IAB207 – Rapid Web Application Development

2019 S2

Workshop 05

Introduction to Flask

Agenda

- Introduction
- Python practice
  - Dictionary object
- Exercise 1 (10 mins)
- Exercise 2 (20 mins)
- Exercise 3 (30 mins)
- Exercise 4 (10 mins)

Assessment 2 – Due Sept 15, 11:59 PM

- Download the Assessment on Blackboard
  - Read through the assessment
- Create **static** HTML pages
  - Use Bootstrap
- **Four** pages
  - Landing or Main Page
  - Item details page
  - Item creation page
  - Manage items page
- Support navigation using tags.

Assessment 3 – Week 13 Workshop

- **Team work**
  - Team size (3-4)
  - Peer Review of your contributions
  - Workshop participation
- **Workshop**
  - Week 7 – Identify team members in the same workshop
  - Week 8 - Finalize team members
  - Week 13- Present/Demo your marketplace
- **Monday Workshop Teams**
  - Different time will provided in the week 13

marketplace

Workshop Participation

- http://bit.ly/2TUzbyK

Introduction

- Try out important concepts of Web applications using Flask
- Build the landing page of the web application

Work with Dictionary

- You will use the Dictionary object often

- The following sections in w3schools will introduce you to basic operations. Try the examples (https://www.w3schools.com/python/python_dictionaries.asp)
  - Dictionary
  - Accessing items
  - Changing values
  - Loop through a Dictionary
  - Adding items
  - Removing items

Resources on Blackboard

Week 3	IAB207 2019 S2 – Week 03 – Bootstrap Essentials.pdf	IAB207 2019 S2 – Workshop 02 – HTML5 and CSS3.pdf	IAB207-19S2-Lecture3Demo.zip	Bootstrap Mastering Layout with Bootstrap •Layout Overview •Containers and rows •Use Columns (~12 minutes) Using Navs and Navbar components •Navbar Overiew •Create a navbar •Use branding and text •Add a dropdown to navigation (~30 minutes)
Week 4	IAB207_Python_Programming_Concepts.pdf Covers a few essential and advanced topics. Lecture covers concepts that would be used in the following weeks.	IAB207 2019 S2 – Workshop 03 – Bootstrap Essentials.pdf	IAB207_19S2_Lecture4Code.zip	beginners_python_cheat_sheet_5_classes.pdf beginners_python_cheat_sheet_4_functions.pdf beginners_python_cheat_sheet_2_lists.pdf beginners_python_cheat_sheet_3_dictionaries.pdf beginners_python_cheat_sheet_1_all.pdf Some python cheatsheets you can refer to when writing the code.
Week 5	IAB207 Week5 Introduction_to_WebFrameworks.pdf	IAB207 2019 S2 – Workshop 04 –Python.pdf	IAB207_Lecture5_Flask_code.zip flask_session.py	Flask Flask cheat sheet
Week 6	IAB207 Week6 Templates and Forms1.pdf	IAB207 2019 S2 – Workshop_Debug_Flask.pdf IAB207 2019 S2 – Workshop 05 – Flaskv1.pdf	week6_lecture_forms.zip week6_lecture_templates.zip week6_lecture_variables.zip Download the code and try during the lecture	

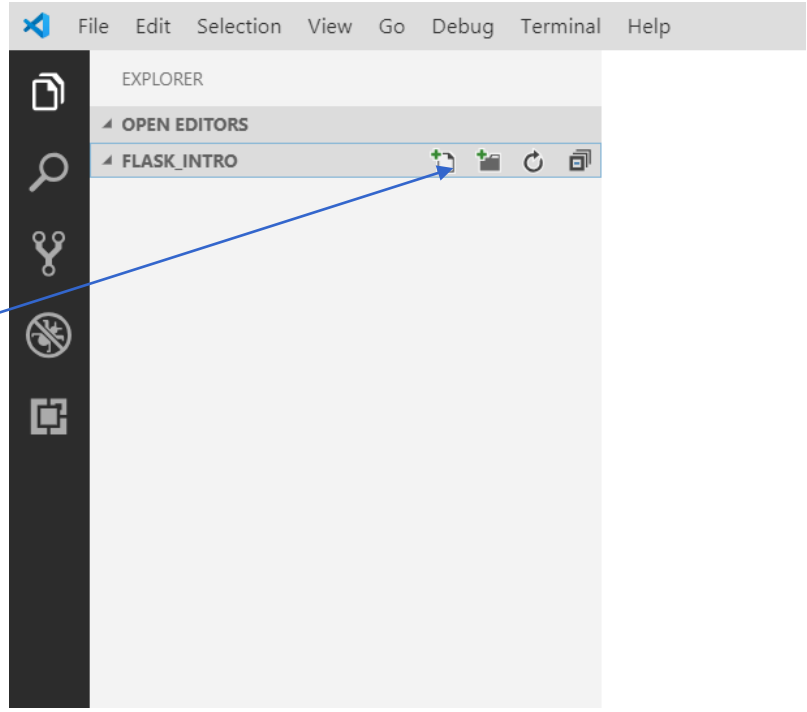
- Download lecture code and run it
- Additional resources like cheat sheets, Lynda links are provided

Exercise 1 (10 minutes)

- Set up Flask
  - If you have not done it in the last workshop
  - Refer to slides from workshop 4

Create a VS Code Project

- Open Visual Studio Code
- File-> Open Folder-> Browse Directory-’New Folder’-> week5
- New File main.py
- New File **travel/__init__.py**
- Creates a folder travel with the file
- Check the Python interpreter chosen by the IDE at the bottom left corner.



Python Virtual Environment

- We will **not** be using virtual environment. Interested students can try and use virtualenv

Installing virtualenv

Note: If you are using Python 3.3 or newer, the `venv` module is the preferred way to create and manage virtual environments. venv is included in the Python standard library and requires no additional installation. If you are using venv, you may skip this section.

virtualenv is used to manage Python packages for different projects. Using virtualenv allows you to avoid installing Python packages globally which could break system tools or other projects. You can install virtualenv using pip.

Exercise 2 (20 minutes)

- Create a sample Flask application
- Understand request and response
- Render your landing page with HTML template

Create a simple Flask application

- Edit __init__.py file
- Import Flask (`from flask import Flask`)
- Create a function with a name – `create_app`
  - You could name it anything, but good to have a name that tells what the function does
- In the function
  - Create a flask application and have the function return the application
    - Hint: `app=Flask(__name__)`

https://git.io/fjFFU

Running the Flask Application

- Running Flask web application from terminal or command line in windows
  - Open command line
  - Change directory to one level **above** travel folder
  - Set variable FLASK_APP=travel
    - SET FLASK_APP=travel (Windows)
    - export FLASK_APP=travel (Mac)
  - flask run

```
 Directory of C:\[redacted]iab207_workshop\week5

09/08/2019  01:52 PM    <DIR>          .
09/08/2019  01:52 PM    <DIR>          ..
07/08/2019  02:03 PM    <DIR>          .vscode
09/08/2019  01:53 PM               106 main.py
09/08/2019  01:51 PM    <DIR>          travel
07/08/2019  02:04 PM    <DIR>          __pycache__
               1 File(s)            106 bytes
               5 Dir(s)  399,365,009,408 bytes free

C:\[redacted]iab207_workshop\week5>set FLASK_APP=travel

C:\[redacted]a\iab207_workshop\week5>C:\Python\Scripts\flask.exe run
 * Serving Flask app "travel"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
travel
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Running the Flask Application in Visual Studio Code Editor

- Add the set of statements to execute the file in main.py

main.py > ...

```
1   from travel import create_app
2
3   if __name__=='__main__':
4       napp=create_app()
5       napp.run(debug=True)
```

- Right click main.py -> Run the ‘Python file in Terminal’

What does this application do

- Access the URL (http://127.0.0.1:5000/) in a web browser and check the server request and response
- What type of request is this?
- What was the response status from the server?
- Stop running the application
  - Check the response from the server

Exercise 3 (35 minutes)

- Use Blueprints
- Render an HTML page
- Understand request types
- Add and delete a session

Add some functionality to the application

- Create another file views.py (in travel folder)
- In this file, you will add some functionality by defining routes
- Create a Blueprint (I have named it mainbp)
  - `from flask import Blueprint`
  - `<blueprintname> = Blueprint('xyz', __name__)`
- Define route
  - `@<blueprintname>.route('/')`
- Define a view function that returns a string '<H1>Hello</H1>'

Update the __init__.py

- Update the __init__.py create_app function
  - Import views module
  - Register blueprint

```
import views
app.register_blueprint(views.<blueprintname>)
```

- https://git.io/fjFFT
- https://git.io/fjFFk

Run the Flask Application

- Access the URL (http://127.0.0.1:5000/) in a web browser and check the server request and response
- What type of request is this?
- What was the response status from the server?

Connect the dots

- Multiple functions required for a solution
- Most changes require
  - HTML update
  - View function update
  - Database (later)
- Go through the code and understand the flow

Understand request methods

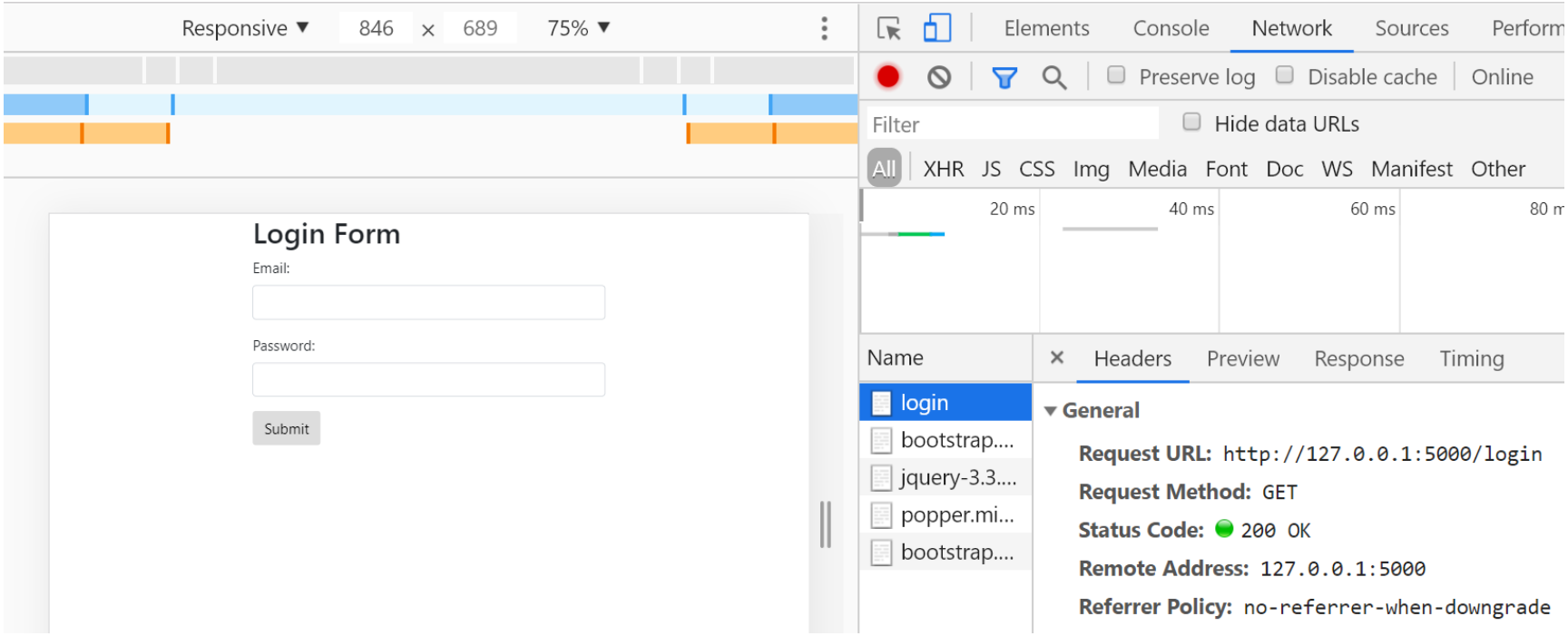
- Create a login.html (folder travel/templates/login.html)

- This is a form
  - https://git.io/fjFFI
  - Update the ‘method’ and ‘action’ in the form ‘GET’ and ‘/login’
- Create a ‘/login’ route in views.py
  - This function renders the HTML form when called
  - Import render_template from flask (from flask import render_template)

```
@main.route('/login') #route name with
def login():                # view function
        return render_template('login.html')
```

Run the Flask Application

- Access URL (http://127.0.0.1:5000/login)
  - The /login route renders the login.html template
  - Inspect the request in the browser



Run the Flask Application

- Enter the details in the form
- Click submit
- Check the URL in the browser (inspect the request in the browser)
- Discuss your understanding of the execution of the code with your team member

Print the Email ID and password

- In your view function login() – print the email id and password.
- Hint: access the request object
  - `from flask import request`
  - Use `request.args.get`

Change the method type in the Form

- In the login.html, change attribute of the form method='POST'
- `Run __init__.py`
- Access URL (http://127.0.0.1:5000/login)
- What does the page show? Discuss with your team member

Solution

By default view function support only ‘GET’ Request

Add ‘POST’ method to the route decorator

```
@mainbp.route('/login', methods=['GET','POST'])
def login():
        print(request.args.get('email'))
        print(request.args.get('pwd'))
        return render_template('login.html')
```

Set a session variable

- Set the secret key in create_app function of __init__.py

```
app.secret_key='anythingyoulike'
```

- In the login() function add (import session to avoid syntax error)
  - Request.args.get only works when the method is ‘GET’
  - To support accessing values of both ‘GET’ and ‘POST’ use

```
session['email']=request.values.get('email')
```

- Update index() function

```
def index():
        if 'email' in session:
                str='<h1>hello world' + session['email'] + '</h1>'
        else:
                str='<h1>hello world</h1>'
        return str
```

Run the Flask Application

- Run the flask application
- Access the URL http://127.0.0.1:5000/

- Access the URL http://127.0.0.1:5000/login
- Enter the emailid and password

- Access the URL http://127.0.0.1:5000/
- The session is now stored

Delete the session

```
@mainbp.route('/logout')
def logout():
        if 'email' in session:
                session.pop('email', None)
        return 'Session has been cleared'
```

1. Run the flask application

2. Access the URL http://127.0.0.1:5000/
    1. (Does the session exist – if not, login and access again)

3. Access the URL http://127.0.0.1:5000/logout

4. Access the URL http://127.0.0.1:5000/

Exercise 4 (10 minutes)

- Render your html landing page

- Create an index.html in the templates folder
  - Use the html you created in workshop 3
  - Copy the html code from (https://git.io/fjAeI)

- Replace the existing lines of code in the index() function to render the template

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

Render the landing page of your Travel Web application

- Run your flask application again

Summary

- Created your first web application using Flask
- Looked at different types of requests
- Used your index.html page from the previous workshop