**Personal Portfolio Website using Cline**

**Cover**

Creating a Personal Portfolio Website using Cline

Languages: Python, SQL, CSS, HTML, Javascript

Frameworks: Flask, SQLite, Jinja, Bootstrap, Font Awesome

Tools: Cline

**# Why do it?**

There’s lots of website and portfolio builders out there (Squarespace, Notion etc). But the purpose of this process is to improve my technical skills including Python, building a better understanding over time of what’s happening under the hood, and learn how to direct AI to do this process efficiently.

**# Initial Prompting**

To get familiar with using AI coding agents I built this website using Cline. In an hour or so the site (hosted locally) was up and running. The instructions were simple – it needed:

* A landing page
* Functionality for the owner to login and upload projects to be displayed on that page
* Flask-based setup (as I’m most familiar with Python) with Bootstrap (as I’ve used it before)
* A simple design and code base (so I can understand what’s been done)

This initial stage was done smoothly. The virtual environment wasn’t created properly so a start.bat file was created to automatically launch the virtual environment and start the flask server. The challenges came when it was asked to format the front page (move the image and name in the header bar) – whether these layout issues were a function of Cline’s limitations of those of Bootstrap would become clear.

**# Architecture**

Responding to a follow-up prompt, Cline produced a diagram of the architecture in Mermaid.

An application folder is setup that contains:

* **models.py:** python classes are created, each mapped to a SQLAlchemy database (using db.Model) for:
  + The **User**, including:
    - Ability to test whether the user is active, authenticated etc
    - Personal details, password, contact details etc
    - Associated functions to set and check the password (using werkzeug to generate hash)
  + **Projects**, including:
    - ID, title, description, content, creation date, publication date etc
  + The classes all include a special method only for the benefits of developers: \_\_repr\_\_ – it defines what the class calls itself if you print it (the name of the user)

Outside this application folder, multiple files are set-up including:

* **.env:** an environment file that contains the flask configuration, database url, admin credentials (username, password, email) and social links
* **config.py**: this uses the dotenv python library to load to a class ‘Config’ the details set up in the .env file, as well as other settings such as for uploads (size, folder, extensions)

Back to the application folder, files are added:

* **forms.py:** sets up classes for the various forms required (login, profile, projects)
* **routes.py:** 
  + User, db and Project are imported from models.py
  + The various forms are imported from forms.py
  + Functions are create to check and save files
  + A Blueprint is created, in which routes are saved (rendering templates) for the home page, about page, project page, login/logout, editing personal information and project upload… (and error handling)
* **\_\_init\_\_.py:** this special method marks the directory as a python package so the User and SQLAlchemy can be imported from model.py
  + Flask and the configuration file are also imported
  + An instance of the flask app is created, the configuration settings are loaded
  + Login messages are set

Folders are also included for templates (html) and static files defining formatting and containing user uploads:

* The template pulls in content from boostrap and fontawesome to speed things up
* Jinja enables the base template to be filled in by additional templates created

Back outside the application folder,

* run.py is setup to initialise the database and create an admin user, using the config file.
  + This also runs the app is the script is run not as a module.
* Other files are built as standard – requirements (modules and versions needed), README, .gitignore (to avoid uploading everything including large database files to GIT)

**A diagram of a company

AI-generated content may be incorrect.**

**A screenshot of a diagram

AI-generated content may be incorrect.**

**A diagram of a software system

AI-generated content may be incorrect.**

**What it looks like:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**