For better task managment

PROJET : BASIC TICKETING SYSTEM

2024 October 26

EL GHOZZI WAEL

GENARAL IDEA

Ticketing system is a solution to make sharing tasks between team members easier and also tracking status of each task.

Functionalities :

- Logging in to personal account.
- Each user can visualise tickets assigned to him : closed & open.
- User can edit his tickets and save changes.
- User can close his open tickets.
- Website have sessions for logged in users.
- Application contain section where you can hear some jokes.

USED PYTHON/ FLASK TECHNIQUES

- **Rendering Templates** : Use of `**render_template**` for dynamic HTML.
- **User Authentication** : Manage login/logout with sessions and redirects.
- **API Integrations** : Consume external APIs.
- **Form Handling** : Submit forms using `**POST**` requests.
- **Data Management** : Use Python lists.

RENDERING HTML TEMPLATES

In Flask,
render_template loads HTML pages and passes data to them, allowing dynamic content. It enables navigation to different pages and makes links or content adaptable based on passed parameters.

```
# Simple redirection
return render_template('LoginPage.html')

# Redirection to HTML page with parameters
return render_template('HomePage.html', tickets=tickets, user=user)
```

ROUTING WITH @APP.ROUTE

In Flask,
@app.route() defines routes for handling requests.
Routes can include dynamic values, enabling reuse of a single function for multiple contexts. This allows flexibility in displaying different content using the same page.

```python
@app.route('/ticket/<int:ticket_id>')
def ticket_details(ticket_id):
    ticket = next((ticket for ticket in tickets if ticket['id'] == ticket_id), None)
    if ticket:
        return render_template('ticket_details.html', ticket=ticket)
    else:
        return "<h1>Ticket not found</h1>", 404
```

LOGIN/ SESSIONS

LogIn logic using sessions :

Using flask, In order to login, techniques like sessions, redirection, request are used as following

```python
# session configuration
app.secret_key = 'waelKey'
app.permanent_session_lifetime = timedelta(minutes=1)

@app.route('/', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        user = request.form.get('username')
        password = request.form.get('password')

        # Check if the credentials are valid
        if checkCredentials(user, password):
            session.permanent = True
            session['user'] = user
            return redirect(url_for("home"))
        else:
            flash('Password or username is not correct', category='error')
    return render_template('LoginPage.html')

# Function to validate user credentials
def checkCredentials(user, password):
    return USER_CREDENTIALS.get(user) == password
```

CONSUMING EXTERNAL APIS : PYTHON PART

Consuming APIs and fetch data from external APIs is possible using following logic

```python
@app.route('/get_joke')
def get_joke():
    joke_api_url = 'https://official-joke-api.appspot.com/jokes/general/random'
    try:
        response = requests.get(joke_api_url)
        if response.status_code == 200:
            joke = response.json()[0]
            return jsonify(joke) # Return JSON object
        else:
            return jsonify({'error': 'Could not fetch joke'}), 500
    except requests.exceptions.RequestException as e:
        return jsonify({'error': str(e)}), 500
```

CONSUMING EXTERNAL APIS : HTML PART

in order to initiate the api call in python part of code and pass the value fetched we can use the logic below implemented in javascript

```javascript
// JavaScript code to call the function in our python code and pass response to HTML
document.getElementById('joke-button').addEventListener('click', async function () {
        try {
            const response = await fetch('/get_joke');
            const data = await response.json();
            document.getElementById('joke-container').textContent = `${data.setup}... ${data.punchline}`;
        } catch (error) {
            console.error('Error fetching joke:', error);
        }
    });
```

REDIRECT

In Flask, redirect() and url_for()

Are used to navigate between pages with giving destination. you can use them to send the user back to the home page. **url_for()** helps build dynamic URLs, making navigation more flexible.

```
@app.route('/logout')
def logout():
    session.pop('user', None)
    flash('You have been logged out.', category='info')
    return redirect(url_for('login'))
    });
```

INITIATE REDIRECTING LOGIC FROM HTML

To redirect using a link in HTML,

we can use the <a> tag with the target route,like shown in the following example Additionally, you can clear session data before redirecting to the login page for a clean logout experience.

```
<a class="nav-link" href="{{ url_for('logout') }}">Logout</a>
```

FORMS

Forms and submitting input values

In order to send the data to python logic, we use post method and flask route.

```
<form action="/" method="POST"> <!--action is set to the login page route in python code-->
    <div class="form-group">
        <label for="user">Username</label>
        <input type="text" name="username" class="form-control" id="user" placeholder="Enter username" required>
        <small id="emailHelp" class="form-text text-muted">We'll never share your username with anyone else.</small>
        <label for="password">Password</label>
        <input type="password" name="password" class="form-control" id="password" placeholder="Enter password" required>
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
    <!--submitting will send a request to flask login page route-->
</form>
```

RECEIVING FORMS REQUESTS

Forms and submitting input values

In order to recive the message sent from the HTML form we use request with a get method

```
@app.route('/', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        # use request to fetch data from the form input data
        user = request.form.get('username')
        password = request.form.get('password')

        # Check if the credentials are valid
        if checkCredentials(user, password):
            session.permanent = True
            session['user'] = user # Add user to the session and redirect to hame page
            return redirect(url_for("home"))
        else:
            flash('Password or username is not correct', category='error')
    return render_template('LoginPage.html')
```

DATA STORING/ PYTHON LISTS

I use lists to store hardcoded data like ticket information and user credentials. These lists can be manipulated as the application runs, allowing dynamic updates.

DEMO!

THANKS

FOR YOUR ATTENTION