IAB207 – Rapid Web Application Development

2019 S2

Workshop 06

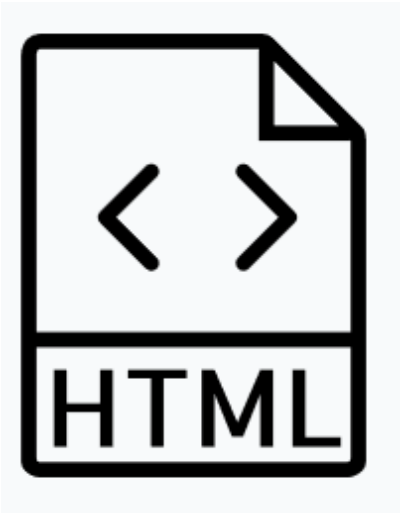
Templates and Forms

Agenda

- Introduction
- Outcomes
- Last Week Recap
- Exercise 1: Reusing common parts of HTML
- Exercise 2: Parameter passing in Templates
- Exercise 3: Working with Flask Forms

Assessment 2 – Due Sept 15, 11:59 PM

- 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.
- **Refer to Week 3 Lecture code and Workshop 3 (in week 4)**
  - **Look for examples on Bootstrap documentation page**

Assessment 3 – Week 13 Workshop

- **Will discuss the assessment in the workshop next week. Please attend next 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 end of next week
  - Week 13- Present/Demo your marketplace
- Students in the Monday batch will get less time
  - Some time during the week will be assigned

Recap

- Worked with a simple Flask application
- Worked with GET/POST methods
- Rendered an html file through the Flask application

Workshop Participation

- http://bit.ly/2PaewIf

Generic ‘Tip’

- When you think the code should work and it doesn’t
  - Close browser and restart – clears some cache
  - Stop and Start the Flask server (Ctrl+C and start)
  - If nothing works – restart visual studio code ☺
- If it still doesn’t work, then there is high probability of a bug in the code

Introduction

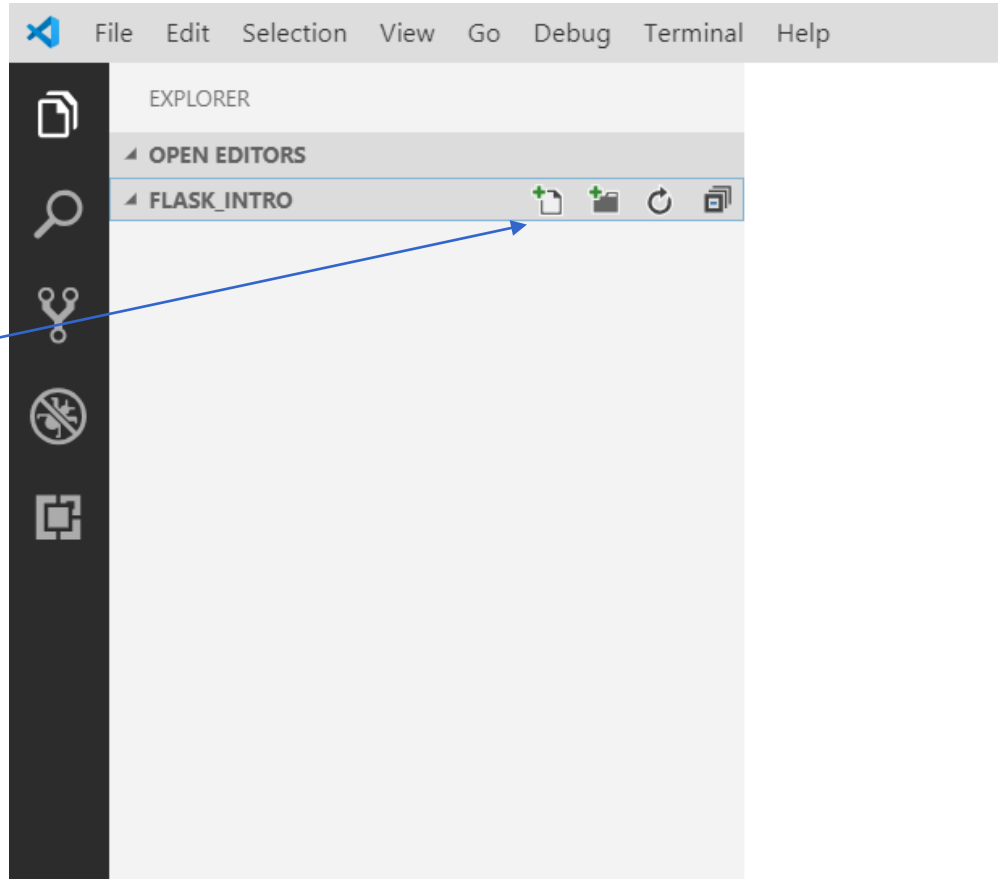
- Add some dynamic content to our web application
- Work with creating forms in the application

Exercise 1 (15 minutes)

- Creating a reusable template

Create a VS Code Project

- Open Visual Studio Code
- File-> Open Folder-> Browse Directory-'New Folder'-> week6
- Copy the folder travel and main.py from week5
- Check the Python interpreter chosen by the IDE at the bottom left corner.



Create a common html file (base.html)

- In the templates folder, create a file base.html
  - This file will contain all the reusable HTML of your application – the portions of header, footer.

- From your index.html, copy the initial sections of the html: head, title, navbar
  - After you copy the section – delete the code in index.html

Create a common html file (base.html)

- Create a place holder for additional header content that can be added in other pages by adding the following lines

  {% block header %}

  {% endblock %}

  </header>

- Create a place holder for additional content after the </header> tag

  {% block content %}

  {% endblock %}

Create a common html file (base.html)

- Add the footer content
- Add the </body> </html> tag to end of the file

Refer to code for help: https://git.io/fjd9g

Update HTML to reuse base.html

- Update index.html
  - It would now have some fragments of HTML

- To reuse base.html add this at the beginning of index.html

```
{% extends 'base.html' %}
```

- Place the jumbotron code in the header block placeholder

```
{% block header %}
<<<<All the jumbotron content>>>>
<% endblock %}
```

Update HTML to reuse base.html

- Place the remaining content to the content block placeholder

  {% block content %}

  <<<<All the remaining html content>>>>

  <% endblock %}

https://git.io/fjd92

Run Flask Application (main.py)

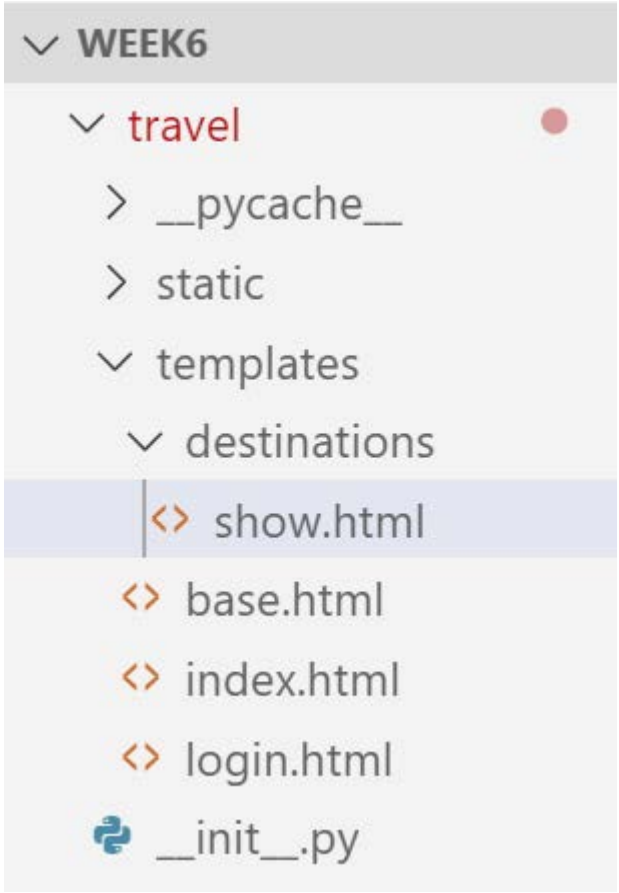
- Access the URL (http://127.0.0.1:5000/) in a web browser and check the server request and response
- The main page should appear as it had in workshop 5

Exercise 1 (contd..)

- Create another page that reuses base.html

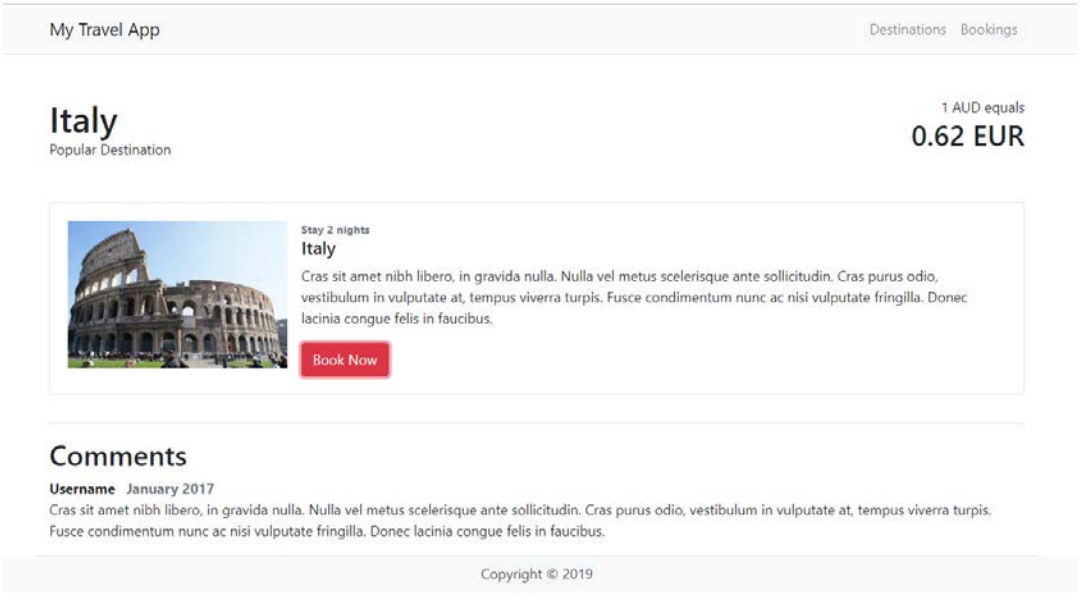
Create a page providing details of a destination

- Create a folder destinations in the templates folder
- Create show.html page
- Reuse ‘base.html’ :
  - {% extends 'base.html' %}



Destinations Page

- Format the page and add html content within
  - {% block content %}
  - {% endblock %}



Code for a page above is: https://git.io/fjdQP

Create a Blueprint

1. We will create another Blueprint that handles all destination related code

2. Create a destinations.py file under the travel folder

3. Create a Blueprint to manage all functions related to the 'destinations'
    1. Notice the url_prefix => all routes for this blueprint will have '/destination' url prefixed

```
#create a blueprint
bp = Blueprint('destination', __name__, url_prefix='/destinations')
```

Create a Blueprint

1. Create a view function to show the destinations page

2. Add missing imports that are highlighted by VS Code as errors (Blueprint, render_template)

```
#create a page that will show the details fo the destination
@bp.route('/<id>')
def show(id):
    return render_template('destinations/show.html')
```

https://git.io/fjpqn

Register the Blueprint

- Update create_app() function in the __init__.py file to register the blueprint

```
def create_app():

    app=Flask(__name__)  # this is the name of the module/packa
    app.debug=True
    app.secret_key='utroutoru'

    #importing views module here to avoid circular references
    # a commonly used practice.
    from . import views
    app.register_blueprint(views.mainbp)

    from . import destinations
    app.register_blueprint(destinations.bp)

    return app
```

Run the Flask Application

- Run the application (main.py)
  - Run Python file in Terminal

- Access http://127.0.0.1:5000/destinations/1

Exercise 2

- Passing variable to the template

Pass variables to template html

- The HTML files have static content
- We will create a destination and comment object and pass it to the show.html template
  - The view function will need to pass the destination object
  - The HTML will need to show this information

Pass variables to template html

- Request is made to the route /destinations/1.
  - The view function show() will get the destination object
  - render_template will use the destination object and generate HTML

Create Destination Class

- Create a models.py file in the travel folder

- Create a class Destination with a __init__ method that instantiates the following attributes
  - name
  - description
  - image
  - currency
  - comments (list)

Create Comment class

- In the models.py, create class Comment
- The __init__ method that instantiates the following attributes
  - user
  - text
  - created_at

https://git.io/fjd7R

Create data in the application

- Create a function that returns destination object in destinations.py
  - Create your own destination object
  - You will need to import Comment and Destination from models

```
from .models import Destination,Comment
```

  - The sample code can be uncommented from https://git.io/fjpqn

Update the function that renders HTML

- Update the show function to pass a variable named `destination` to the template
  - render_template is a function that accepts variable number of named parameters

```
#create a page that will show the details fo the destination
@bp.route('/<id>')
def show(id):
    destination = get_destination()
    return render_template('destinations/show.html', destination=destination)
```


name of the variable accessible in the template

Update HTML to use variables

- The `destination` object passed to render_template can be accessed in show.html

- Replace the static content with `{{ variable }}` in your html

  E.g. replace ‘Italy’ with {{destination.name}}

  Other static content with {{destination.currency}}, {{destination.description}}

Refer to: https://git.io/fjd7M

Update HTML to use variables

- Since destination.comments is a list, add a `for` loop in your html to read each comment

```
{% for comment in destination.comments %}
{% endfor %}
```

- Replace comment with variables

```
{{comment.user}}, {{comment.text}}
```

Refer to: https://git.io/fjd7M

Run the Flask Application

- Run the application (main.py)
  - Run Python file in Terminal

- Access http://127.0.0.1:5000/destinations/1

Exercise 3

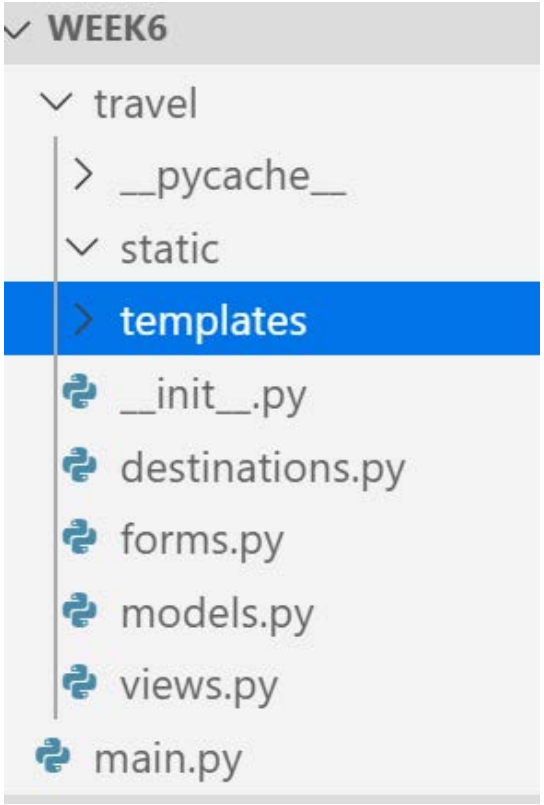
- Use Flask WTForms, Flask-Bootstrap

- Start command line on windows / terminal on mac
- Windows: cd <Python directory>/Scripts
  - `pip3 install flask-wtf`
  - `pip3 install flask-bootstrap`

Create a Form (comment form)

- Create a forms.py in the travel folder

- Add the form to capture user comments
  - The form contains a field, button, and validator to ensure input is entered



Create a Form (comment form)

```
from flask_wtf import FlaskForm
from wtforms.fields import TextAreaField,SubmitField

# this is a form class that inherits from FlaskForm
class CommentForm(FlaskForm):
#the form is simple with a text field and a submit button
        text = TextAreaField('Comment')
        submit = SubmitField('Create')
```

Render the form in the HTML

- Flask bootstrap provides an easy mechanism of rendering forms
  - https://pythonhosted.org/Flask-Bootstrap/forms.html

- We will change the action URL. By default the action is '.' - the same URL that was used to render the HTML form

quick_form(*form, action=".", method="post", extra_classes=None, role="form", form_type="basic", horizontal_columns=('lg', 2, 10), enctype=None, button_map={}, id=""*)

Outputs Bootstrap-markup for a complete Flask-WTF form.

Parameters:
- ***form*** – *The form to output.*
- ***method*** – `<form>` *method attribute.*

Render form using Flask-Bootstrap

1. Import bootstrap in templates/destinations/ show.html.

   {% import "bootstrap/wtf.html" as wtf %}

2. Add the following line of code in the HTML to show the form
   1. **{{ wtf.quick_form(form, "/destinations/{0}/comment".format(1)) }}**
   2. Add it just before the {%for %} loop
   3. The string format function results in the URL '/destinations/1/comment'

Render form using Flask-Bootstrap

1. Initialize Bootstrap in create_app() function of __init__.py

```
bootstrap = Bootstrap(app)
```

If you get an error – remember to import Bootstrap

```
from flask_bootstrap import Bootstrap
```

```
def create_app():
        app=Flask(__name__)
        app.debug=True
        app.secret_key='somerandomvalue'
        bootstrap = Bootstrap(app)
```

Send the flask form to be rendered

- View function should pass the form
- Import CommentForm (Hint: similar to Destination)

```
#create a page that will show the details of the destination
@bp.route('/<id>')
def show(id):
        destination = get_destination()
         # create the comment form and
        cform = CommentForm()
        return render_template('destinations/show.html',
        destination=destination, form=cform)
        #in the html this is access as a variable named form
```

https://git.io/fjpsD Code for reference

Run the application

- Access the URL http://127.0.0.1:5000/destinations/1
- Comment form is visible, inspect the HTML
- Inspect the form in Chrome and look for the CSRF hidden field

Run the application

- Entering a text and submitting gives a 404 error as there is no route defined http://127.0.0.1:5000/destinations/1/comment

Implement the route to submit a comment

- Create a route and the function that handles submission of the comment
  - Import the necessary functions (redirect, url_for, request)

```
@bp.route('/<id>/comment', methods = ['GET', 'POST'])
def comment(id):
        form = CommentForm() # create the CommentForm from whats posted

        if form.validate_on_submit():
                print("Comment posted by the user:", form.text.data)

        # using redirect to redirect it to /destinations/1 notice id=1
        return redirect(url_for('destination.show', id=1))
```

https://git.io/fjpsD Code for reference

Run the application

- Run the application
- Access the URL http://127.0.0.1:5000/destinations/1
- Post a comment.
  - The comment is not stored.
- We will store the data once our database is ready

Thank you ☺