Team Members:
Tyler Dionne (tdionne2021@my.fit.edu), Kendall Kelly (kelly2021@my.fit.edu)

Project Advisor:
Sneha Sudhakaran, ssudhakaran@fit.edu

Project Title:
FIT Automated Transfer Credit Evaluation

Client:
Sneha Sudhakaran

Website:
https://tylerdionne.github.io/ATCE-FIT/index.html

Milestone 4 Progress Evaluation

1. Progress of Current Milestone:

Task	Completion %	Tyler	Kendall	To Do
Create new directory for Flask Project	100%		100%	N/A
Set up virtual environment and install Flask	100%		100%	N/A
Move html files to a ‘templates’ directory	100%		100%	N/A
Modify html templates to use Jinja2 templating	100%	100%		N/A
Create static directory for css, js, images and docs	100%	100%		N/A
Separate css and javascript from the html files and move	100%		100%	N/A

them to the static directory. Move documents to /static/docs				
Ensure references in html use use url_for to reference the static files properly	100%	100%		N/A
Set up app.py with necessary imports and configurations	100%		100%	N/A
Define routes for each html page	100%		100%	N/A
Implement a render_template() method for each route	100%	100%		N/A
Choose and install a database (SQLAlchemy)	100%	100%		N/A
Create database configuration in app.py	100%	100%		Implement a functional user login system using the database.
Set up a basic model for future logins (User model) and create tables to test	100%	100%		Implement a functional user login system using the model.
Test run site locally and ensure functionality of all pages, images, files	100%		100%	N/A

2. Discussion of Each Completed Task:

Set up Flask project structure

For this step the first task is to clone the repository with the contents of our current static website hosted on github pages:

$ git clone https://github.com/tylerdionne/ATCE-FIT

```
sh-3.2$ git clone https://github.com/tylerdionne/ATCE-FIT
Cloning into 'ATCE-FIT'...
remote: Enumerating objects: 171, done.
remote: Counting objects: 100% (24/24), done.
remote: Compressing objects: 100% (23/23), done.
remote: Total 171 (delta 8), reused 0 (delta 0), pack-reused 147 (from 1)
Receiving objects: 100% (171/171), 5.62 MiB | 3.76 MiB/s, done.
Resolving deltas: 100% (77/77), done.
sh-3.2$ cd ATCE-FIT
sh-3.2$ ls
README.md       docs.html       m1              plan
about.html      index.html      m2              s2-plan
atce.html       logo.svg        m3
sh-3.2$
```

The next step is to set up a virtual environment to isolate our project dependencies from other Python projects and system-wide packages.

$ python3 -m venv venv

$ source venv/bin/activate

```
sh-3.2$ python3 -m venv venv

sh-3.2$
sh-3.2$ source venv/bin/activate
(venv) sh-3.2$
```

Install Flask:

$ pip install Flask

```
(venv) sh-3.2$  pip install Flask
Collecting Flask
  Using cached flask-3.1.0-py3-none-any.whl.metadata (2.7 kB)

Collecting Werkzeug>=3.1 (from Flask)
  Using cached werkzeug-3.1.3-py3-none-any.whl.metadata (3.7 kB)
Collecting Jinja2>=3.1.2 (from Flask)
  Using cached jinja2-3.1.5-py3-none-any.whl.metadata (2.6 kB)
Collecting itsdangerous>=2.2 (from Flask)
  Using cached itsdangerous-2.2.0-py3-none-any.whl.metadata (1.9 kB)
Collecting click>=8.1.3 (from Flask)
  Using cached click-8.1.8-py3-none-any.whl.metadata (2.3 kB)
Collecting blinker>=1.9 (from Flask)
  Using cached blinker-1.9.0-py3-none-any.whl.metadata (1.6 kB)
Collecting MarkupSafe>=2.0 (from Jinja2>=3.1.2->Flask)
  Using cached MarkupSafe-3.0.2-cp312-cp312-macosx_11_0_arm64.whl.metadata (4.0 kB)
Using cached flask-3.1.0-py3-none-any.whl (102 kB)
Using cached blinker-1.9.0-py3-none-any.whl (8.5 kB)
Using cached click-8.1.8-py3-none-any.whl (98 kB)
Using cached itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Using cached jinja2-3.1.5-py3-none-any.whl (134 kB)
Using cached werkzeug-3.1.3-py3-none-any.whl (224 kB)
Using cached MarkupSafe-3.0.2-cp312-cp312-macosx_11_0_arm64.whl (12 kB)
Installing collected packages: MarkupSafe, itsdangerous, click, blinker, Werkzeug, Jinja2, Flask
Successfully installed Flask-3.1.0 Jinja2-3.1.5 MarkupSafe-3.0.2 Werkzeug-3.1.3 blinker-1.9.0 click-8.1.8 itsdangerous-2.2.0

[notice] A new release of pip is available: 24.2 -> 25.0.1
[notice] To update, run: pip install --upgrade pip
(venv) sh-3.2$
(venv) sh-3.2$
```

Set up static files

For this step the first task is to make a static directory within the project directory

$ mkdir static

$ mkdir static/css static/js static/images

```
(venv) sh-3.2$ mkdir static
(venv) sh-3.2$ mkdir static/css static/js static/images
(venv) sh-3.2$ cd static
(venv) sh-3.2$ ls
css     images  js
(venv) sh-3.2$
```

Now we want to move all of our static files into this directory so we can reference them using the Jinja2 templating later on in the html.

Given that the css for each html file are not separated into different files we must do so.

For each file repeat the following:

1. Create a css file for each html file in /static/css

2. Move the css code. Copy everything inside the <style> tag in the html file and paste it into the new css file.

3. Link the css file in the html. Replace the <style> block in the html with a <link> tag inside the <head> section. (ex. <link rel="stylesheet" href="/static/about.css">)

Now we want to move our javascript to the static directory as well. Given that atce.html is the only file that uses javascript we can do this using the following method:

1. Create a file atce.js in /static/js

2. Move everything inside of the <script> tag into this file

3. Reference the javascript file in the html (ex. <script src="/static/js/atce.js"></script>)

The next step is to move our documentation into this folder given that our “docs” page has the content for each milestone. This can be done with a /static/docs folder.

Now lastly we want to move any images we use into the /static/images folder.

For our example we only have one logo.png.

Convert static HTML to Flask templates with Jinja2

For this step the first task is to make a /templates directory within the project directory:
$ mkdir templates

Then move the html files into the templates directory:
$ mv *.html templates/

```
(venv) sh-3.2$ mkdir templates
(venv) sh-3.2$ mv *.html templates/
(venv) sh-3.2$ ls
README.md       m1              m3              s2-plan         venv
logo.svg        m2              plan            templates
(venv) sh-3.2$ ls templates
about.html      atce.html       docs.html       index.html
(venv) sh-3.2$
```

The next step is to prepare the html files with Jinja2 templating.

Example 1: The following line
Home

Would be changed to:
Home

Example 2: The following line:
<img src="logo.svg" alt="Logo" style="width:150px; height:auto;">

Would be changed to:
<img src="{{ url_for('static', filename='logo.svg') }}" alt="Logo" style="width:150px; height:auto;">

Example 3: The following line:
<script src="/static/js/atce.js"></script>

Would be changed to:

<script src="{{ url_for('static', filename='js/atce.js') }}"></script>

Example 4: The following line:
<link rel="stylesheet" href="/static/about.css">

Would be changed to:
<link rel="stylesheet" href="{{ url_for('static', filename='css/about.css') }}">

Example 5: The following line:
<td>View Plan, Presentation</td>

Would be changed to:
<td> View Plan, Presentation </td>

Example 6: The following line
<img src="logo.svg" alt="Logo" class="logo">

Would be changed to:
<img src="{{ url_for('static', filename='images/logo.svg') }}" alt="Logo" class="logo">

Create a basic Flask application

First we must create an “app.py” file in the main project directory.

The “app.py” file in a Flask web app serves as the backend. It handles http requests, routing urls to the appropriate html templates and manages the server side logic.

This file should define the applications routes, serve static/dynamic content and 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.

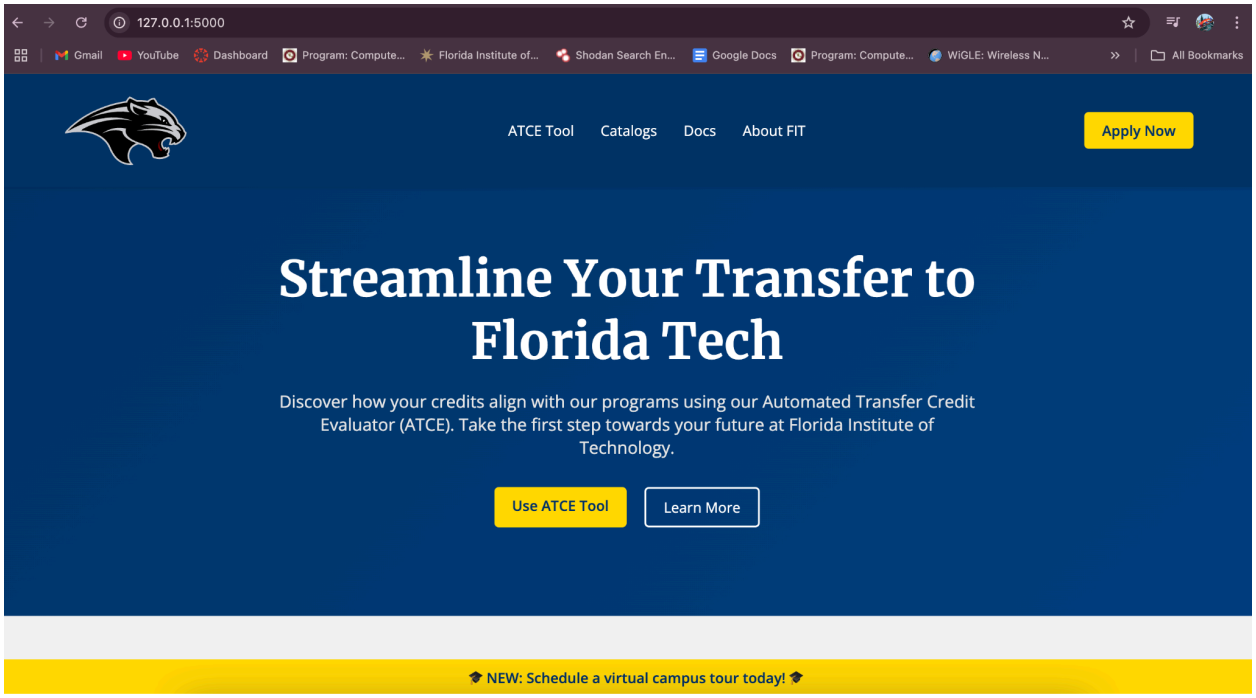
```
from flask import Flask, render_template, url_for

app = Flask(__name__)

@app.route('/')
```

```
def index():
    return render_template('index.html')

@app.route('/docs')
def docs():
    return render_template('docs.html')

@app.route('/atce')
def atce():
    return render_template('atce.html')

@app.route('/about')
def about():
    return render_template('about.html')

if __name__ == '__main__':
    app.run(debug=True)
```

From this point we should now be able to run the application locally from the main project directory using the following command:

$ python3 app.py

Upon navigating to localhost port 5000 we can see the application has successfully launched.



Setting Up Database Connectivity For Future User Logins in Flask

For this task the first step is to install SQLAlchemy with the following command:

$ pip install Flask-SQLAlchemy

```
(venv) sh-3.2$ pip install Flask-SQLAlchemy
Collecting Flask-SQLAlchemy
  Downloading flask_sqlalchemy-3.1.1-py3-none-any.whl.metadata (3.4 kB)
Requirement already satisfied: flask>=2.2.5 in ./venv/lib/python3.12/site-packages (from Flask-SQLAlchemy) (3.1.0)
Collecting sqlalchemy>=2.0.16 (from Flask-SQLAlchemy)
  Downloading SQLAlchemy-2.0.38-cp312-cp312-macosx_11_0_arm64.whl.metadata (9.6 kB)
Requirement already satisfied: Werkzeug>=3.1 in ./venv/lib/python3.12/site-packages (from flask>=2.2.5->Flask-SQLAlchemy) (3.1
.3)
Requirement already satisfied: Jinja2>=3.1.2 in ./venv/lib/python3.12/site-packages (from flask>=2.2.5->Flask-SQLAlchemy) (3.1
.5)
Requirement already satisfied: itsdangerous>=2.2 in ./venv/lib/python3.12/site-packages (from flask>=2.2.5->Flask-SQLAlchemy)
(2.2.0)
Requirement already satisfied: click>=8.1.3 in ./venv/lib/python3.12/site-packages (from flask>=2.2.5->Flask-SQLAlchemy) (8.1.
8)
Requirement already satisfied: blinker>=1.9 in ./venv/lib/python3.12/site-packages (from flask>=2.2.5->Flask-SQLAlchemy) (1.9.
0)
Collecting typing-extensions>=4.6.0 (from sqlalchemy>=2.0.16->Flask-SQLAlchemy)
  Using cached typing_extensions-4.12.2-py3-none-any.whl.metadata (3.0 kB)
Requirement already satisfied: MarkupSafe>=2.0 in ./venv/lib/python3.12/site-packages (from Jinja2>=3.1.2->flask>=2.2.5->Flask
-SQLAlchemy) (3.0.2)
Downloading flask_sqlalchemy-3.1.1-py3-none-any.whl (25 kB)
Downloading SQLAlchemy-2.0.38-cp312-cp312-macosx_11_0_arm64.whl (2.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 17.2 MB/s eta 0:00:00
Using cached typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Installing collected packages: typing-extensions, sqlalchemy, Flask-SQLAlchemy
Successfully installed Flask-SQLAlchemy-3.1.1 sqlalchemy-2.0.38 typing-extensions-4.12.2

[notice] A new release of pip is available: 24.2 -> 25.0.1
[notice] To update, run: pip install --upgrade pip
(venv) sh-3.2$
```

The next step is to 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)
```

The app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db' tells Flask to use a SQLite database stored in the project directory under site.db
The app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False disables unnecessary overhead to improve performance.
The db = SQLAlchemy(app) initializes the SQLAlchemy object with the Flask app instance and the db object will be used to interact with the database.

The next step is to define the User model by adding the following to “app.py” which will define how user data will be stored:

```
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}')"
```

The class User(db.Model) defines a new model class named User. Each model class corresponds to a table in the database and in this case a table named user.

The id = db.Column(db.Integer, primary_key=True) creates a column named id in the user table where the type is an integer and it is the primary key to make sure each user has a unique identifier.

The username = db.Column(db.String(20), unique=True, nullable=False) defines a column for the username and says that it can store strings up to 20 characters long,

must be unique so no two users can have the same username, and the nullable=False parameter makes sure that it can't be empty.

The email and password follow similar rules.

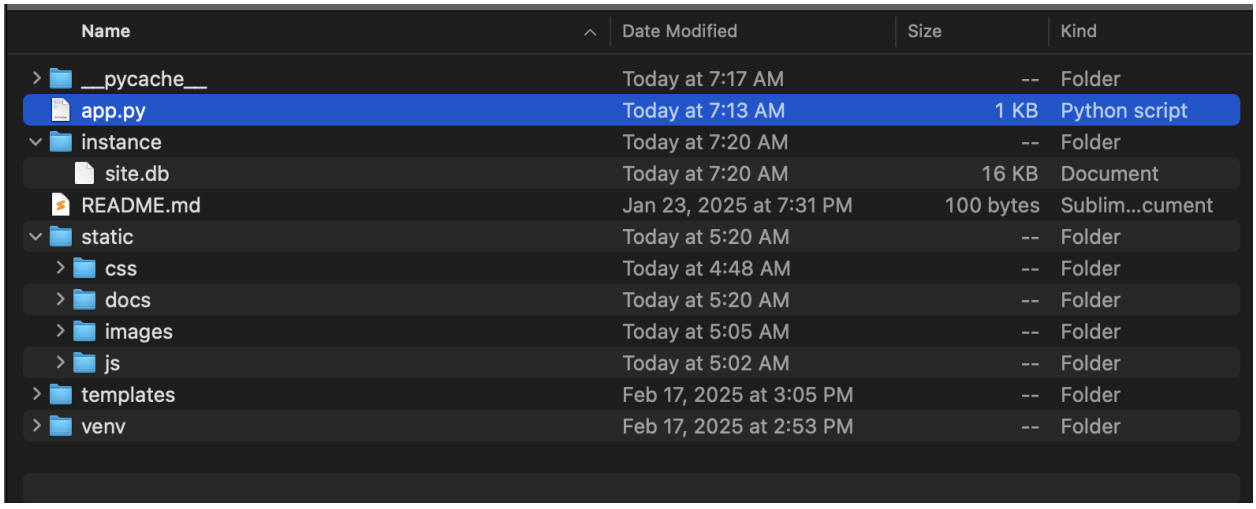
The def __repr__(self) method defines how the User object gets represented as a string so that when you print a User instance or view it this method returns a string that has the username and the email in the format User('username', 'email').

The next step is to create the database and the tables. This can be done by running a few commands in a python shell.

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

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

We now see a site.db file



3. Team Member Contribution of Milestone 4:
Tyler Dionne - Modified html templates to use Jinja2 templating, created static directory for css, js, images and docs, ensured references in html use use url_for to reference the static files properly, chose and installed a database (SQLAlchemy), Create database configuration in app.py, Set up a basic model for future logins (User model) and create tables to test, implemented a render_template() method for each route.

Kendall Kelly - Created new directory for Flask Project, set up virtual environment and install Flask, moved html files to a ‘templates’ directory, separated css and javascript from the html files and move them to the static directory, moved documents to /static/docs, set up app.py with necessary imports and configurations, defined routes for each html page, Test run site locally and ensure functionality of all pages, images, files.

4. Plan for Milestone 5:

Task	Tyler	Kendall
Create a login page HTML template	Will create a login page HTML template	N/A
Design and implement login form with email and password fields	Will design and implement login form with email and password fields	N/A
Add "Login" and "Sign Up" buttons to the main navigation	Will add "Login" and "Sign Up" buttons to the main navigation	N/A
Implement client-side form validation	N/A	Will implement client-side form validation
Set up Flask-Login for session management	N/A	Will set up Flask-Login for session management
Implement user registration route and logic	N/A	Will implement user registration route and logic
Implement login route and authentication logic	N/A	Will implement login route and authentication logic
Add logout functionality	Will add logout functionality	N/A
Connect login form to authentication routes	Will connect login form to authentication routes	N/A

Implement error handling and display messages to users	N/A	Will implement error handling and display messages to users
Create protected routes for logged-in users	Will create protected routes for logged-in users	N/A
Add user profile page to display account information	N/A	Will add user profile page to display account information
Dockerize the Flask application	Will dockerize the Flask application	N/A
Test the containerized application (build and run the Docker container, verify all pages and functionality work correctly, test user registration, login, and logout processes, ensure database connections and data persistence)	Will test the containerized application	Will test the containerized application

5. Date(s) of meeting(s) with Client during the current milestone:
   - Once a week every two weeks

6. Client feedback on the current milestone:
   - See Faculty Advisor Feedback below

7. Date(s) of meeting(s) with Faculty Advisor during the current milestone:
   - Once a week every two weeks

8. Faculty Advisor feedback on each task for the current Milestone:
Faculty Advisor Signature: _____ Date: _____

Evaluation by Faculty Advisor

Faculty Advisor: detach and return this page to Dr. Chan (HC 209) or email the scores to pkc@cs.fit.edu

Score (0-10) for each member: circle a score (or circle two adjacent scores for .25 or write down a real number between 0 and 10)

Tyler Dionne	0	1	2	3	4	5	5.5	6	6.5	7	7.5	8	8.5	9	9.5	10
Kendall Kelly	0	1	2	3	4	5	5.5	6	6.5	7	7.5	8	8.5	9	9.5	10

Faculty Advisor Signature: _____ Date: _____