**Prerequisites**

To complete this project, you will need the following:

* Some familiarity with Python.
* Python installed on a local environment/ cloud environment.
* Knowledge of Basic Linux Navigation and File Management.

Here is a diagram to provide a sense of what the file structure of the project will look like once you have completed the tutorial:

Text

Description automatically generated with medium confidence

**Step 1 — Installing Packages**

There are three main packages you need for your project:

* Flask
* Flask-Login: to handle the user sessions after authentication
* Flask-SQLAlchemy: to represent the user model and interface with the database

You will be using SQLite to avoid having to install any extra dependencies for the database.

First, start with creating the project directory:

mkdir cloud\_flask\_app

Next, navigate to the project directory:

cd cloud\_flask\_app

You will want to create a Python environment if you don’t have one.

Depending on how Python was installed on your machine, your command will look like:

sudo apt-get install python3-venv

Python3 -m venv nasa

The -m flag is for module-name. This command will execute the module venv to create a new virtual environment named nasa. This will create a new directory containing bin, include, and lib subdirectories. And a pyvenv.cfg file.

Next, run the following command:

source nasa/bin/activate

This command will activate the virtual environment.

Run the following command from your virtual environment to install the needed packages:

nano requirements.txt

copy paste the contents of requirement.txt files and save.

pip install -r requirements.txt

Now that you have installed the packages, you are ready to create the app.

**Step 2 — Creating the Main App File**

Let’s start by creating a project directory:

mkdir project

The first file will be the \_\_init\_\_.py file for the project:

This file will have the function to create the app, which will initialize the database and register the blueprints. Now, this will not do much, but it will be needed for the rest of the app.

You will need to initialize SQLAlchemy, set some configuration values, and register the blueprints here:

nano project/\_\_init\_\_.py

Text

Description automatically generated

**Step 3 — Adding Routes**

For the main\_blueprint, the main blueprint will be used to run the application.

First, create main.py:

nano main.py

Timeline

Description automatically generated

Next, create views.py:

For the routes, you will use two blueprints.

For the views\_blueprint, you will have a home page (/) and a profile page (/profile).

nano project/views.py

Text

Description automatically generated

For the auth\_blueprint, you will have routes to retrieve both the login page (/login) and the sign-up page (/signup). Finally, you will have a logout route (/logout) to log out an active user.

Next, create auth.py:

nano project/auth.py

Then add your auth\_blueprint:

Text

Description automatically generated

Text

Description automatically generated

**Step 4 Creating Templates —**

Next, create the templates that are used in the app. This is the first step before you can implement the actual login functionality.

The app will use four templates:

* index.html
* profile.html
* login.html
* signup.html

You will also have a base template that will have code common to each of the pages. In this case, the base template will have navigation links and the general layout of the page.

First, create a templates directory under the project directory:

mkdir -p project/templates

Then create base.html:

nano project/templates/base.html

Next, add the following code to the base.html file:

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

This code will create a series of menu links to each page of the application. It also establishes a block for content that can be overwritten by child templates.

Next, create templates/index.html:

nano project/templates/index.html

Add the following code to the newly created file to add content to the page:

Graphical user interface, text

Description automatically generated

This code will create a basic index page with a title and subtitle.

Next, create templates/login.html:

nano project/templates/login.html

This code generates a login page with fields for Email and Password. There is also a checkbox to “remember” a logged in session.

Text

Description automatically generated

Next, create templates/signup.html:

nano project/templates/signup.html

Add the following code to create a sign-up page with fields for email, name, and password:

Text

Description automatically generated

Next, create templates/profile.html:

nano project/templates/profile.html

This is where we will display the data fetched for the NASA APOD API

Text

Description automatically generated

**Step 5 — Creating User Models**

The user model represents what it means for the app to have a user. This tutorial will require fields for an email address, password, and name. In future applications, you may decide you want much more information to be stored per user. You can add things like birthdays, profile pictures, locations, or any user preferences.

Models created in Flask-SQLAlchemy are represented by classes that then translate to tables in a database. The attributes of those classes then turn into columns for those tables.

Create the User model:

nano project/models.py

Define the User model:

Text

Description automatically generated

This code defines a User with columns for an id, email, password, and name.

Now that you’ve created a User model, you can move on to configuring your database.

**Step 6 — Configuring the Database**

You will be using an SQLite database. You could create an SQLite database on your own, but let’s have Flask-SQLAlchemy do it for you. You already have the path of the database specified in the \_\_init\_\_.py file, so you will need to tell Flask-SQLAlchemy to create the database in the Python REPL.

Ensure that you are still in the virtual environment and in the flask\_cloud\_app directory.

If you stop your app and open a Python REPL, you can create the database using the create\_all method on the db object:

python

from project import db, create\_app, models

db.create\_all(app=create\_app())

exit()

You will now see a db.sqlite file in your project directory. This database will have the user table in it.

**Step 7 — Setting Up the JavaScript for fetching data from Nasa APOD API**

One of the most popular websites at NASA is the [Astronomy Picture of the Day](http://apod.nasa.gov/apod/astropix.html). In fact, this website is one of the [most popular websites](https://analytics.usa.gov/) across all federal agencies.

HTTP Request

GET https://api.nasa.gov/planetary/apod

concept\_tags are now disabled in this service. Also, an optional return parameter *copyright* is returned if the image is not public domain.

Query Parameters

| Parameter | Type | Default | Description |
| --- | --- | --- | --- |
| date | YYYY-MM-DD | *today* | The date of the APOD image to retrieve |
| start\_date | YYYY-MM-DD | none | The start of a date range, when requesting date for a range of dates. Cannot be used with date. |
| end\_date | YYYY-MM-DD | *today* | The end of the date range, when used with start\_date. |
| count | int | none | If this is specified then count randomly chosen images will be returned. Cannot be used with date or start\_date and end\_date. |
| thumbs | bool | False | Return the URL of video thumbnail. If an APOD is not a video, this parameter is ignored. |
| api\_key | string | DEMO\_KEY | api.nasa.gov key for expanded usage |

Example query

<https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY>

Text

Description automatically generated

**Step 8 — Run the application**

The FLASK\_DEBUG environment variable is enabled by setting it to 1. This will enable a debugger that will display application errors in the browser.

Ensure that you are in the flask\_cloud\_app directory and then run the project:

python3 main.py

Now, in a web browser, you can navigate to the five possible URLs and see the text returned that was defined in auth.py and views.py.

For example, visiting localhost:5000/ displays: Home:

Graphical user interface, text, application

Description automatically generated

Conclusion

We built a login system for an app using Flask-Login and Flask-SQLAlchemy in this app. By initially constructing a user model and saving the user's information, we have demonstrated how to authenticate a user. Then we had to check that the user's password was correct by hashing it and comparing it to the one saved in the database. Finally, we introduced authorisation to the app by using the @login required decorator on a profile page to restrict access to only logged-in users.

For simple apps, the code you wrote in this article will suffice, but if you want more functionality right away, you might consider using the Flask-User or Flask-Security libraries, which are both built on Flask. Finally, we have have demonstrated the use of Nasa APOD API to display picture of the day and randomly generated picture of data by passing a random date to the API.