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 5 Progress Evaluation

1. Progress of Current Milestone:

Task	Completion %	Tyler	Kendall	To Do
User Model Update	100%	0%	100%	N/A
Authentication System Setup	100%	50%	50%	N/A
Login Page Design & Development	100%	100%	0%	N/A
Registration Page Implementation	100%	0%	100%	N/A
Authentication CSS Styling	100%	50%	50%	N/A
Route Protection & Authorization	100%	100%	0%	N/A
Session Management	100%	0%	100%	N/A
User Interface Integration	100%	100%	0%	N/A
Testing	100%	50%	50%	N/A

2. Discussion of Each Completed Task:

User Model Update

Expanded on the already existing User db model to use the UserMixin argument which comes from the Flask-Login library and add necessary attributes and methods for the authentication system. This meant adjusting the User class to properly handle the login sessions, user ID retrieval and authentication state checking.

Added the SQLAlchemy columns to store the username, email and a hashed password to make sure the user's data is protected.

```
class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(20), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(60), nullable=False)

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

This addition of UserMixin makes it so the model can handle is_authenticated, is_active, and is_anonymous properties required by Flask-Login.

Authentication System Setup
Integrated the Flask extensions Flask-Login and Flask-Bcrypt into app.py to create a authentication system.

Involved configuring a secure random secret key and setting up a login manager.

```
app.config['SECRET_KEY'] = os.urandom(24)  # generate random secret key for sessions
bcrypt = Bcrypt(app)
login_manager = LoginManager(app)
login_manager.login_view = 'login'
login_manager.login_message_category = 'info'

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))
```

Login Page Design & Development
Designed a dark themed login page to match the aesthetic of our application and allow the user to fill out a form to login. The page has form validation, error messaging, and a remember me feature.

login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login - Automated Transfer Credit Evaluator</title>
    <link
href="https://fonts.googleapis.com/css2?family=Merriweather:wght@300;400;700&family=Open+Sa
ns:wght@400;600&display=swap" rel="stylesheet">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/index.css') }}">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/auth.css') }}">
</head>
<body>
    <header>
        <img src="{{ url_for('static', filename='images/logo.svg') }}" alt="Logo" class="logo"
style="width:150px; height:auto;">
        <nav>
            <ul>
                <li><a href="{{ url_for('atce') }}">ATCE Tool</a></li>
                <li><a href="#admissions">Catalogs</a></li>
                <li><a href="{{ url_for('docs') }}">Docs</a></li>
                <li><a href="{{ url_for('about') }}">About FIT</a></li>
            </ul>
        </nav>
        <a href="{{ url_for('login') }}" class="cta-button">Apply Now</a>
    </header>
    <main>
        <div class="auth-container">
            <div class="auth-box">
                <h2>Log In</h2>

                {% with messages = get_flashed_messages(with_categories=true) %}
                    {% if messages %}
                        {% for category, message in messages %}
                            <div class="alert alert-{{ category }}">{{ message }}</div>
                        {% endfor %}
                    {% endif %}
                {% endwith %}

                <form method="POST" action="{{ url_for('login') }}">
                    <div class="form-group">
                        <label for="username">Username</label>
                        <input type="text" id="username" name="username" required>
                    </div>
                    <div class="form-group">
                        <label for="password">Password</label>
                        <input type="password" id="password" name="password" required>
                    </div>
                    <div class="form-check">
                        <input type="checkbox" id="remember" name="remember">
```

```
                <label for="remember">Remember Me</label>
            </div>
            <button type="submit" class="auth-button">Log In</button>
        </form>
        <div class="auth-footer">
            <p>Don't have an account? <a href="{{ url_for('register') }}">Register</a></p>
        </div>
    </div>
  </div>
 </main>
</body>
</html>
```

Note the {% %} is Jinja2 templating and that is how we display the messages like “passwords don't match” or “email already registered” these flash messages get defined in the register method in the main app file app.py.

Needed proper login route handling with POST request processing:

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('index'))

    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')
        remember = True if request.form.get('remember') else False

        user = User.query.filter_by(username=username).first()

        # check if user exists and password is correct
        if user and bcrypt.check_password_hash(user.password, password):
            login_user(user, remember=remember)
            next_page = request.args.get('next')
            return redirect(next_page) if next_page else redirect(url_for('index'))
        else:
            flash('Login unsuccessful. Please check username and password.', 'danger')
```

Registration Page Implementation

Made a complete user registration system that validates inputs and securely stores user data:

```
@app.route('/register', methods=['GET', 'POST'])
```

```
def register():
    if current_user.is_authenticated:
        return redirect(url_for('index'))

    if request.method == 'POST':
        username = request.form.get('username')
        email = request.form.get('email')
        password = request.form.get('password')
        confirm_password = request.form.get('confirm_password')

        # check if username or email already exist
        user_exists = User.query.filter_by(username=username).first()
        email_exists = User.query.filter_by(email=email).first()

        if user_exists:
            flash('Username already taken. Please choose a different one.', 'danger')
        elif email_exists:
            flash('Email already registered. Please use a different one.', 'danger')
        elif password != confirm_password:
            flash('Passwords do not match.', 'danger')
        else:
            # hash the password and create new user
            hashed_password = bcrypt.generate_password_hash(password).decode('utf-8')
            new_user = User(username=username, email=email, password=hashed_password)
            db.session.add(new_user)
            db.session.commit()
            flash('Your account has been created! You can now log in.', 'success')
            return redirect(url_for('login'))
```

The system has duplicate checking logic and password hashing for security:

```
hashed_password = bcrypt.generate_password_hash(password).decode('utf-8')
```

Authentication CSS Styling

Made a dedicated CSS file (auth.css) to style the authentication pages:

```
.auth-container {
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: calc(100vh - 120px);
    padding: 40px 20px;
    background-color: #1a1a1a;
}

.auth-box {
```

```
    width: 100%;
    max-width: 450px;
    padding: 40px;
    background-color: #222;
    border-radius: 8px;
    box-shadow: 0 10px 25px rgba(0, 0, 0, 0.5);
}

.auth-box h2 {
    margin-bottom: 30px;
    color: #fff;
    text-align: center;
    font-family: 'Merriweather', serif;
    font-weight: 700;
}

.form-group {
    margin-bottom: 25px;
}

.form-group label {
    display: block;
    margin-bottom: 8px;
    color: #ccc;
    font-family: 'Open Sans', sans-serif;
    font-size: 14px;
}

.form-group input {
    width: 100%;
    padding: 12px 15px;
    background-color: #333;
    border: 1px solid #444;
    border-radius: 4px;
    color: #fff;
    font-size: 16px;
    transition: border-color 0.3s, box-shadow 0.3s;
}

.form-group input:focus {
    border-color: #0077cc;
    box-shadow: 0 0 0 2px rgba(0, 119, 204, 0.2);
    outline: none;
}

.form-check {
    display: flex;
    align-items: center;
    margin-bottom: 25px;
}
```

```
.form-check input {
    margin-right: 10px;
}

.form-check label {
    color: #ccc;
    font-family: 'Open Sans', sans-serif;
    font-size: 14px;
}

.auth-button {
    width: 100%;
    padding: 12px;
    background-color: #0077cc;
    border: none;
    border-radius: 4px;
    color: #fff;
    font-size: 16px;
    font-weight: 600;
    cursor: pointer;
    transition: background-color 0.3s;
}

.auth-button:hover {
    background-color: #0066b3;
}

.auth-footer {
    margin-top: 25px;
    text-align: center;
}

.auth-footer p {
    color: #999;
    font-size: 14px;
}

.auth-footer a {
    color: #0077cc;
    text-decoration: none;
    transition: color 0.3s;
}

.auth-footer a:hover {
    color: #0066b3;
    text-decoration: underline;
}

.alert {
```

```
    padding: 12px 15px;
    margin-bottom: 20px;
    border-radius: 4px;
    font-family: 'Open Sans', sans-serif;
    font-size: 14px;
}

.alert-danger {
    background-color: rgba(220, 53, 69, 0.2);
    border: 1px solid rgba(220, 53, 69, 0.3);
    color: #f17a87;
}

.alert-success {
    background-color: rgba(40, 167, 69, 0.2);
    border: 1px solid rgba(40, 167, 69, 0.3);
    color: #5dd879;
}

.alert-info {
    background-color: rgba(23, 162, 184, 0.2);
    border: 1px solid rgba(23, 162, 184, 0.3);
    color: #5dbecd;
}
```

The styling has responsive form elements, hover effects, and formatted alert messages for errors.

Route Protection & Authorization

Used the @login_required decorator to protect sensitive routes from unauthorized access. This way when you try to click on the ATCE tool without being logged in you cannot use it.

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

This works with with the login manager to properly redirect unauthorized users:

```
login_manager = LoginManager(app)
login_manager.login_view = 'login'  # name of the route to redirect users to login
login_manager.login_message_category = 'info'
```

After the user logins the system automatically redirects it back to the originally requested protected page using Flasks next parameter.

```
next_page = request.args.get('next')
return redirect(next_page) if next_page else redirect(url_for('index'))
```

Session Management
Made complete session handling with secure login, logout, and remember me functionality:

```
login_user(user, remember=remember)
```

For logout used proper session clearing:

```
@app.route('/logout')
def logout():
    logout_user()
    return redirect(url_for('index'))
```

The login page has the remember me box:

```
<div class="form-check">
    <input type="checkbox" id="remember" name="remember">
    <label for="remember">Remember Me</label>
</div>
```

User Interface Integration
Updated main navigation and index page to dynamically display options based on authentication status. This way say a user successfully logs into the site they will no longer see the Login and Register button on the navigation bar they will see a Logout button and vice versa based on the value of user.is_authenticated

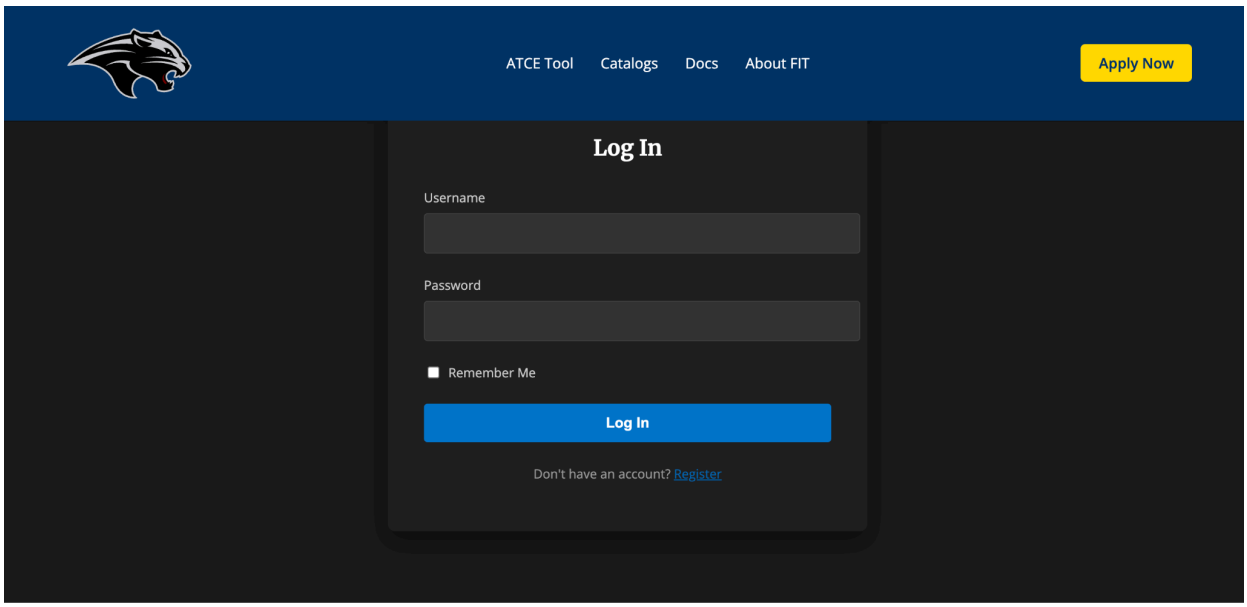
```
{% if current_user.is_authenticated %}
<li><a href="{{ url_for('logout') }}" class="nav-link">Logout</a></li>
{% else %}
<li><a href="{{ url_for('login') }}" class="nav-link">Login</a></li>
<li><a href="{{ url_for('register') }}" class="nav-link">Register</a></li>
{% endif %}
```

Changed the "Get Started" button button to also change based on user's authentication state:

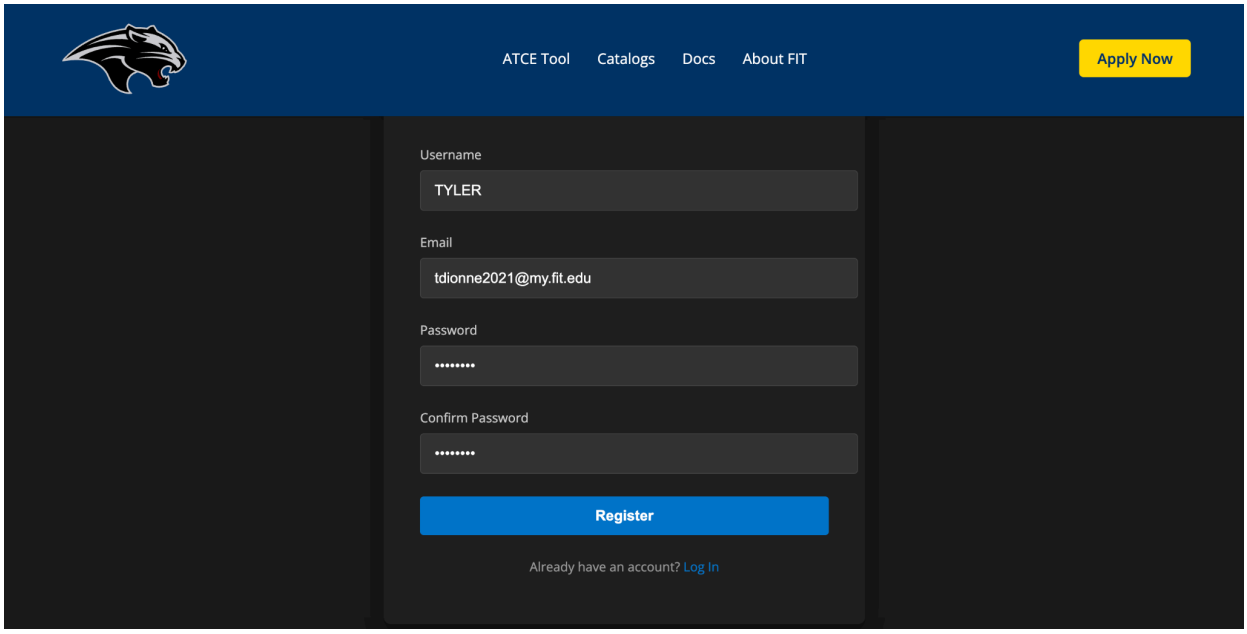
```
<div class="button-group">
   {% if current_user.is_authenticated %}
   <a href="{{ url_for('atce') }}" class="cta-button">Use ATCE Tool</a>
   {% else %}
   <a href="{{ url_for('login') }}" class="cta-button">Get Started</a>
   {% endif %}
   <a href="#" class="secondary-button">Learn More</a>
</div>
```

Demo

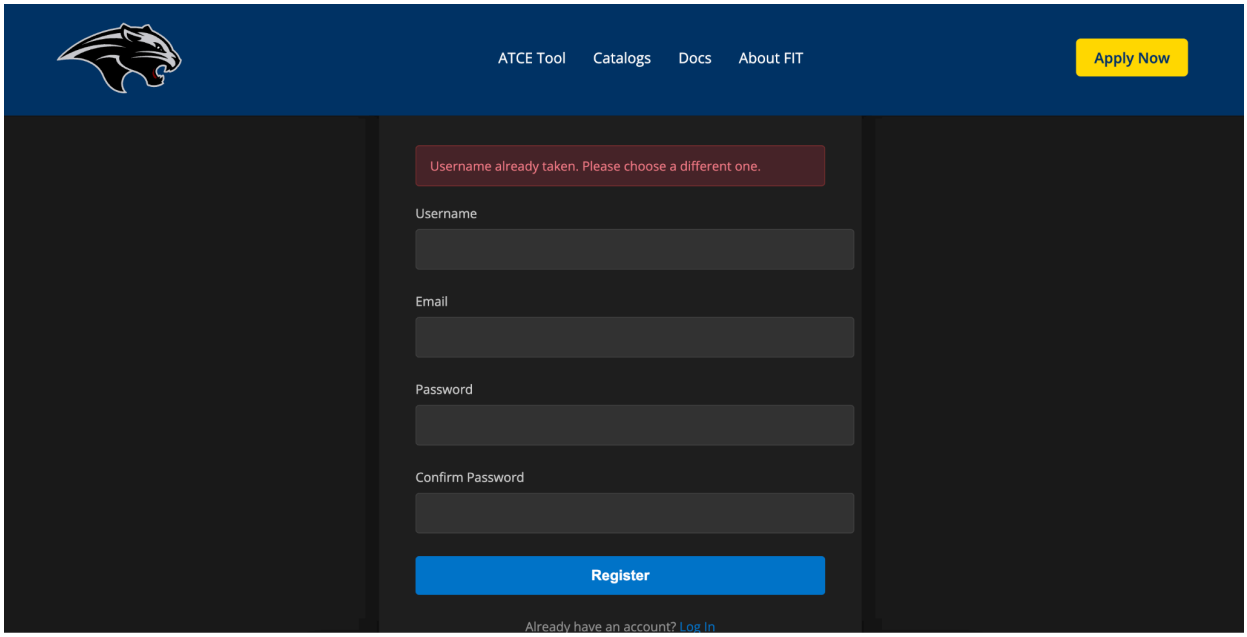
From the homepage go to the Login button on the menu at the top.

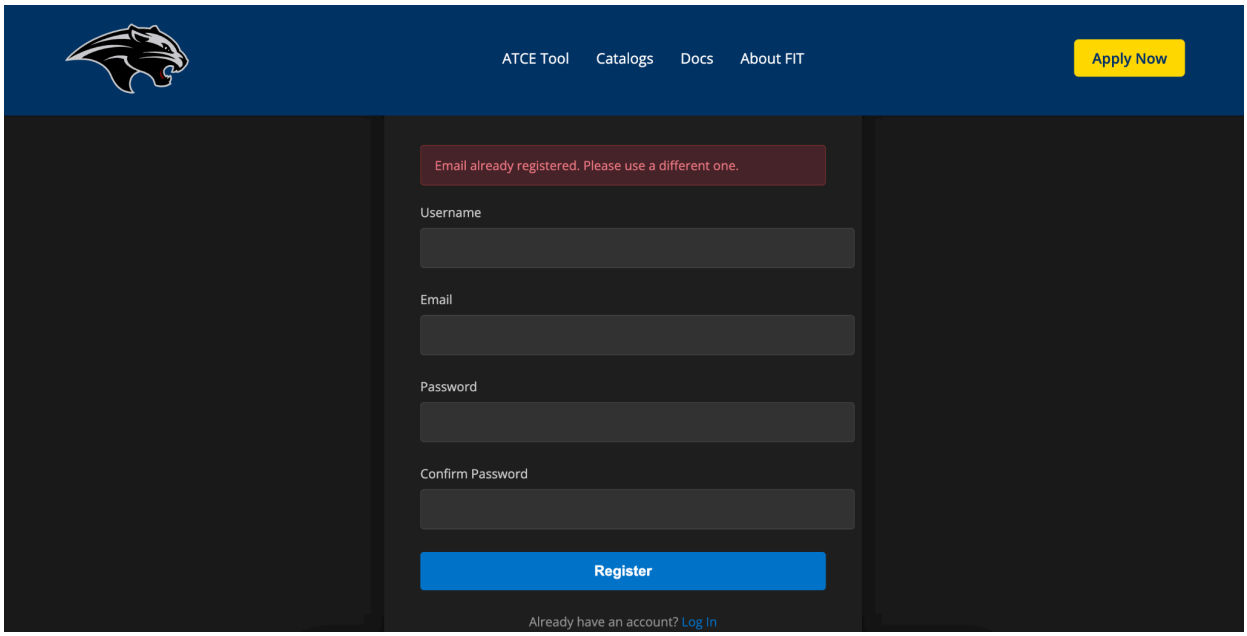


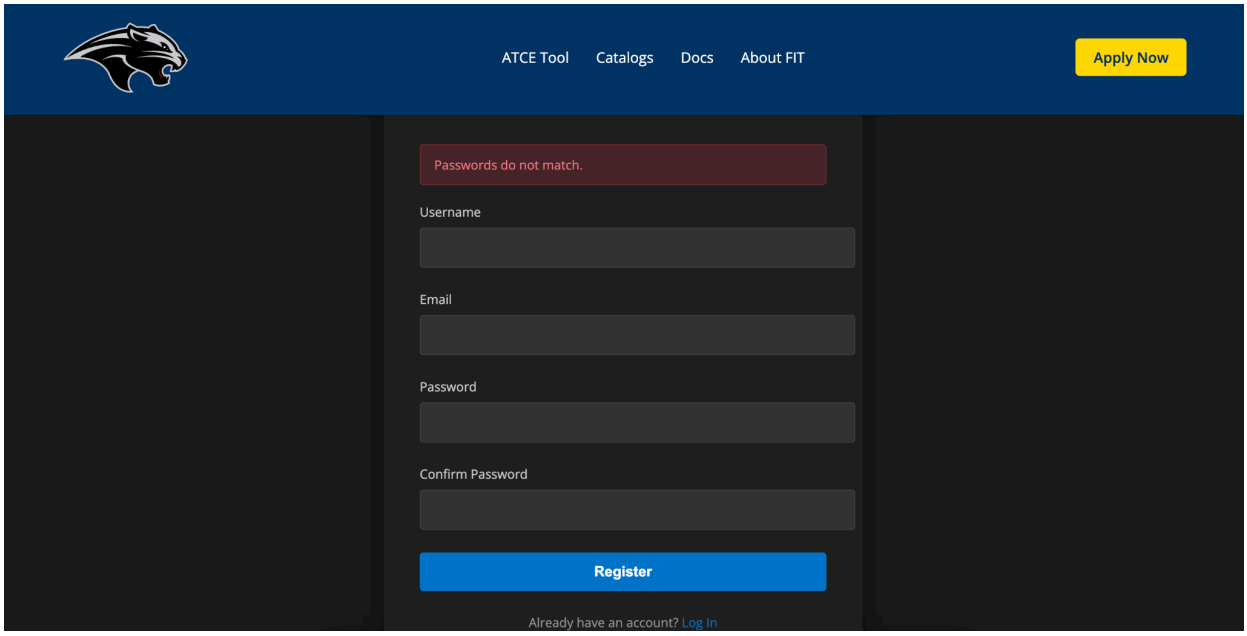
Do not have account so go to register and enter info:



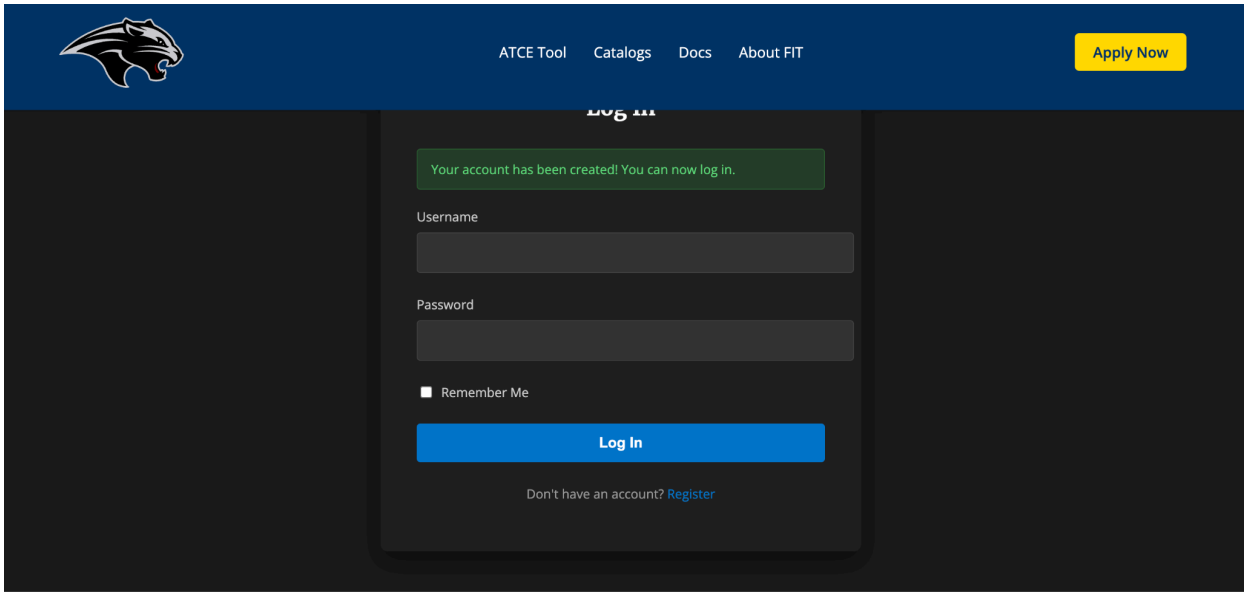
Test warning messages by entering the same username as an already registered user, the same email as an already registered user, and two different passwords.


ATCE Tool
Catalogs
Docs
About FIT
Apply Now
Username already taken. Please choose a different one.
Username
Email
Password
Confirm Password
Register
Already have an account? Log In

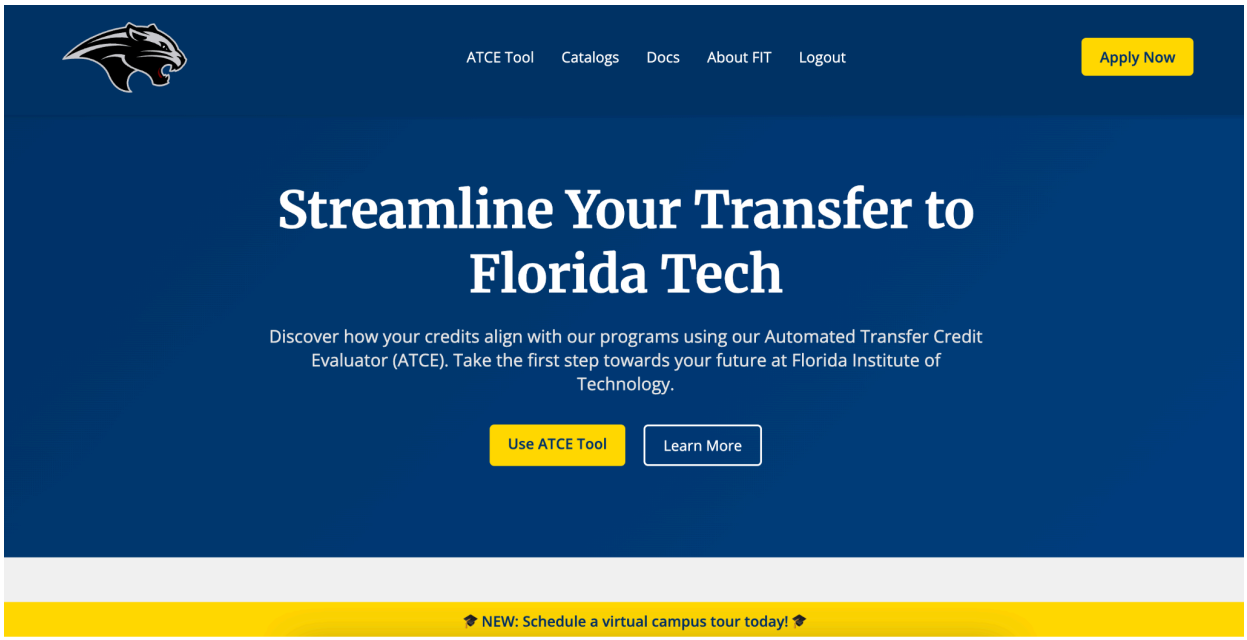

ATCE Tool
Catalogs
Docs
About FIT
Apply Now
Email already registered. Please use a different one.
Username
Email
Password
Confirm Password
Register
Already have an account? Log In



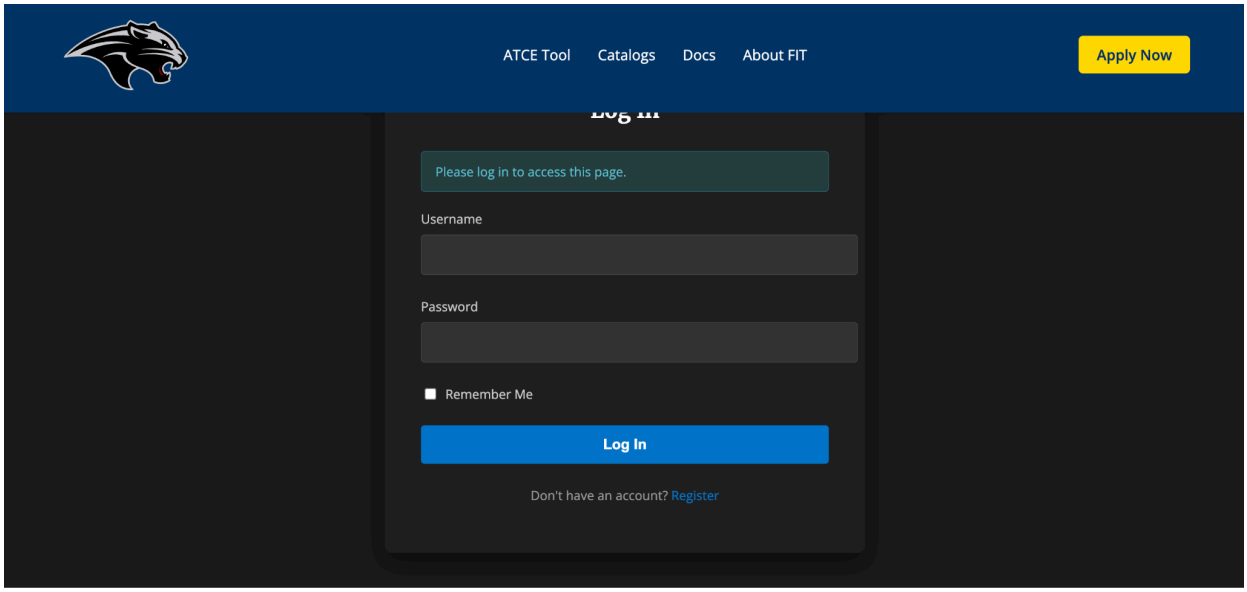
Test success message by successfully registering.



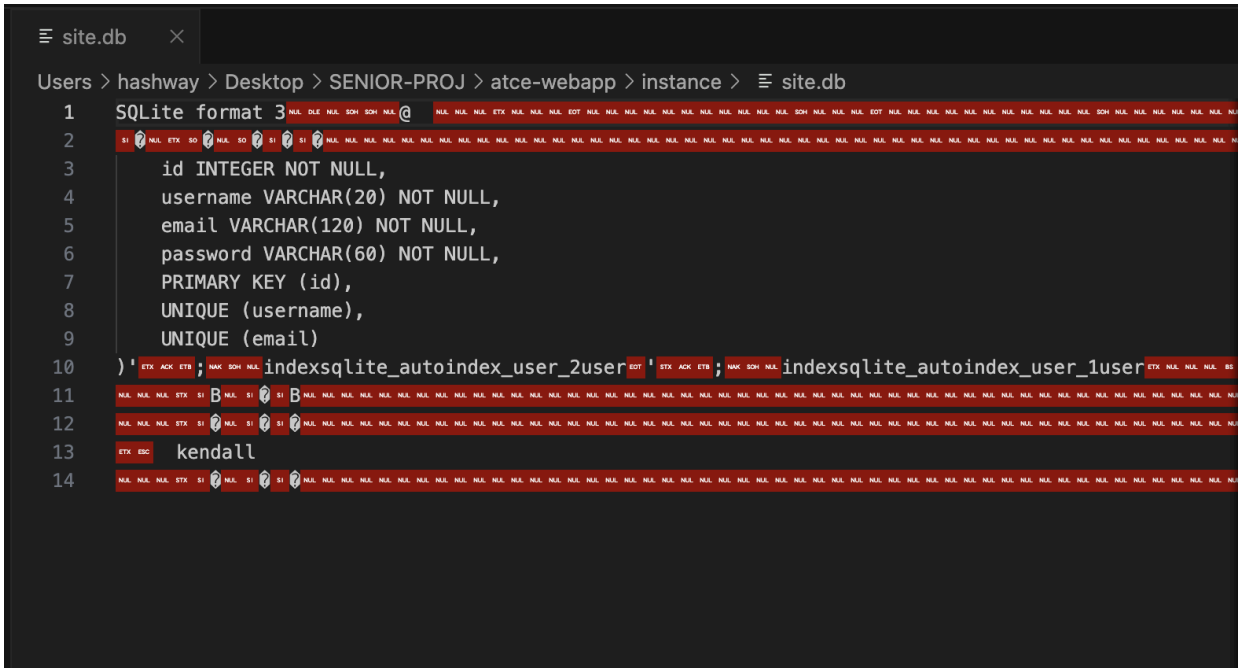
Test login functionality. (Note: When logged in won't see login or register buttons on the menu or the Get Started button)



Test protected routes (/atce without being logged in)



Open instance/site.db in vscode to check to make sure database is working and storing the user data.



Can see that the database is working successfully because we see the format and the one user we created with the username “kendall”.

3. Team Member Contribution of Milestone 5:
Tyler Dionne - Authentication System Setup, Login Page Design & Development, Authentication CSS Styling, Route Protection & Authorization, User Interface Integration,

Kendall Kelly - User Model Update, Authentication System Setup, Registration Page Implementation, Authentication CSS Styling, Session Management, Testing

4. Plan for Milestone 6:

Task	Tyler	Kendall
Conduct basic pentesting on the web application	Will conduct basic pentesting on the web application.	Will conduct basic pentesting on the web application.
Implement basic built in security features within the Flask main app.py	Will implement basic built in security features within the Flask main app.py.	N/A
Review common web application vulnerabilities and proactive measures to be implemented in our web application	Will review common web application vulnerabilities and proactive measures to be implemented in our web application.	Will review common web application vulnerabilities and proactive measures to be implemented in our web application
Review how the file upload feature	N/A	Will review how the file

inside of the atce tool may be exploited and how this may be prevented and implement it		upload feature inside of the atce tool may be exploited and how this may be prevented and implement it.
Review how having a user login feature on a website and managing the users sensitive info can be dangerous and what protective measures can be implemented	Will review how having a user login feature on a website and managing the users sensitive info can be dangerous and what protective measures can be implemented.	N/A
Conduct a final security audit on the web application validating security	Will conduct a final security audit on the web application validating security.	Will conduct a final security audit on the web application validating security.
Dockerize the application	Will dockerize the application.	N/A

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: _____