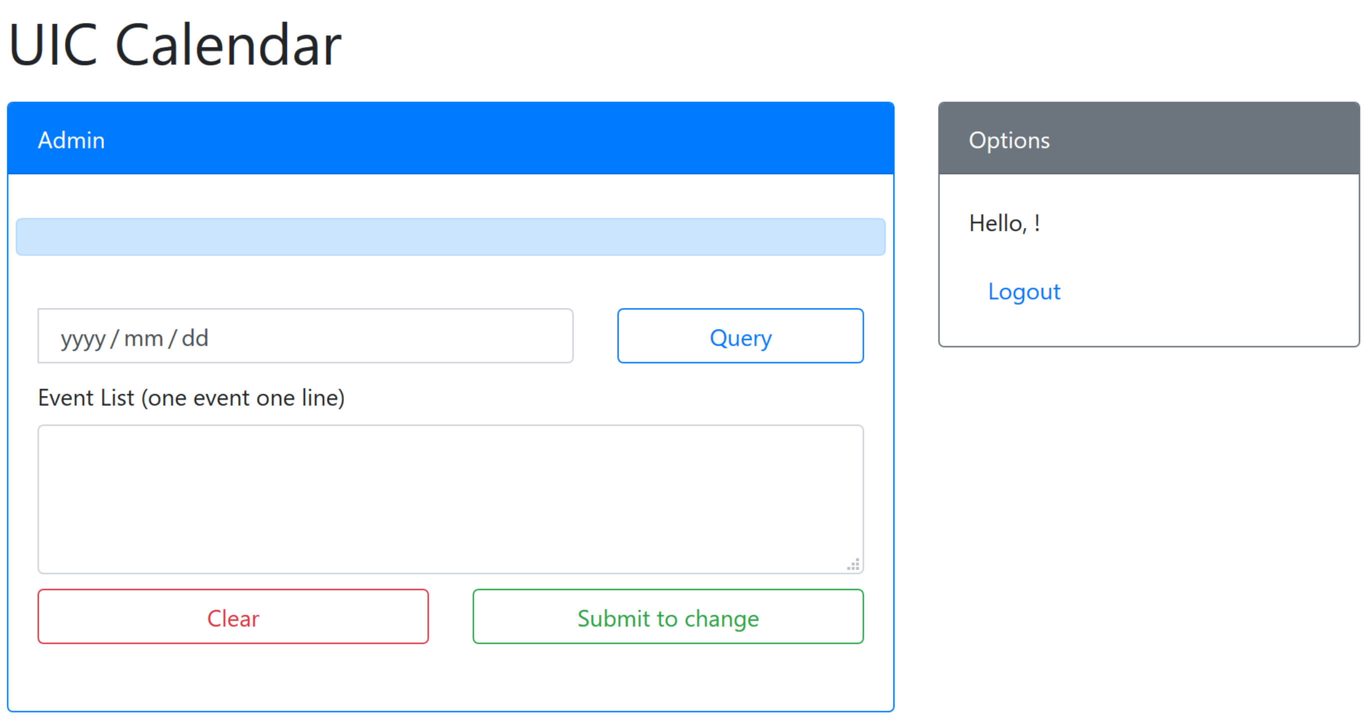
Flask: Login

Task 5.1: Follow this tutorial, and finish the following required login function.

The last topic of this series of Flask tutorials is login. In the `templates` folder, the template `admin.html` is ready.



In this tutorial on login (and logout), you are required to make this page to be restricted by logged in user. And once the user logged in, his/her username will be displayed on the panel on your right-hand side. What's more, if an unauthenticated user tries to access this admin page, the system will jump to `login` with a warning message.

Like the `login` function, let's simplify this user system with only one user `admin`, and its password is `admin` too.

`Flask-Login` Module

Good news! In Flask, you can directly use this third-party module to directly handle user logins. That means you do not necessary to handle sessions.

Install `Flask-Login`

Since we are using a third-party package, we need to install it to your computer first. Simply install it by `pip` or `conda`.

```
pip install flask-login
# or
conda install flask-login
```

Configure `Flask-Login` on Your Website

Initialize `LoginManager`

To configure `Flask-Login` to your website, the first thing you need is to create the object for `LoginManager` class and initialize it. You can do it by the following code,

```
#application.py

from flask-login import LoginManager

login_manager = LoginManager()
login_manager.init_app(app)
```

Set up a secret key

As you may know, `session` is used to handle login & logout in many places, so as Flask. Before we can move on, we need to set s secret key for the session by the following code,

```
#application.py

app.secret_key = b'my-secret-key'
```

The secret key should be a random byte. At here, I just give a meaningful phase to keep it simple.
Please read the following comments when you do actual projects!

> **How to generate good secret keys**
>
> A secret key should be as random as possible. Your operating system has ways to generate pretty random data based on a cryptographic random generator. Use the following command to quickly generate a value for Flask.secret_key (or SECRET_KEY):
>
> ```
> $ python -c 'import os; print(os.urandom(16))'
> b'_5#y2L"F4Q8z\n\xec]/'
> ```

Create a user class

Required by `flask-login`, a user class should be created. You can simply copy-and-paste the following code to your file, before your Flask view code.

```python
# application.py

from flask-login import UserMixin

class CalendarAdmin(UserMixin):
    """User class for flask-login"""

    def __init__(self, id):
        self.id = id
        self.name = 'admin'
        self.password = 'admin'
```

The above user class inherit the base class `UserMixin` provided by `flask-login` module and set up the id, name, and password property for the user.

> You should modify the above code if you want to use this method on your project.

Set up user loader

Required by `flask-login`, we also need to set up a user loader. You can also copy-and-paste the following code to your file, right after the user class implementation.

```python
# application.py

@login_manager.user_loader
def load_user(user_id):
    return CalendarAdmin(user_id)
```

View Methods

Now let's move to the view methods. On 3.Flask-Forms, we discussed how to create the login form. Till now, you should have a `login()` method like below,

```
# application.py

@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm(request.form)
    if request.method == 'POST' and form.validate():
        if form.username.data == 'admin' and form.password.data == 'admin':
            message = 'Login succeed!'
            return render_template('login.html', message=message)
        else:
            message = 'Login failed.'
            return render_template('login.html', message=message)
    else:
        message = 'Test username: admin, password: admin'
        return render_template('login.html', message=message)
```

We need to change the code **if the username and password are correct** from display a message to *redirect* to `/admin` with login information, other remains the same.

According to the document, we should do something like this, read the code and its comments for explanations.

```
# application.py

from flask import redirect
from flask-login import login_user

@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm(request.form)
    if request.method == 'POST' and form.validate():
        if form.username.data == 'admin' and form.password.data == 'admin':
        # Create an object of the authorized user class.
        test_admin_user = CalendarAdmin('admin')
        # Pass the user object ot `flask-login` method `login_user()`.
        login_user(test_admin_user)
        # Exit this function by redirect to the next page.
        return redirect('./admin')
    else:
        ....
```

Then add a decorator for the `admin()` method to prevent unauthorized access.

```python
# application.py

from flask-login import login_required

@app.route('/admin', methods=['GET', 'POST'])
@login_required
def admin():
    return render_template('admin.html')
```

Finally, add a `logout()` method to allow users to log out.

```python
# application.py

from flask import redirect
from flask-login import login_required, logout_user

@app.route('/logout')
@login_required
def logout():
    logout_user()
    # Redirect to homepage
    return redirect('.')
```

Template

Finally, change the `admin.html` template to enable it display the current username.

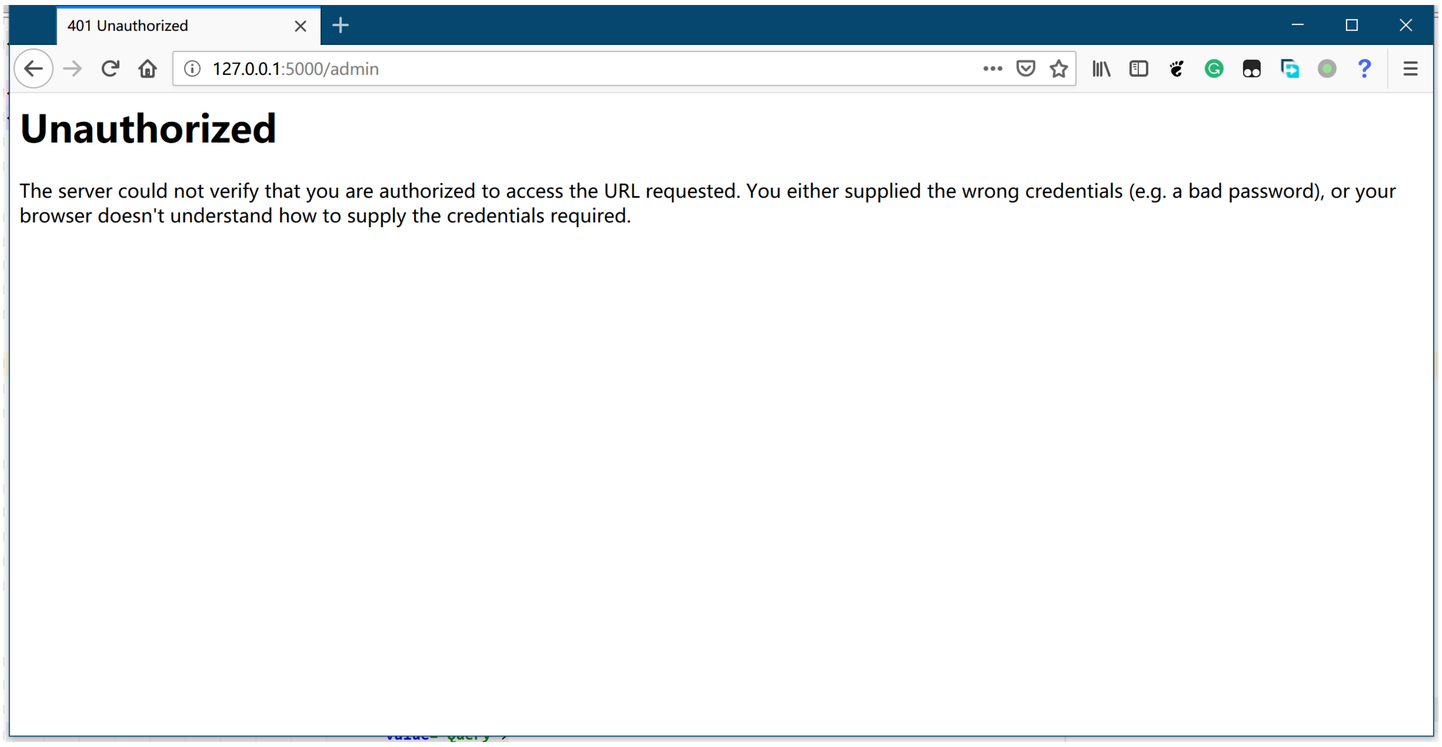
```html
<!-- admin.html -->
<!-- Before -->
<div class="card-body">
    <p>Hello, <b></b>!</p>
    <a href="logout">
        <button type="button" class="btn btn-link">Logout</button>
    </a>
</div>

<!-- After -->
<div class="card-body">
    {% if current_user.is_authenticated %}
    <p>Hello, <b>{{ current_user.name }}</b>!</p>
    {% endif %}
    <a href="logout">
        <button type="button" class="btn btn-link">Logout</button>
    </a>
</div>
```

Examine Your Result

By now, you should finish the mission.

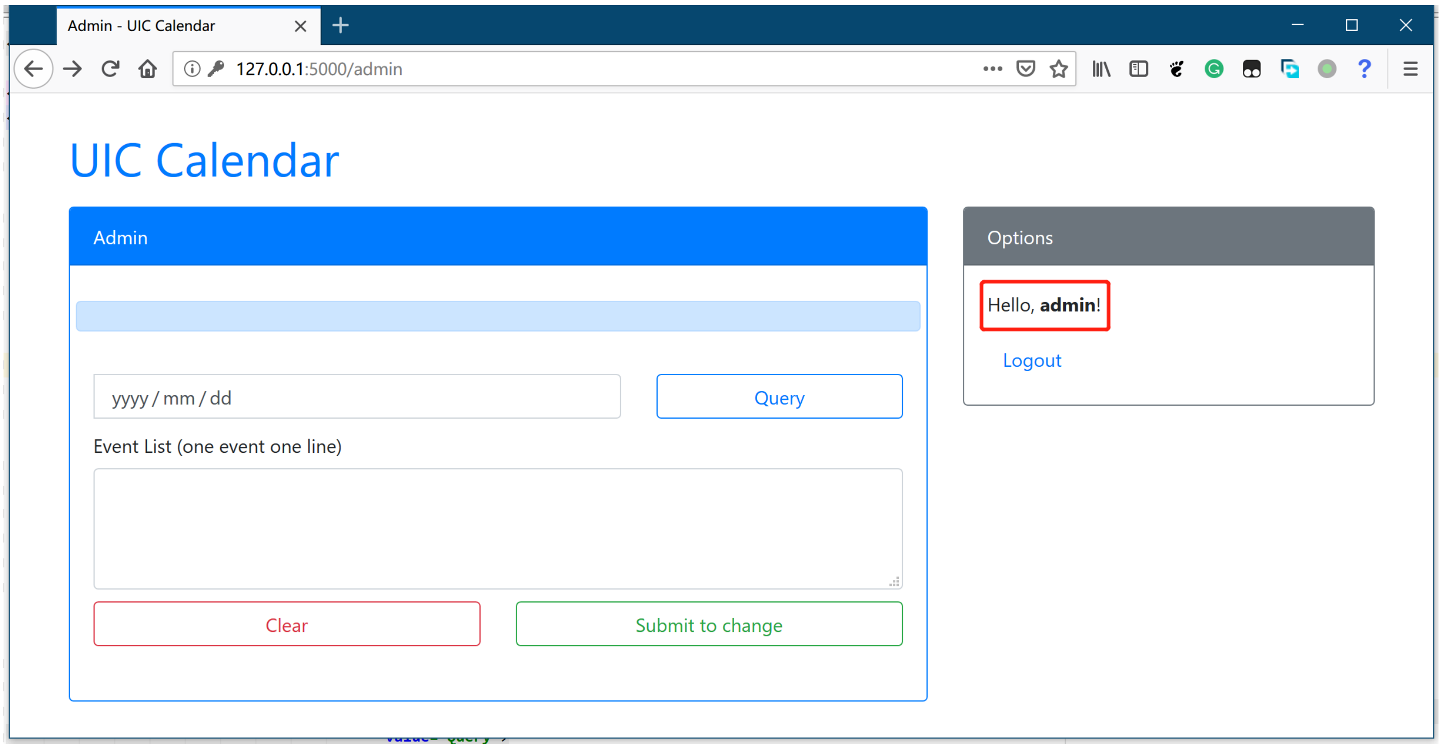
If you try to access the admin page without login, a 401 error will be raised.



Unauthorized

The server could not verify that you are authorized to access the URL requested. You either supplied the wrong credentials (e.g. a bad password), or your browser doesn't understand how to supply the credentials required.

If you log in with the correct username and password, you will see the admin page with your username displayed, and you can click the 'Logout' button to log out.



References

- https://flask-login.readthedocs.io/en/latest/
- http://flask.pocoo.org/docs/1.0/quickstart/#sessions

- https://github.com/shekhargulati/flask-login-example