IAB207 – Rapid Web Application Development

2019 S2

Workshop 08

Login and Refine Application

Download folder structure for Assessment 3

Outcomes

- Identify team members (15 min)
- Complete the Travel application (15 min)
- Use Flask-login for authentication (45 min)
- Download folder structure for assessment 3 and get started (30 min)

Workshop Participation

- http://bit.ly/33S1nqD

Exercise 1 (15 minutes)

- Identity your team and team members
- Register on Blackboard
- Create a Trello board – invite your tutor

Exercise 1 (15 minutes)

- Create dynamic content for index.html

Update views.py

- Remove the routes for login and logout.

- Update the index() function
  - Read destinations from database
  - Pass the list of destinations to index.html using render_template
  - Import Destination from models

```
def index():
        destinations = Destination.query.all()
        return render_template('index.html', destinations=destinations)
```

Updating index.html

- The index.html is a static file
  - Change it to populate the card with the destinations in the database
  - Hint: Use the `{% for destination in destinations %}`

    `{% endfor %}`
  - Provide a href to link it to /destinations/<destination.id> route

```
<a href="{{url_for('destination.show', id=destination.id)}}"
class="btn btn-danger">View Details</a>
```

https://git.io/Je329

Exercise 1 (45 minutes)

- Flask-Login for User authentication support

Flask-Login Support

- This functionality will be very similar for the application you build
- Reuse the functionality for your web application

Flask Login Support

- Create login and registration forms

- Create a user.html that renders the login and register forms

- Create an authentication blueprint for login, registration and logout view functions

Login and Register Forms

- Create a login form that takes a user name and password (forms.py)

- Create a register form that takes user name, email id, password, re-confirms password (forms.py)

- https://git.io/fjF3u (forms)

HTML to render the forms

- Create a HTML file that uses bootstrap to render the HTML (user.html)
- Create the file in the templates directory
- https://git.io/fjF30 (user.html)

Authentication Blueprint

- Create auth.py file

- Create a blueprint and create routes for login, register, logout
- Import the LoginForm from forms.py

```
#create a blueprint
bp = Blueprint('auth', __name__)

@bp.route('/login', methods=['GET','POST'])
def login():
       loginForm = LoginForm()
       return render_template('user.html', form=loginForm,
              heading='Login')
```

Register blueprint in main.py

- Register the bluepint in __init__.py

```
from . import auth
app.register_blueprint(auth.bp)
```

Run the application

- Run main.py
  - Access http://127.0.0.1:5000/login
  - http://127.0.0.1:5000/register

Add Flask-Login Support

- https://flask-login.readthedocs.io/en/latest/

- In the models.py update the User class to inherit from a class called UserMixin

- The UserMixin add some additional attributes to the User class and allows Flask-Login to use the class

```
from flask_login import UserMixin


class User(db.Model, UserMixin):
```

Initialize Flask-Login in create_app (__init__.py)

- Import LoginManager from flask_login
- Create a login_manager object
- set the login_view of the login_manager: points to the view function that has the login functionality
- Initialize the login_manager with Flask app
- Add a function that gets User from userid :
  - Flask-Login keeps a cookie with the userid information.
  - We need to create a function that takes as input the userid and returns the User object from the database

Initialize Flask-Login in create_app

- Update the create_app function
- Add these after the initialization of Bootstrap

```
#initialize the login manager
login_manager = LoginManager()
#set the name of the login function that lets user login
# in our case it is auth.login (blueprintname.viewfunction name)
login_manager.login_view='auth.login'
login_manager.init_app(app)

#create a user loader function takes userid and returns User
from .models import User # importing here to avoid circular references
@login_manager.user_loader
def load_user(user_id):
return User.query.get(int(user_id))
```

Register function

- Updated the register function that creates a user when the form is successfully submitted
- Use the generate_password_hash function to store the password hash and not the password

https://git.io/fjFsU

Login function

- Updated the login function that retrieves a user with the username
- It uses the check_password_hash function to validate the user password and the password hash in the database
- If the user has successfully logged, call the login_user function of flask login and set pass the user object

https://git.io/fjFsU

Run the application

- Run main.py

- Test the register functionality, create a user
- Try creating another user with the same name

- Try login and enter incorrect user, password and test the functionality

Use the @login_required decorator

- We now want to ensure that only a user who has logged in is able to create a destination
- Add the import in destinations.py

```
from flask_login import login_required
```

- Add the decorator to enforce login

```
@bp.route('/create', methods = ['GET', 'POST'])
@login_required #decorator between the route and view function
def create():
```

Use the @login_required decorator

- The current comment creation and view functionality does not take the user information (destinations.py)

- Once a user logs in, the user information can be accessed from flask login current_user

```
from flask_login import login_required, current_user
```

- Add the decorator to enforce login to add a comment
- Store the user details while creating the comment

```
comment = Comment(text=form.text.data,
destination=destination_obj, user=current_user)
```

Add logout functionality

- In auth.py , add the logout functionality

```
from flask_login import login_user, login_required,logout_user



@app.route('/logout')
@login_required
def logout():
       logout_user()
       return 'You have been logged out'
```

Update navbar with user details

- The navbar can be updated to indicate if the user had logged in
- Use the current_user.name in the base.html

https://git.io/fjFGC

Run the application

- Run main.py
- Test your application
- Debug the code and identify the code paths

Download Assessment 3 folder structure

1. Download assessment_skeleton.zip from Blackboard
2. Copy your Assessment 2 HTML files in the templates directory
3. Copy your css/images files in the static folder
4. Update the location of images/css in the HTML file to read from static folder

Workshop/Lecture reference for your Assessment

Requirements	Reference
Landing page (#1)	Workshop
Search results page (#2)	Some reference in Workshop Landing page
Item details page (#3)	Workshop
Seller Manage Item page (#4)	-
Updating Sale Information (#5)	-
Item creation form (#5)	Workshop
User registration page (#6)	Workshop
User login page (#7)	Workshop
Seller past sales page (#9)	-
Error handling (#10)	Lecture Week 6 (Template reuse), Workshop

Ready to work on your Assessment!

1. Create a view function that renders the landing HTML page

Images for your Assessment

- Use images that support creative commons license
- Search for images on this website
  - https://ccsearch.creativecommons.org/
  - https://www.pexels.com/creative-commons-images/

- While your application is not for commercial purpose, it will be available on the Internet

Install software

- DB Browser for SQLlite
- Useful to test DB objects

Additional Software

- Team member responsible for the deployment
  - Install Git
  - Install Heroku Command Line Interface
- Details will be provided in the next workshop

THANK YOU!