt in the directory I’m going to be making the web application in and write the command “pipenv –python 3.10.1” to create a virtual environment in python version 3.10.1.

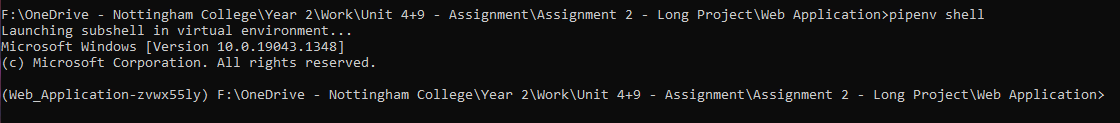
## Installing all modules and dependencies for the web application:

Text

Description automatically generatedText

Description automatically generatedA picture containing text

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidenceUp next I need to install all of the modules I will be using for this application. To do this, I start a shell within the virtual environment and install everything as seen below:

Text

Description automatically generated

## Text Description automatically generatedCreating the directories and files for the program:

To begin with, there are 2 things that need to be created before doing anything else. I need to create the “main.py” file which is the file that will be run and starts the entire web application up, and the “website” directory which will contain the main bulk of the code for the application itself. Also after creating the virtual environment and installing the dependencies, there are 2 files already in the main directory called “Pipfile” and “Pipfile.lock” which contain details on the environment such as packages and version requirements etc.

## Text Description automatically generated with medium confidenceSetting up the website directory:

For the website directory, there are a few sub-directories that I need to make which will all contain the front-end elements of the application (HTML, CSS and JS). These sub-directories will be called “templates” and “static”. Furthermore, inside the “static” sub-directory there will be 3 more sub-directories each called “styles” for CSS files, “js” for JavaScript files and “img” for any images. This sub-directory is called “static” because they contain static files, these files should not be changed at all once deployed. Additionally, the sub-directory “templates” speaks for itself in terms of its naming, you create templates which you later render in the python files which will be displayed later on in development.

## Text Description automatically generated with medium confidenceCreating the files that will be needed for the application:

This is where I create all of the python files which make the web application function, as well as all of the front-end files that the python files will render and utilise once the python functions have been written. The “\_\_init\_\_.py” file contains all of the main configuration and the setup of the application. The “auth.py” file will contain all of the code that makes the login function work, and handles requests from the login form that will be written in the front-end files, and again rendered by the python functions. The “database.db” file is as said in the file name, the database for this web application, now because this database is local it may pose as a security threat, however since the vital data such as a user password will be encrypted it won’t be an issue. The “models.py” file will contain the database classes that will be called in “auth.py” which allows us to write to the database. The “views.py” is essentially the same as “auth.py” however instead of containing the database functions and handling requests, its simply there to handle the rendering of the templates found in the front-end files. Finally, the front-end files that will be used and explained later on are “base.html”, “home.html”, “login.html”, “styles.css” and “main.js”.

## Setting up the application:

A screenshot of a computer

Description automatically generated with medium confidenceText

Description automatically generatedText

Description automatically generatedThis is where the actual back-end coding begins. The first file that will be written to is the “\_\_init\_\_.py” file, we need to initialise the application and some other things later on (because they aren’t need right now). Most of what is going on will be explained through comments in the screenshots provided below. The terminal screenshot displays the password generator I used for making the secret key for the application.

## Creating the views:

Graphical user interface, text

Description automatically generatedWhen it comes to Flask and web applications in particular, I need to create views that perform functions, their main use is to handle requests on specific pages and return a render template function that displays the front-end of the web page.

## Creating the base template:

When using python alongside html for a web application, you have the ability to write python-like code within the html files, this is specifically useful for creating blocks and using variables from the route functions. The blocks are best useful for creating base html files which act similar to a template, as shown below.

## Text Description automatically generatedBase.html:

## Text Description automatically generatedHome.html:

## Registering the blueprints to the application:

In order to utilise these blueprints (views), we need to register them in the “\_\_init\_\_.py” file, by doing this I import the “views.py” file and insert it into the “register\_blueprint” function alongside its “url\_prefix”. How that is done can be found in the screenshot below.

## Text Description automatically generatedUpdated version of “\_\_init\_\_.py”:

## Graphical user interface, application Description automatically generatedRunning main.py to check everything is working and initialised properly:

Up until now, after the first time running the main.py file, what we should find is the page title is “Home · College Tournament” and not “ · College Tournament”, and we should see a h1 header saying “Hello World!”. This is the result after running main.py and viewing the website on localhost.

## Creating the login page (front-end):

Since this application will only be available for the teachers to update, we need to have some kind of restriction in place to ensure students can’t get access to the website and alter scores and entities’ (teams or individuals) data. This will be in the form of a login page, with no signup option available (the login details will be pre-defined in the database) that only teachers will know and have access to. This will be a stylish but simple login form that any user will be prompted with, and the home page route and any other route we create within the application will use a special decorator that only allows logged in users to view. The screenshots below are showcases of the front-end code alongside some explanations of what certain pieces actually do and why I have used it.

## Adding some extra pieces to “base.html”:

The first element that I will be adding to the base is a navigation bar, this will later have some alterations to make the navigation and names cleaner and more concise but that will only be available to us once the login page, both front and back-end are complete.

# Second web-based approach – Python Flask and TailwindCSS:

## Why I decided to change from bootstrap to tailwind:

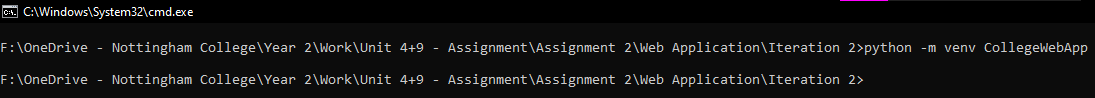
Graphical user interface, text

Description automatically generatedJust before I was about to fully commit to the front-end of this application, I actually discovered a better CSS framework that I could use that would benefit me more for this project in particular, called TailwindCSS. Despite this framework being more compatible with JS-based frameworks such as Vue, it is still applicable to a pythonic applications. The main feature of Tailwind that appeals to me and this project more than Bootstrap does, is the fact that Tailwind provides much better customisation and also has a few other features such as a VSC extension providing “intelli-sense”, making development easier, and better utility classes.

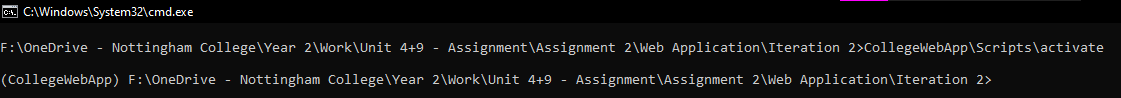
## Preparing the workspace:

Before creating this, I am keeping the first and current iteration of this project in the same directory, just separating them into 2 sub-directories to keep them in the same place. The previous iteration being named “Iteration 1”, and the second (current) iteration being named “iteration 2”.

### Creating the virtual environment:

For this iteration, I have changed from using “pipenv” for the virtual environment to “venv”. This serves the same purpose as Pipenv however it is easier to use.

### Entering the virtual environment:

To enter the virtual environment, it is extremely simple. I just need to run the command “CollegeWebApp\Scripts\activate” and I will be inside the virtual environment, completely ignoring any other packages I already have installed on my machine. You can tell that you have successfully entered the virtual environment when the name of the environment is in brackets to the left of the directory location in command prompt.

### Text Description automatically generatedInstalling the python packages needed:

Graphical user interface, text

Description automatically generatedInstalling the required python packages is extremely simple. The command is simply “pip install <package\_name>” and this will fetch the package from something called PyPi and install it to the virtual environment for me to use in my application. The packages I need to install for this project are the following:

* Flask
* Flask-login
* Flask-sqlalchemy

Note how I only install flask-login and flask-sqlalchemy. Since Flask itself is a dependency of both of these packages, they will automatically install Flask, and a few other dependencies if they are not present in the environment they are being installed to, removing the need for me to install Flask separately.

### Creating the files and directories I need to begin this application:

To begin with, there are only a few files and directories I need to make to have the structure of the application initialised. To begin with, to separate the application from the virtual environment that I created called “CollegeWebApp”, I will create a directory which will act as the application’s “root” directory called “app”. This will contain all of the files relating to this application. Text

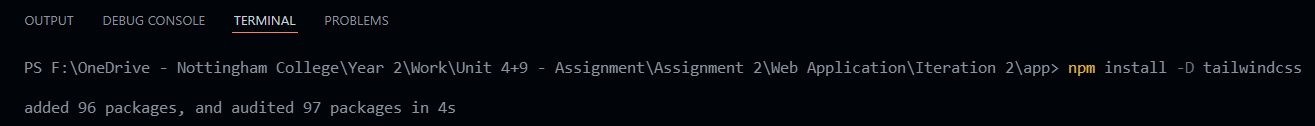
Description automatically generated

Using the windows command “dir /S” will display every file and directory inside the directory you run the command in. The screenshot above displays the root directory’s contents at the top, as well as the contents of the “website” directory at the bottom, all of which are accompanied with an explanation on their purpose for this application.

### Installation and Setup of TailwindCSS:

**IMPORTANT NOTE:** Node.js is required for the installation of TailwindCSS.

From this point onwards, any terminal usage I’ll be doing inside the built-in terminal on my IDE, Visual Studio Code (VSC).

First of all, once node.js is installed I need to install the CSS framework I’ll be using for this project, TailwindCSS. I can do this by running the following in the terminal.

“Added 96 packages” shows us that tailwindcss has been successfully installed, and to double check, I can check the newly created “node\_modules” directory in my workspace and check for the “tailwindcss” sub-directory, which is shown to the right.

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generatedNext I need to generate a configuration file for tailwind. Features like this in particular are what made Tailwind a more favourable option over bootstrap, its flexible configuration and vast amount of utility classes. Running “npx tailwindcss init” creates a file called “tailwind.config.js” which is where any configuration for tailwind will take place.

Graphical user interface, text

Description automatically generated

To finish off initial configuration, I just need to insert the file location of all of my template files, these being the HTML files. The screenshot below shows this.

### Graphical user interface, text, chat or text message Description automatically generatedSetting up the main CSS file to inherit Tailwind’s classes:

First of all, I need to create the CSS file that will be holding all of tailwind’s classes. I will be creating a file called “input.css” in the “website/static/styles” directory and adding 3 short lines of code that when they are run, will import all of Tailwind’s classes.

Finally, in order to have these classes imported, I need to run the following command in the terminal: “npx tailwindcss -i ./website/static/styles/input.css -o ./website/static/styles/styles.css --watch”

Text

Description automatically generatedAs you can see below, the above command has been run and we now have a new file called “styles.css” which now contains all of the imported TailwindCSS classes. “input.css” is no longer needed so that file will be deleted and now the setup of both Tailwind and the workspace environment is complete.

## Building the application:

### Initialising the Flask Application:

The file structure will be slightly different compared to the previous iteration of this web application. Upon doing further research into Flask applications I have discovered better ways to utilise this package and this will be documented throughout this project.

Text

Description automatically generatedTo begin with, I need to head into the “website” directory and enter the file called “app.py” which will contain the class that manages the entire application.

This snippet here is the very bare bones, basic initialisation of the Flask application. At the minute this application will do entirely nothing, but can still be run and will create a local development server as seen below.

Text

Description automatically generated

### Creating a test blueprint:

In Flask terminologies, a blueprint is essentially something we register to our application that contains a collection of routes and other app-related functions, a blueprint is simply an object that allows you to define application functions, without requiring the application object (the class I defined earlier inside “app.py”) at the time of definition.

First of all, I need to create a new file called “views.py”, this file will contain a singular blueprint that will handle all web pages that are not related to logging in to the application (log-in page). They will be put in a separate file later on in development of the application. The purpose of each element in the screenshot below is explained with a corresponding comment.

A screenshot of a computer

Description automatically generated with medium confidence

#### Assigning the test blueprint to the application:

Text

Description automatically generated

Graphical user interface, application

Description automatically generatedNow if I run the application, and go to <http://localhost:5000/> in my browser, it should render the HTML that I wrote in the test function inside “views.py”, as seen below. (Screenshot on the right is from inspect element, to check correct HTML elements have been pushed)

Graphical user interface, text

Description automatically generated with medium confidence

At this point, everything is working as intended, so now it is time to start building the home page, this will also be where we can verify that the installation of Tailwind is properly successful by attempting to use the classes that have been imported into “styles.css”.

## Building the home page:

*\*A full list of tools, and links to them where necessary, can be found near to the end of this document.*

### Setting up the route:

The home page is going to have a number of features, as shown in the designs however like any other project, they are bound to have reworks or changes made to them mid-development. Any changes that are made from the initial designs will be documented and justified towards the end of this document. Before the development of the home page begins though, a “home.html” file needs to be created inside of the “templates” directory, and it needs to be linked to a route inside the newly created “views” blueprint.

Text

Description automatically generated

*\*Note: the test route has now been changed to “/test” and home takes “/”.*

To ensure the routes have been switched properly and the new “home.html” template is being rendered, I will add some markdown to the HTML file, differentiating from the html defined in the test function to make sure everything is updated.

Graphical user interface, application, chat or text message

Description automatically generatedA helpful feature from Flask is that I don’t need to stop, and restart the application after every change, since the application is in development mode, it automatically updates the contents of the app after every save of the code. The updated “/” route can be seen in the screenshot on the left, and the usage of “/test” can be seen on the right.

Graphical user interface, application

Description automatically generated

The test function is no longer needed so I’m going to remove that. From now is where the actual HTML based on the designs produced begins.

### Start of development based on designs:

#### Preparing the home.html file:

Before the elements are created, the html file needs some tags and external files to be linked as seen below. You’ll notice that I referenced the CSS file slightly differently than a normal CSS file would be. This has nothing to do with it being Tailwind, it’s actually due to the template tool that Flask uses to render the HTML, called Jinja2. It is also what allows me to use python in the HTML file later.

Text

Description automatically generated

#### Navigation Bar:

This is a preview of the navigation bar in the designs I had produced:

