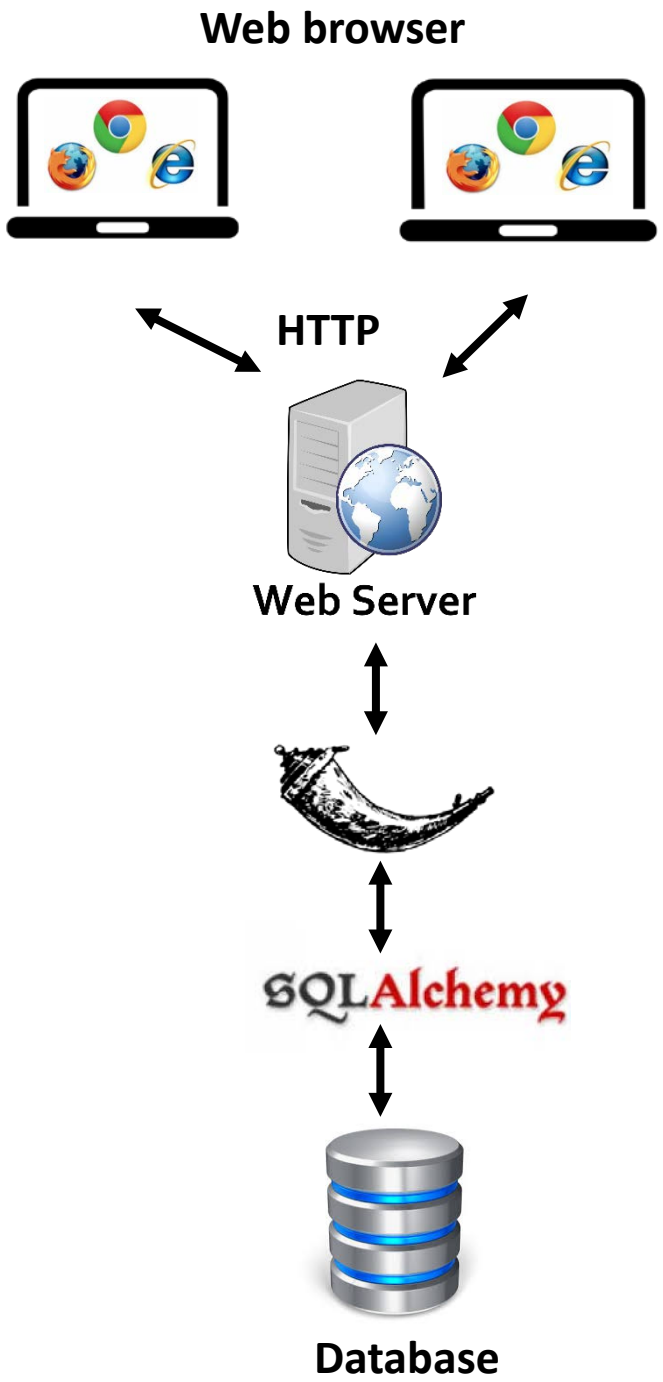
Building Web Application

Aims of this lecture

- Recap some important elements of Flask
  - Flask Web application
  - Flask Forms



Assessment 2 – Due Sept 15, 11:59 PM

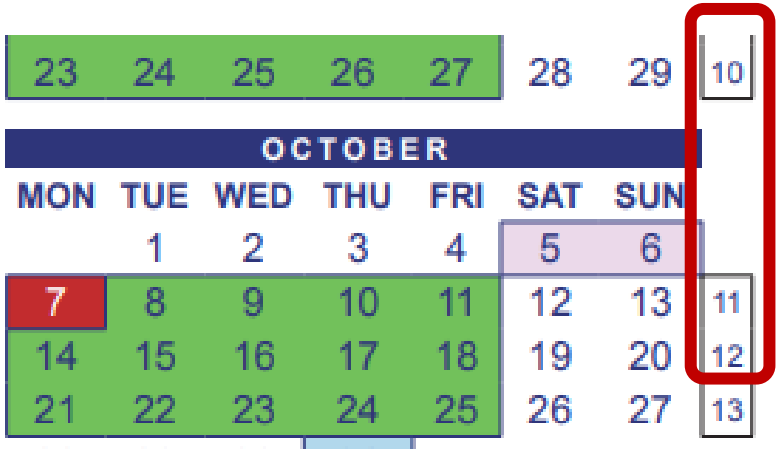
- Create **static** HTML pages
  - Use Bootstrap for styling
  - Use of CSS separately is not necessary

- **Four** pages
  - Landing or Main Page
  - Item details page
  - Item creation page
  - Manage items page

- Support navigation using tags

Assessment 3 - Released

- Break it down into tasks – Have a plan
  - Create your templates (Reuse code from your assessment 2)
    - View Item
    - Landing page
  - Create the database (models.py)
    - Assessment contains information about the 4 Database objects
  - Create Forms (forms.py)
    - Item Form
    - User login
    - User Registration
  - Incrementally update the view functions (views.py)
    - Bid Item - simple implementation
    - Manage items
    - Mark as sold – simple implementation
- Use the Workshop code as a reference
  - Bootstrap, HTML, Flask functions



Workshop time for your assessment

- Directly deploy your application on Heroku in Week 11 or Week 12
- User authentication on your application

Why are we using Flask?

- URL routing
  - Some action to be taken when a URL on Web server is accessed
- HTML templating
  - Use HTML templates and fill/generate some content
  - Use Forms to support user input
- Database support
  - Extension with SQLAlchemy
- Support some degree of security
  - User authentication, session encoding, csrf tokens

Build the application using the following folder structure

```
├─ development
│   ├─ travel
│   │   ├─ __init__.py

│   │   ├─ views.py
│   │   ├─ forms.py
│   │   ├─ models.py
│   │   └─ users.py

│   │   ├─ static

│   │   ├─ templates
│   │   │   ├─ index.html
│   │   │   ├─ error.html
```

Package folder

Package file

1. functions,
2. forms
3. database objects
4. authentication

static folder – all images, css files

HTML templates

You can have subfolders in templates directory to organize your HTML files

Flask & Flask Extensions

1. flask — URL routing, HTML Templating
2. flask-wtf — Forms
3. flask-bootstrap — Forms
4. flask-sqlalchemy — Database
5. flask-login — Security

Basic Flask application

- Import Flask from the package

```
from flask import Flask
```

- Create the Flask application
- Set some of its attributes

```
def create_app():
    app=Flask(__name__)
    app.debug=True
    app.secret_key='thisisasecretkey122'
```

- Every function, class, object that is not defined in the module (python file) has to be imported

```
    from .views import mainbp
    app.register_blueprint(mainbp)
```

- Registering a Blueprint
  - The Blueprint contains the URL routes

```
    return app
```

From a module in the current package
.views → import mainbp

URL routing

- Import Blueprint

```
from flask import Blueprint
```

- Create the Blueprint

```
# name: first argument is the blueprint name
# second argument – name of the module

mainbp = Blueprint('main', __name__)
```

- Specify the URL route
  - Every URL route should have a function
  - Functions are called ‘view’ functions in Flask terminology

```
@mainbp.route('/')
def index():
    str='<h1>hello world</h1>'
    return str
```

- View function returns a response
  - Response can be an HTML string

```
@mainbp.route('/<id>')
def viewfunction(id):
```

Demo – URL routing and HTML Template

- Create a Flask application
  - Blueprint

- Render an HTML form
  - Access the request object

HTML Templating

- Looks for HTML files under the templates directory

```
from flask import Blueprint, render_template
```

- Import the relevant functions, classes

- New route `/login` that returns a the response `login.html`

```
@mainbp.route('/login')
def login():
        return render_template('login.html')
```

Access HTTP request

- Every view function has access to HTTP request
- User inputs are passed through HTTP request
  - `request : Object`
  - `values : Dictionary`
- URL parameters – GET method
- Form – POST method

```
from flask import ( Blueprint,
        render_template, request )


def login():
    # access any request parameter
        print(request.values.get('email'))


    # access request parameter passed through
     URL (GET method)
        print(request.args.get('pwd'))


    # access request parameter passed through
     form (POST method)
        print(request.form.get('email'))
```

Questions?

Template Reuse – Template inheritance

- Reuse some HTML content
- Create a base.html file which will have reusable HTML content
- HTML should be complete

```
<!doctype HTML>
<HTML>
<header>
<H1> Header </H1>

{% block header %}
{% endblock %}

</header>

<body>
<H1> Content </H1>

{% block content %}
{% endblock %}

</body>

</HTML>
```

Reused HTML

Place holder

Reused HTML

Place holder

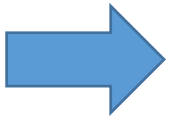
Reused HTML

Template Reuse – Template inheritance

- Extend from base.html

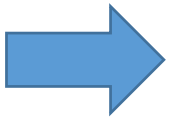
```
{% extends 'base.html' %}
```

- Add HTML content in the header placeholder



```
{% block header %}
<h3> Header from sample file </h3>
{% endblock %}
```

- Add HTML content in the content place holder



```
{% block content %}
<h3> Content in sample file </h3>
{% endblock %}
```

Demo – Template inheritance

- Create a base.html
  - Add reusable code
  - Add placeholders

- Create an HTML file
  - Add html code within the placeholders

Flask utility functions

- `url_for(viewfunction_string)`
  - Takes as input the view function name
  - `BlueprintName.viewfunctionName`
  - Parameters of the view function can be passed

```
url_for(‘index')

url_for('destination.show)

url_for('destination.show', id=1)
```

- `redirect(urlstring)`
  - Direct to another URL

Template – Variable Passing

- Passing dynamic content to the HTML file
- Create object/string in view function (python code)

```python
@mybp.route('/trial')
def trial():
    #pass the HTML and variables
    return render_template('trial.html',
        lecture_name='IAB207', workshop=8)
```

- Pass it to the template rendering function (render_template)
- Access it in the HTML file

```html
<!DOCTYPE html>
<html>
<body>
<p>{{ lecture_name }}</p>
<p>The unit has {{workshop}} workshops.</p>
</body>
</html>
```

Demo – Template Variables

- Create an Object
  - Add some attributes
- Use render_template to pass the object
- Use the object.attribute in the HTML

Questions?

Flask Forms

- Create a class that that is derived from FlaskForm

```
class ContactForm(FlaskForm):
        user_name = StringField('Name' )
        email = StringField('Email Address')
        submit = SubmitField("Submit")
```

- Add all fields that require user input
  - Different Field types supported by Flask Form

```
class DestinationForm(FlaskForm):
        name = StringField('Country',
                validators=[InputRequired()])

        description = TextAreaField('Description',
                        validators=[InputRequired()])

        image = StringField('Cover Image',
                        validators=[InputRequired()])

        submit = SubmitField("Create")
```

- Add a Submit Field

Render FlaskForm

- Use Bootstap to make it easy,
- Any form can be rendered using this HTML
- Use render_template to pass the form and heading

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

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

{% block content %}
  <div class="container">
     <div class="col-md-12">

        <h1>{{heading}}</h1>

        {{wtf.quick_form(form)}}

      </div>
  </div>

{% endblock %}
```

```
return render_template('create.html',
                       heading='Create destination', form=form)
```

Accessing User Inputs

Support multiple request types

```
@bp.route('/create', methods = ['GET', 'POST'])
def create():
        form = DestinationForm()
        if form.validate_on_submit():
                # if the form was successfully submitted
                # access the values in the form data
                print(form.name.data)

        return render_template('destinations/create.html', form=form)
```

Access form data from : `form.fieldname.data`

Use this to access form data

Change the POST URL of the form

Posting the Form to the same URL that created the form /destinations/1.

```
{{  wtf.quick_form(form)   }}
```

Posting the Form to a different URL

```
{{   wtf.quick_form(form, "/destinations/{0}/comment".format(1))     }}
```

What is the output of this format function?

```
"/destinations/{0}/comment".format(1))
```

Questions?

Summary

- Recap some important elements of Flask
  - Flask Templates
  - Flask Forms

Thank you!