Automated Transfer Credit Evaluator
Milestone 4 Evaluation

Tyler Dionne & Kendall Kelly

Milestone 4 Overview

- Converted a static GitHub Pages website to a dynamic Flask application
- Set up proper project structure with templates, static files, and database
- Prepared for user authentication functionality

Task 1: Flask Project Structure Setup

- Clone the github repository
  - $ git clone https://github.com/tylerdionne/ATCE-FIT
- Create virtual environment
  - $ python3 -m venv venv
  - $ source venv/bin/activate
- Install Flask
  - $ pip install Flask

Task 2: Set Up Static Files

- Set up a static directory inside of the project directory
  - $ mkdir static
  - $ mkdir static/css static/js static/images
- Move all static files to new directory
- Separate css from files:
  - Create css file for each html file in /static/css
  - Copy css code inside the <style> tag in the html file and paste it into the new css file.
  - Replace the <style> block in the html with a <link> tag inside the <head> section. (ex. <link rel="stylesheet" href="/static/about.css">)

Task 2: Set Up Static Files (Cont.)

- Move all javascript (only in atce.html) to the static directory
    - Create a file atce.js in /static/js
    - Move everything inside <script> tag into this file
    - Reference the javascript file in the html (ex. <script src="/static/js/atce.js"></script>)
- Move all documentation for each milestone from docs page into this folder by using a /static/docs folder
- Move images to this folder using a /static/images folder. Our only image is logo.png

Task 3: Convert Static HTML to Flask Templates with Jinja2

- Create /templates directory in project directory and move html files to /templates:
  - $ mkdir templates
  - $ mv *.html templates/
- Prepare html files with Jinja2 templating:
  - Example 1: Home
    - Home
  - Example 2: <img src="logo.svg" alt="Logo" style="width:150px; height:auto;">
    - <img src="{{ url_for('static', filename='logo.svg') }}" alt="Logo" style="width:150px; height:auto;">
  - Example 3: <script src="/static/js/atce.js"></script>
    - <script src="{{ url_for('static', filename='js/atce.js') }}"></script>

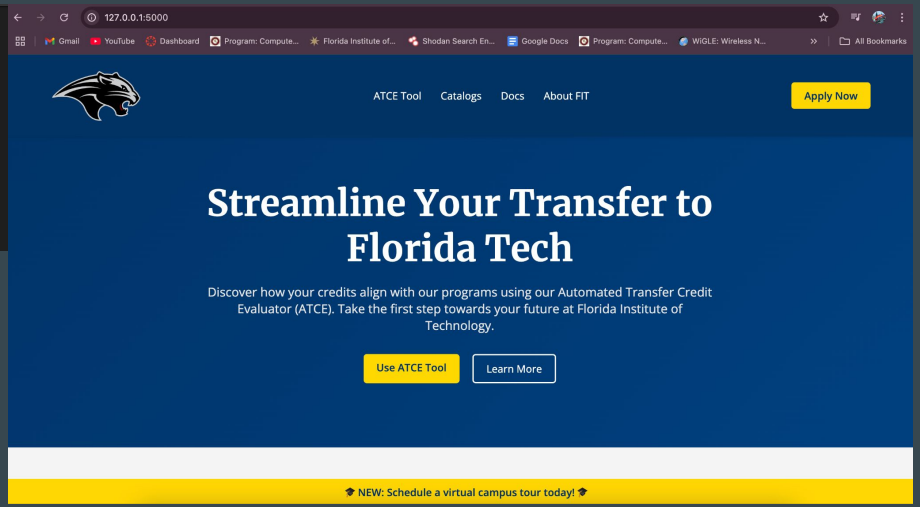
Task 4: Create Basic Flask Application

- Create app.py in main project directory to serve as backend.
- app.py should:
  - Define application's routes
  - Define server static/dynamic content
  - Start the web server
- The @app.route() function defines the routes for each html page.
- The render_template() function loads the respective html file from the /templates directory.

Task 4: Create Basic Flask Application (Cont.)

- Run the application by using the following command: $ python3 app.py

```
(venv) sh-3.2$ ls
README.md       app.py          static          templates       venv
(venv) sh-3.2$ python3 app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 293-917-817
```



Task 5: Set Up Database Connectivity for User Logins with Flask

- Install SQLAlchemy using $ pip install Flask-SQLAlchemy
- Configure the database in Flask by adding the following to “app.py”:

```
from flask_sqlalchemy import SQLAlchemy
…
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
```

- Define the User model and how user data will be stored by adding the following to app.py:

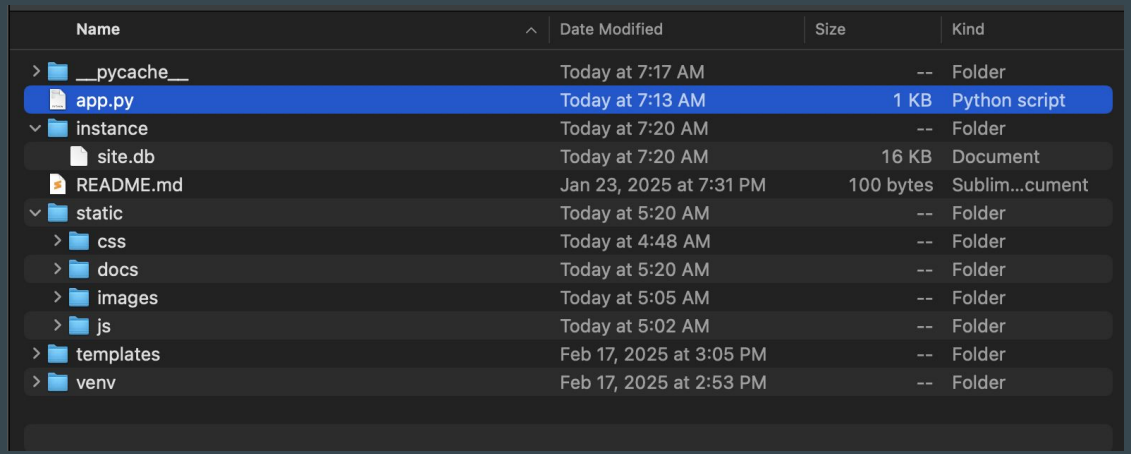
```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)  # Unique user ID
    username = db.Column(db.String(20), unique=True, nullable=False)  # Username (must be unique)
    email = db.Column(db.String(120), unique=True, nullable=False)  # Email (must be unique)
    password = db.Column(db.String(60), nullable=False)  # Hashed password

    def __repr__(self):
        return f"User('{self.username}', '{self.email}')"
```

Task 5: Set Up Database Connectivity for User Logins with Flask

- Create database and tables by running the following in a python shell:

```
$ python
>>> from app import app, db
>>> with app.app_context():
>>> … db.create_all()
```

- We now see a site.db file



Milestone 5 Plans

- Create a login page HTML template
- Design and implement login form with email and password fields
- Add "Login" and "Sign Up" buttons to the main navigation
- Implement client-side form validation
- Set up Flask-Login for session management
- Implement user registration route and logic
- Implement login route and authentication logic
- Add logout functionality
- Connect login form to authentication routes
- Implement error handling and display messages to users
- Create protected routes for logged-in users
- Add user profile page to display account information
- Dockerize the Flask application
- Test the containerized application

Questions?