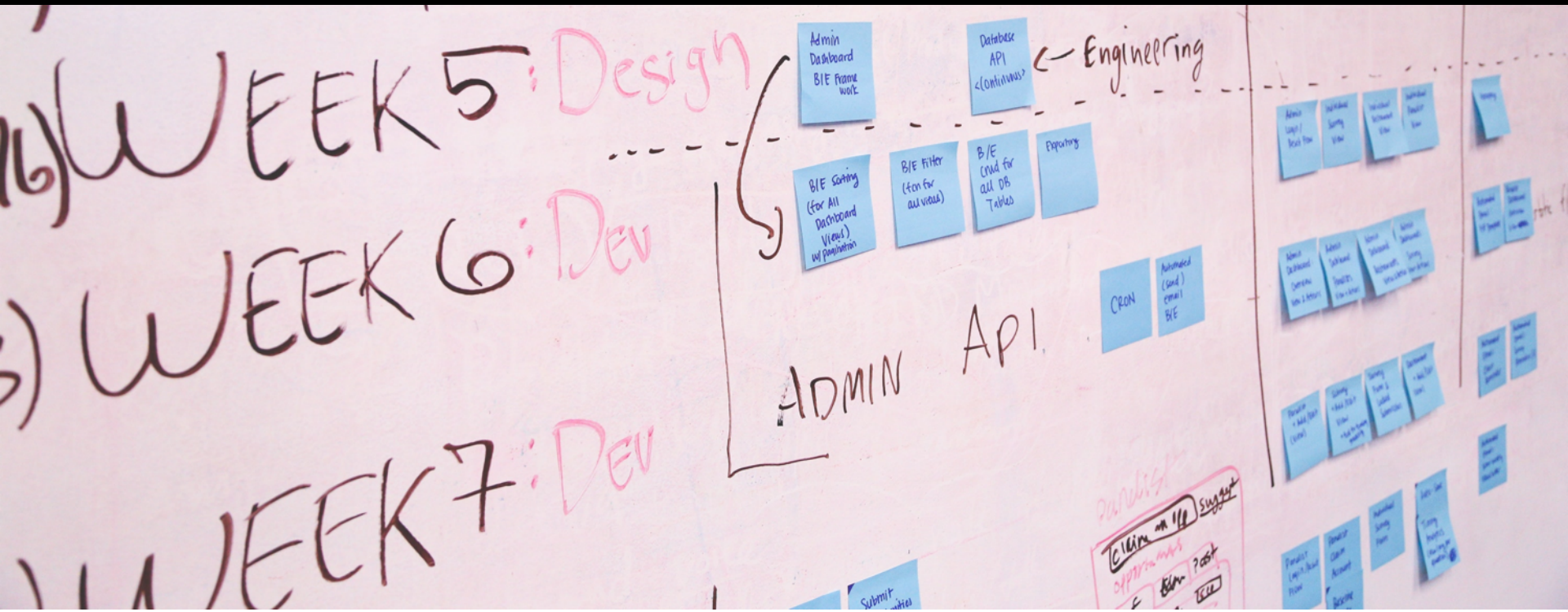


COM5961 DATA DRIVEN PRODUCTS & SERVICES DESIGN:
LESSON 9 - CODE TESTING AND INTEGRATION

Bernard Suen
Center for Entrepreneurship
Chinese University of Hong Kong

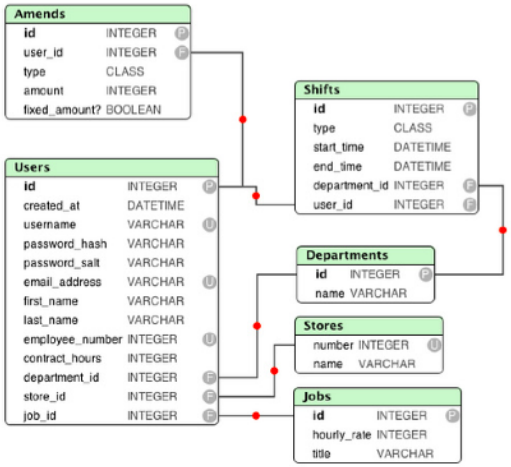
Today's Agenda (All lab practices)

1. Flask SQLAlchemy versions for local JupyterNotebook and remote PythonAnywhere development
2. Introduction to standard HTML form object and Bootstrap form.
3. Understanding HTTP request methods and CRUD database operations mapping.
4. Walkthrough of a basic Flask CRUD app with Bootstrap components for template development.
5. Code testing and integration.

Flask SQLAlchemy versions for local Jupyter Notebook and remote PythonAnywhere development

Source: commons.wikimedia.org

Source: Caius Durling (Flickr)



M(odel)
(SQL/NoSQL data models and processing functions)

V(iew)
(Flask templates)

Routing control to pre-defined data and functions

Source: pexels.com

C(ontroller)
(e.g. Routes, rights and authentication)

Testing version compatibility (See Jupyter Notebook).

Jupyter Notebook

Check installed packages (including those come with Jupyter and those installed by you) and versions

```
In [ ]: !pip list
```

```
# Compatible versions
# Flask 2.2.2
# Flask-SQLAlchemy 2.5.0 (version 3.0.0 is incompatible)
# SQLAlchemy 1.3.22
```

PythonAnywhere

These versions used on Jupyter Notebook and PythonAnywhere have been tested and are working fine.

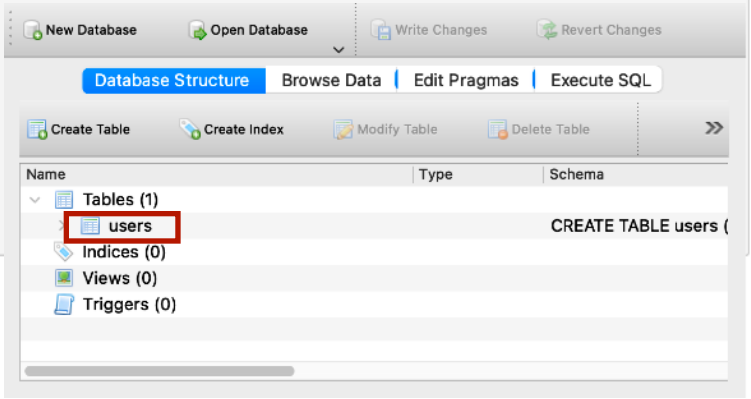
3.9	Flask	2.0.0	A simple framework for building complex web applications.
3.9	Flask-SQLAlchemy	2.4.4	Adds SQLAlchemy support to your Flask application.
3.9	SQLAlchemy	1.3.22	Database Abstraction Library

https://www.pythonanywhere.com/batteries_included/

Create database

In [15]:
```
from flask import Flask                                    # Import the Flask module from the flask package
from flask_sqlalchemy import SQLAlchemy                    # Import the SQLAlchemy module from
                                                           # the flask_sqlalchemy package
app = Flask(__name__)                                      # Setup the Flask app
db = SQLAlchemy(app)                                       # Setup the SQLAlchemy db object to work with app
app.config['SQLALCHEMY_DATABASE_URI']  = 'sqlite:///demo.db'   # Configure the 'SQLALCHEMY_DATABASE_URI' to
                                                           # link to the SQLite db file called demo.db

app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False       # Disable modification track
app.config['SECRET_KEY'] = 'mysecretkey'                   # Setup SECRET_KEY for session cookies tracking


class User(db.Model):                                      # Define User table class and call it 'users'
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)  # Define primary key id.
    username = db.Column(db.String(50))                    # Define other fields.
    email = db.Column(db.String(255))
    desc = db.Column(db.String(255))
    password = db.Column(db.String(80))

try:
    db.create_all()
    print("Database successfully created.")
except Exception as e:
    print("Exception occurred.",e)
```

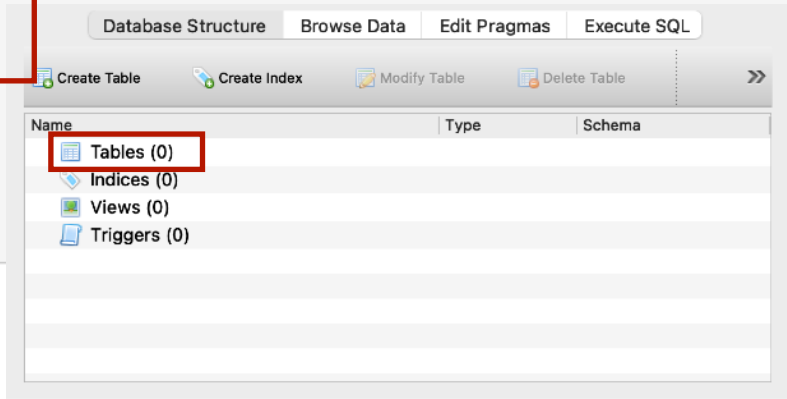
```
Database successfully created.
```



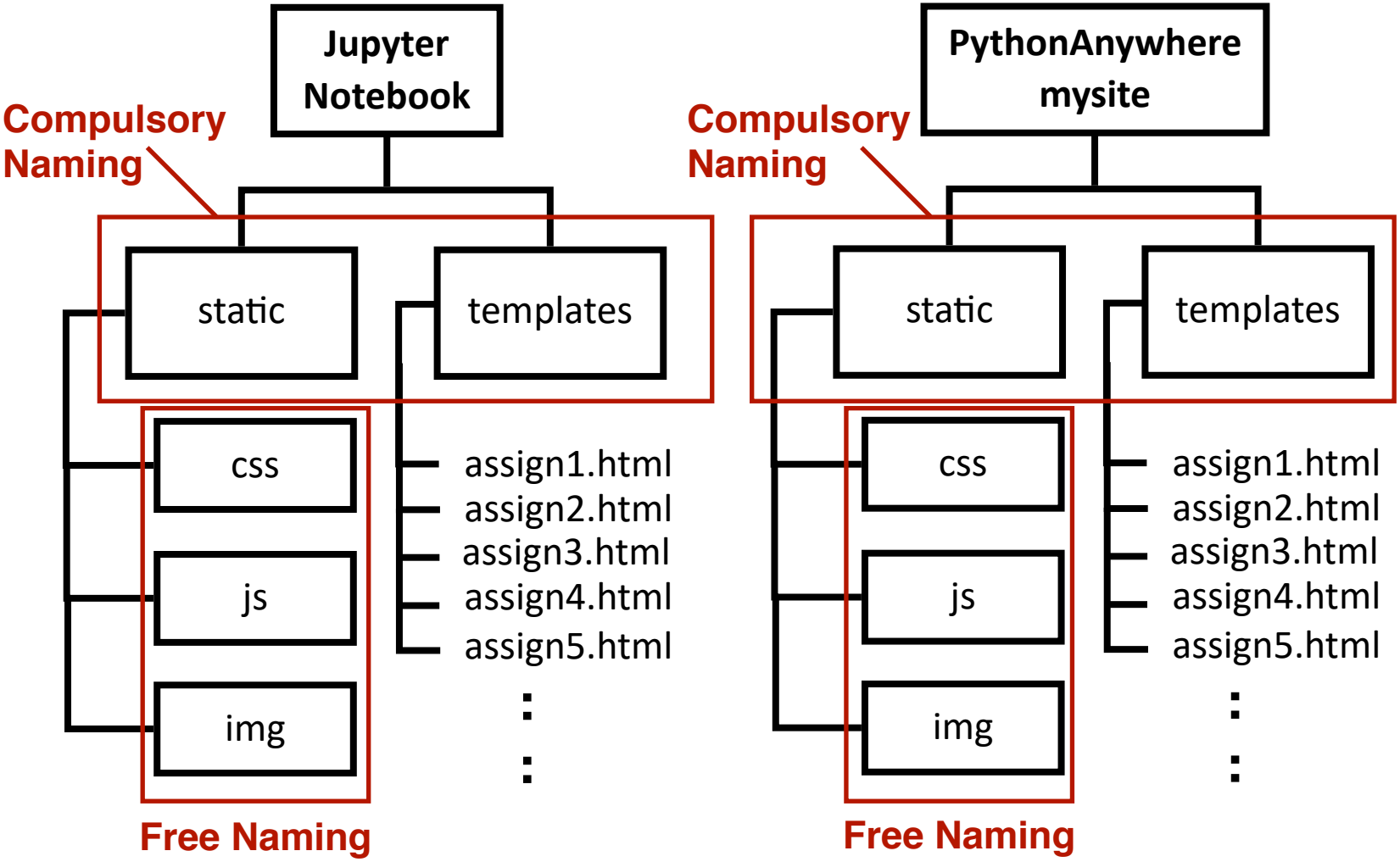
Delete (drop) database using SQLAlchemy

In [2]:
```
from flask import Flask                                          # Import the Flask module from the flask package
from flask_sqlalchemy import SQLAlchemy                          # Import the SQLAlchemy module from
                                                                 # the flask_sqlalchemy package
app = Flask(__name__)                                            # Setup the Flask app
db = SQLAlchemy(app)                                             # Setup the SQLAlchemy db object to work with app
app.config['SQLALCHEMY_DATABASE_URI']  = 'sqlite:///demo.db'     # Configure the 'SQLALCHEMY_DATABASE_URI' to
                                                                 # link to the SQLite db file called demo.db

app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False             # Disable modification track
app.config['SECRET_KEY'] = 'mysecretkey'                         # Setup SECRET_KEY for session cookies tracking


class User(db.Model):                                            # Define User table class and call it 'users'
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)  # Define primary key id.
    username = db.Column(db.String(50))                          # Define other fields.
    email = db.Column(db.String(255))
    desc = db.Column(db.String(255))
    password = db.Column(db.String(80))

try:
    result = db.engine.execute('DROP TABLE IF EXISTS users')
    print("Table successfully dropped.")
except Exception as e:
    print("Exception occurred.",e)
```

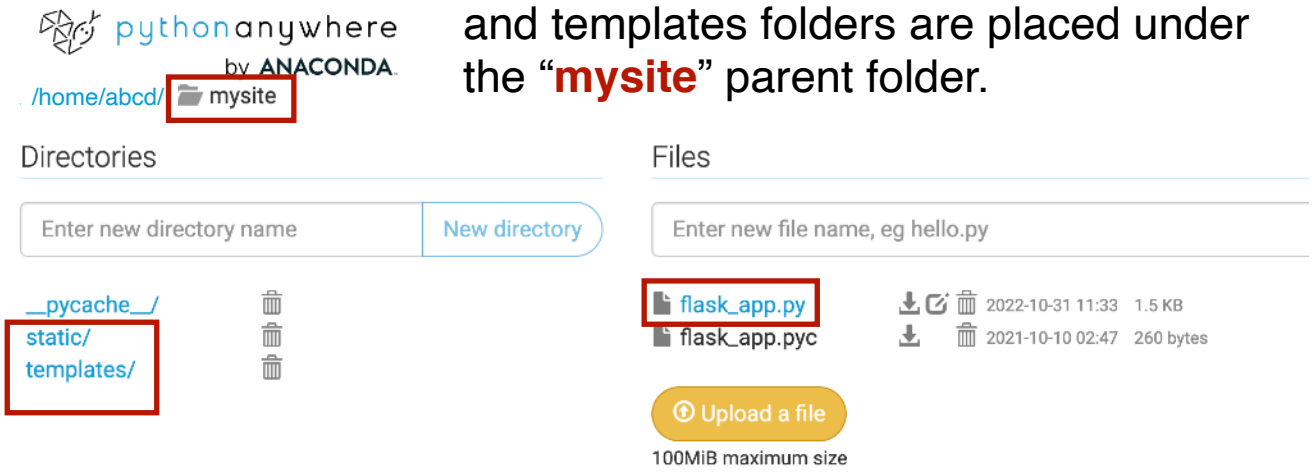
```
Table successfully dropped.
```



Directory structure comparison.


Jupyter Notebook
Compulsory Naming
static
templates
css
js
img
assign1.html
assign2.html
assign3.html
assign4.html
assign5.html
Free Naming
PythonAnywhere mysite
Compulsory Naming
static
templates
css
js
img
assign1.html
assign2.html
assign3.html
assign4.html
assign5.html
Free Naming

For easy comparison, all supporting static and templates folders are placed under the directory tree “/home/abcd/mysite” parent folder.



On PythonAnywhere, all supporting static and templates folders are placed under the “**mysite**” parent folder.

Example #1 (See Jupyter Notebook)

jupyter base.html 19 hours ago

Logout

File Edit View Language HTML

Use of base template.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="{{url_for('static', filename='css/style.css')}}">
</head>
<body>
<header><!--header section-->
<!-- <label id="menu" for="my_checkbox">Click For Menu</label> -->
<label id="menu" for="my_checkbox">
    <span class="bar"></span>
    <span class="bar"></span>
    <span class="bar"></span>
</label>
</header>
<input type="checkbox" id="my_checkbox">
<nav id="hidden">
    <ul>
        <li {% if active == 1 %} class="active"{% endif %}><a href="/assign1">Assignment 1</a></li>
        <li {% if active == 2 %} class="active"{% endif %}><a href="/assign2">Assignment 2</a></li>
        <li {% if active == 3 %} class="active"{% endif %}><a href="/assign3">Assignment 3</a></li>
        <li {% if active == 4 %} class="active"{% endif %}><a href="/assign4">Assignment 4</a></li>
        <li {% if active == 5 %} class="active"{% endif %}><a href="/assign5">Assignment 5</a></li>
    </ul>
</nav>
<main>
{% block header %}
{% endblock %}
{% block content %}
{% endblock %}
</main>
<footer><!--footer region--></footer>
</body>
</html>
```

Use of url_for function for path reference.

Use of conditional and template variables.

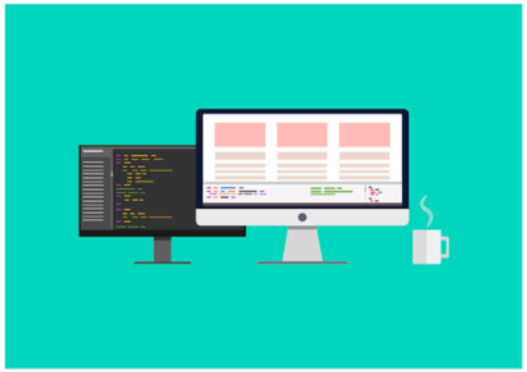
Use of url_for Reference

<link rel="stylesheet" href="{{url_for('static', filename='css/style.css')}}">

<img class="…" src="{{url_for('static', filename='….')}}" alt="">

<script src="{{ url_for('static', filename='js/scripts.js') }}"></script>

Assignment 1



This is my first experience in creating a web page and sharing with the world. It was unbelievable that I can do that as a person coming from a non-technical background. After all, the learning hasn't been as difficult as I thought. Hopefully, I can do more interesting things with the newly acquired skills.

My current design for the first assignment is based on one main idea: How can I make my creative works more accessible and inspiring for other people? That's why I have chosen a simple horizontal navigation bar with the current navigation option highlighted and the paragraph heading displayed in H1 tag.

Simplicity is important but amusement, the entertaining impact my works create, is equally so. I still have to work harder on both. They are design attributes that can delight people and make them willing to see my works more.

More details about my design can be found here.

My Journals

Making Remote Works Collaborative and Accountable

Rethink Digital Transformation

```python
@app.route("/")
def index():
    return render_template('index.html')

@app.route("/assign1")
def assign1():
    return render_template('index.html')

@app.route("/assign2")
def assign2():
    return render_template('assign2.html')

@app.route("/assign3")
def assign3():
    return render_template('assign3.html')

@app.route("/assign4")
def assign4():
    return render_template('assign4.html')

@app.route("/assign5")
def assign5():
    return render_template('assign5.html')
```

Assign value 1 to the variable “active” and pass it to the template as a template variable.

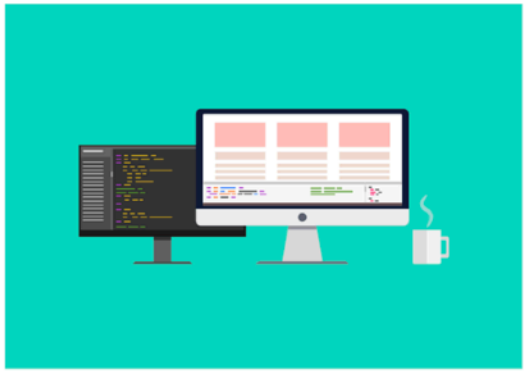
```python
@app.route("/assign1")
def assign1():
    active = 1
    return render_template('index.html', active=active)

@app.route("/assign2")
def assign2():
    active = 2
    return render_template('assign2.html', active=active)

@app.route("/assign3")
def assign3():
    active = 3
    return render_template('assign3.html', active=active)

@app.route("/assign4")
def assign4():
    active = 4
    return render_template('assign4.html', active=active)

@app.route("/assign5")
def assign5():
    active = 5
    return render_template('assign5.html', active=active)
```

```
<nav id="hidden">
    <ul>
        <li class="active"><a href="/assign1">Assignment 1</a></li>
        <li><a href="/assign2">Assignment 2</a></li>
        <li><a href="/assign3">Assignment 3</a></li>
        <li><a href="/assign4">Assignment 4</a></li>
        <li><a href="/assign5">Assignment 5</a></li>
    </ul>
</nav>
```

The template variable being passed from the back-end will be tested inside the template code using if conditional to determine if the CSS class should be activated to highlight the menu option.

```
<nav id="hidden">
    <ul>
        <li {% if active == 1 %} class="active"{% endif %}><a href="/assign1">Assignment 1</a></li>
        <li {% if active == 2 %} class="active"{% endif %}><a href="/assign2">Assignment 2</a></li>
        <li {% if active == 3 %} class="active"{% endif %}><a href="/assign3">Assignment 3</a></li>
        <li {% if active == 4 %} class="active"{% endif %}><a href="/assign4">Assignment 4</a></li>
        <li {% if active == 5 %} class="active"{% endif %}><a href="/assign5">Assignment 5</a></li>
    </ul>
</nav>
<main>
{% block header %}
{% endblock %}
{% block content %}
{% endblock %}
</main>
```

Assignment 1



This is my first experience in creating a web page and sharing with the world. It was unbelievable that I can do that as a person coming from a non-technical background. After all, the learning hasn't been as difficult as I thought. Hopefully, I can do more interesting things with the newly acquired skills.

My current design for the first assignment is based on one main idea: How can I make my creative works more accessible and inspiring for other people? That's why I have chosen a simple horizontal navigation bar with the current navigation option highlighted and the paragraph heading displayed in H1 tag.

Simplicity is important but amusement, the entertaining impact my works create, is equally so. I still have to work harder on both. They are design attributes that can delight people and make them willing to see my works more.

More details about my design can be found here.

My Journals

Making Remote Works Collaborative and Accountable

Rethink Digital Transformation

Example #2 (See Jupyter Notebook)

Man must explore, and this is exploration at its greatest

Problems look mighty small from 150 miles up

Posted by Start Bootstrap *on September 24, 2021*

I believe every human has a finite

blog.html

```
{% block content %}
<!-- Main Content-->
<div class="container">
    <div class="row">
        <div class="col">
            <b>Name</b>
        </div>
        <div class="col">
            <b>Jan</b>
        </div>
        <div class="col">
            <b>Feb</b>
        </div>
        <div class="col">
            <b>Mar</b>
        </div>
    </div>
    {% for row in  entries  %}
    <div class="row">
        <div class="col">
                {{ row["Student"] }}
        </div>
        <div class="col">
                {{ row["Jan"] }}
        </div>
        <div class="col">
                {{ row["Feb"] }}
        </div>
        <div class="col">
                {{ row["Mar"] }}
        </div>
    </div><!-- end row -->
    {% endfor %}
</div><!-- end container -->
<hr />
 </div>
{% endblock %}
```

Read data from “weekly_hours.csv” into pandas data frame object and transform into a list of dictionary object for passing to the “blog.html” template as template variable “entries”.

App.ipyb

```
@app.route("/")
def index():
    return render_template('index.html')

@app.route("/blogs")
def blogs():
    df = pd.read_csv("weekly_hours.csv")
    list = df.to_dict('records')
    return render_template('blogs.html', entries = list)
```

Each Entry loops through the template as “row” variable.

```
1 Jan,Feb,Mar,Student
2 85,90,88,David Chan
3 75,72,999,Peter Wong
4 80,60,100,John Lui
```

weekly_hours.csv

Name	Jan	Feb	Mar
David Chan	85	90	88
Peter Wong	75	72	999
John Lui	80	60	100

Converting CSV into SQL data through code (See Jupyter Notebook).

Read data from CSV file "weekly_hours.csv" into pandas dataframe "df".

In [5]:
```
import pandas as pd
df = pd.read_csv('weekly_hours.csv')
list = df.to_dict('records')          # Assign the dataframe to a list
print(list)                           # Print it out to see the data structure
```

```
[{'Jan': 85, 'Feb': 90, 'Mar': 88, 'Student': 'David Chan'}, {'Jan': 75, 'Feb': 72, 'Mar': 999, 'Student': 'Peter Won
g'}, {'Jan': 80, 'Feb': 60, 'Mar': 100, 'Student': 'John Lui'}]
```

App.ipyb

```
@app.route("/")
def index():
    return render_template('index.html')

@app.route("/blogs")
def blogs():
    df = pd.read_csv("weekly_hours.csv")
    list = df.to_dict('records')
    return render_template('blogs.html', entries = list)
```

List of dictionaries passed to template “blogs.html”.

Read data from CSV file into a dataframe and output the database to a SQLite database.

In [48]:
```
import pandas as pd                                    # Import pandas library package
import sqlite3                                         # Import SQLite3 library package
con = sqlite3.connect('demo.db')                       # Connect to SQLite database using sqlite3 connect method
# load the data into a Pandas DataFrame.
weekly_hours = pd.read_csv('weekly_hours.csv')         # Read csv into a panda dataframe called weekly_hours
cursor = con.cursor()                                  # Setup record pointer cursor to prepare execution
drop_sql = "DROP TABLE IF EXISTS weekly_hours"         # Drop weekly_hours table to prepare for new import
try:
    cursor.execute(drop_sql)                           # Execute drop_sql string which contains "DROP TABLE IF ..."
    con.commit()
    print("Table successfully dropped.")
except Exception as e:
    print("Exception occurred.",e)

# write the data to a sqlite table.
weekly_hours.to_sql('weekly_hours', con, if_exists='append', index = False) # Save weekly_hours
print("Table successfully created.")                   # dataframe to SQL table
sql = "SELECT * FROM weekly_hours"                     # Set up SQL query statement
df = pd.read_sql_query(sql,con)                        # Read result into dataframe
con.close()                                            # Close connection
df                                                     # Display data frame
```

```
Table successfully dropped.
Table successfully created.
```

Out[48]:

	Jan	Feb	Mar	Student
0	85	90	88	David Chan
1	75	72	999	Peter Wong
2	80	60	100	John Lui

Converting data from list of tubles to list of dictionaries

In [11]:

```
from flask import Flask                                          # Import the Flask module from the flask package
from flask_sqlalchemy import SQLAlchemy                          # Import the SQLAlchemy module from
                                                                 # the flask_sqlalchemy package
app = Flask(__name__)                                            # Setup the Flask app
db = SQLAlchemy(app)                                             # Setup the SQLAlchemy db object to work with app
app.config['SQLALCHEMY_DATABASE_URI']  = 'sqlite:///demo.db'     # Configure the 'SQLALCHEMY_DATABASE_URI' to
                                                                 # link to the SQLite db file called demo.db

app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False             # Disable modification track
app.config['SECRET_KEY'] = 'mysecretkey'                         # Setup SECRET_KEY for session cookies tracking

class User(db.Model):                                            # Define User table class and call it 'users'
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True) # Define primary key id.
    username = db.Column(db.String(50))                          # Define other fields.
    email = db.Column(db.String(255))
    desc = db.Column(db.String(255))
    password = db.Column(db.String(80))

try:
    result = db.engine.execute('select * from weekly_hours')
    print( 'Table successfully selected.')
except Exception as e:
    print("Exception occurred.",e)

weekly_hours = []
for i in result:          # Loop through SQL query result and add it to a users list
    weekly_hours.append(i)
print(weekly_hours)
dataset = []                        # Define a list called dataset
wk_hours_rec={}                  # Define a dictionary called user_rect to store individual user record
for i in weekly_hours:           # Loop through the weekly_hours list to assign each list element to the individual wee
    wk_hours_rec['Jan'] = i[0]       # Field by field assignment for storing list element as dictionary value pair
    wk_hours_rec['Feb'] = i[1]       # four value pairs
    wk_hours_rec['Mar'] = i[2]
    wk_hours_rec['Student'] = i[3]
    # print(i)                       # Print out each wk_hours_rec list element
    # print(user_rec)                # Print out each wk_hours_rec value pair
    dataset.append(wk_hours_rec.copy())# Append each wk_hours_rec record to the dataset list
print(dataset)
```

List of tuples

List of dictionaries

```
Table successfully selected.
[(85, 90, 88, 'David Chan'), (75, 72, 999, 'Peter Wong'), (80, 60, 100, 'John Lui')]
[{'Jan': 85, 'Feb': 90, 'Mar': 88, 'Student': 'David Chan'}, {'Jan': 75, 'Feb': 72, 'Mar': 999, 'Student': 'Peter Won
g'}, {'Jan': 80, 'Feb': 60, 'Mar': 100, 'Student': 'John Lui'}]
```

List of tuples

List of dictionaries

blog.html

Read data from “demo.db” into list of tuble.

App.ipyb

```
@app.route("/blogs")
def blogs():
    result = db.engine.execute('select * from weekly_hours')
    weekly_hours = []
    for i in result:          # Loop through SQL query result
        weekly_hours.append(i)
        dataset = []                  # Define a list called
        wk_hours_rec={}               # Define a dictionary ca
    for i in weekly_hours:            # Loop through the week
        wk_hours_rec['Jan'] = i[0]        # Field by field ass
        wk_hours_rec['Feb'] = i[1]        # Four value pairs
        wk_hours_rec['Mar'] = i[2]
        wk_hours_rec['Student'] = i[3]
        dataset.append(wk_hours_rec.copy())# Append each wk
    return render_template('blogs.html', entries = dataset)
```

List of dictionaries passed to “blogs.html” as entries.

```
{% block content %}
<!-- Main Content-->
<div class="container">
      <div class="row">
          <div class="col">
              <b>Name</b>
          </div>
          <div class="col">
              <b>Jan</b>
          </div>
          <div class="col">
              <b>Feb</b>
           </div>
          <div class="col">
              <b>Mar</b>
          </div>
      </div>
     {% for row in  entries  %}
     <div class="row">
          <div class="col">
                    {{ row["Student"] }}
          </div>
          <div class="col">
                    {{ row["Jan"] }}
          </div>
          <div class="col">
                    {{ row["Feb"] }}
          </div>
          <div class="col">
                    {{ row["Mar"] }}
          </div>
     </div><!-- end row -->
     {% endfor %}
</div><!-- end container -->
<hr />
 </div>
{% endblock %}
```

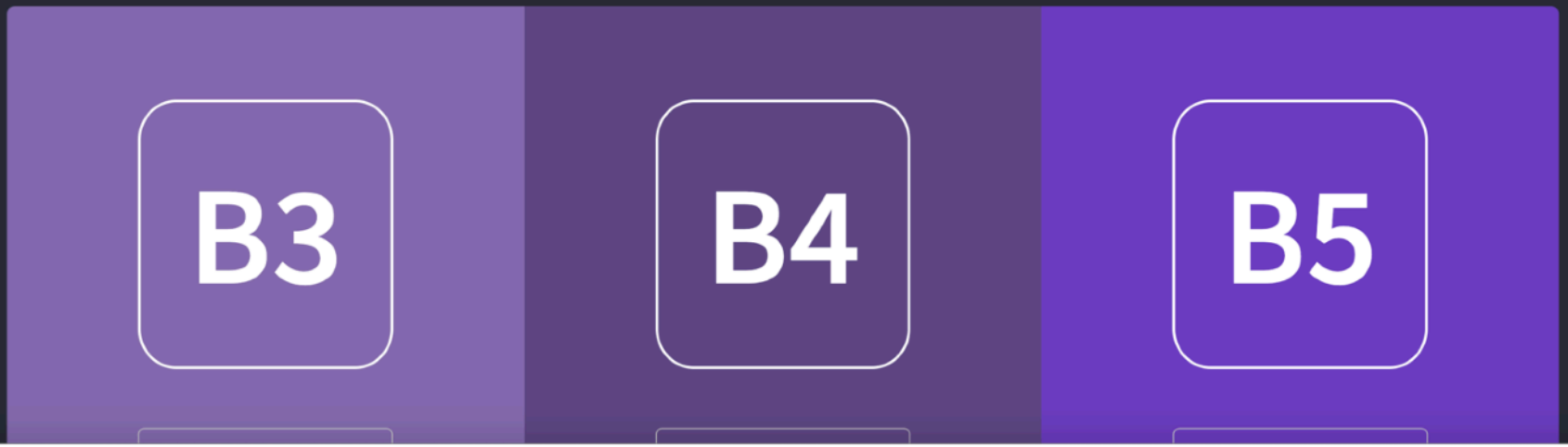
Each Entry loops through the template as “row” variable.

Name	Jan	Feb	Mar
David Chan	85	90	88
Peter Wong	75	72	999
John Lui	80	60	100

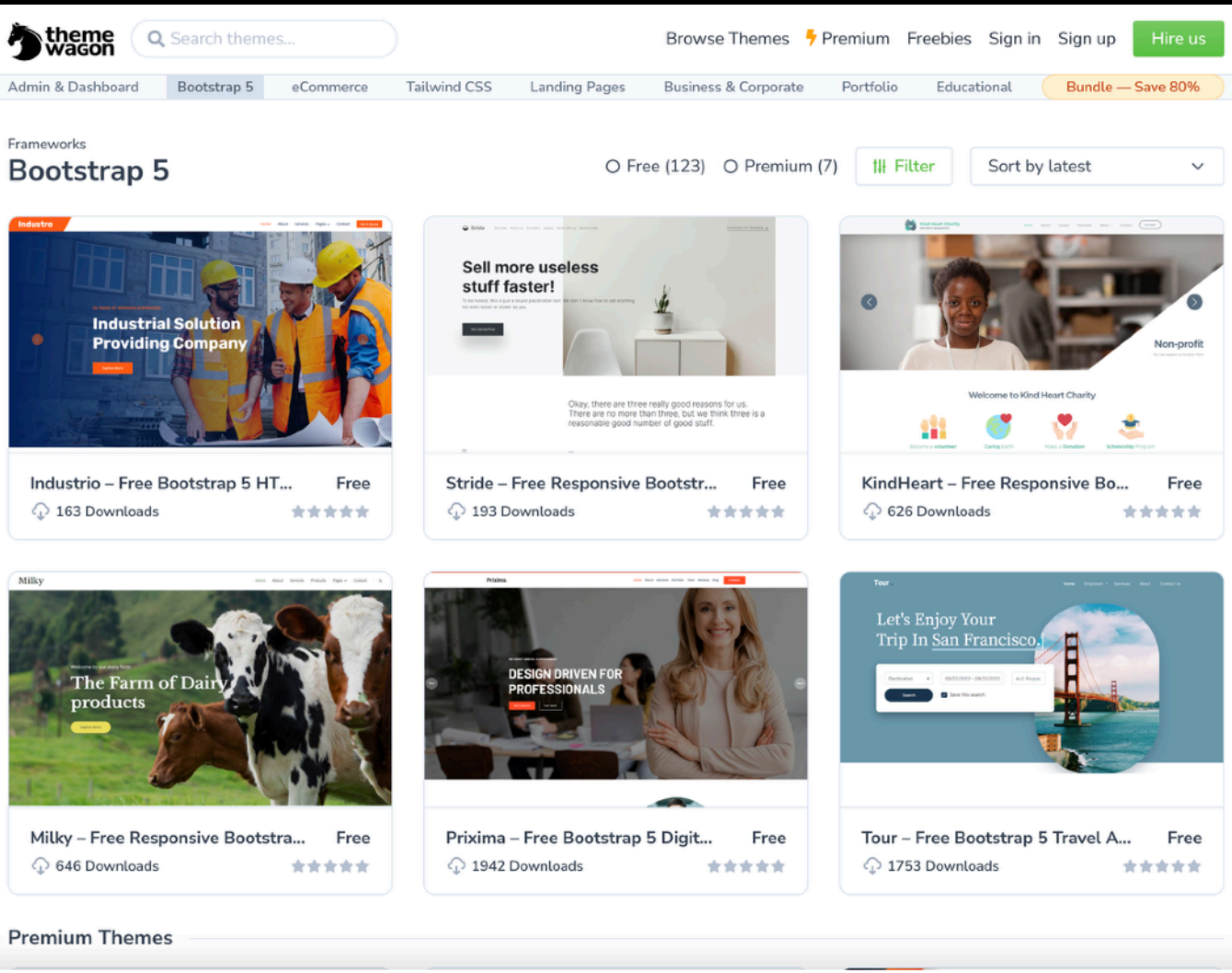
More Boostrap 5 Template Examples

Bootstrap Versions

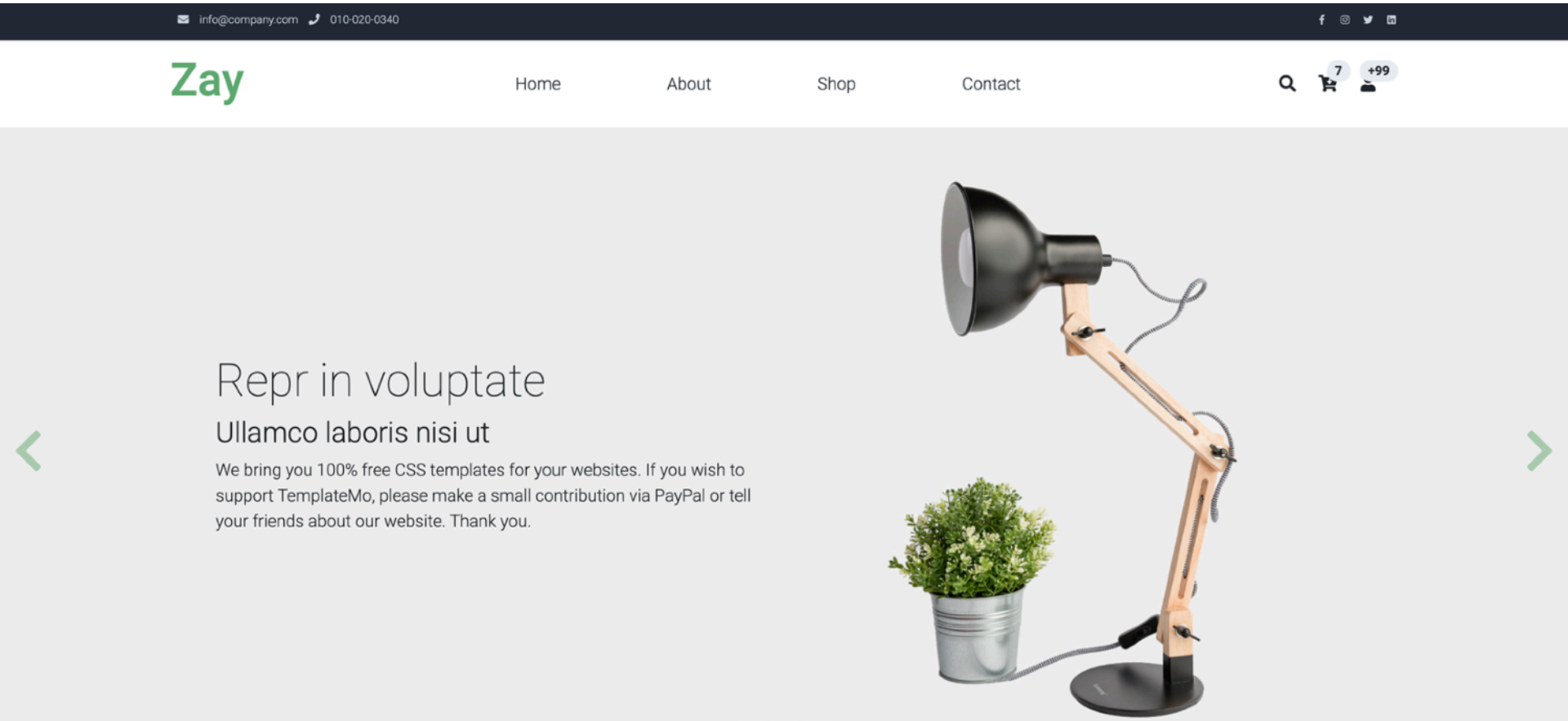
Bootstrap is the world's most famous free CSS framework.
You can choose between the following versions:



1. https://bootstraptaste.com/ (including different Bootstrap versions)
2. http://blacktie.co/ (including different Bootstrap versions)



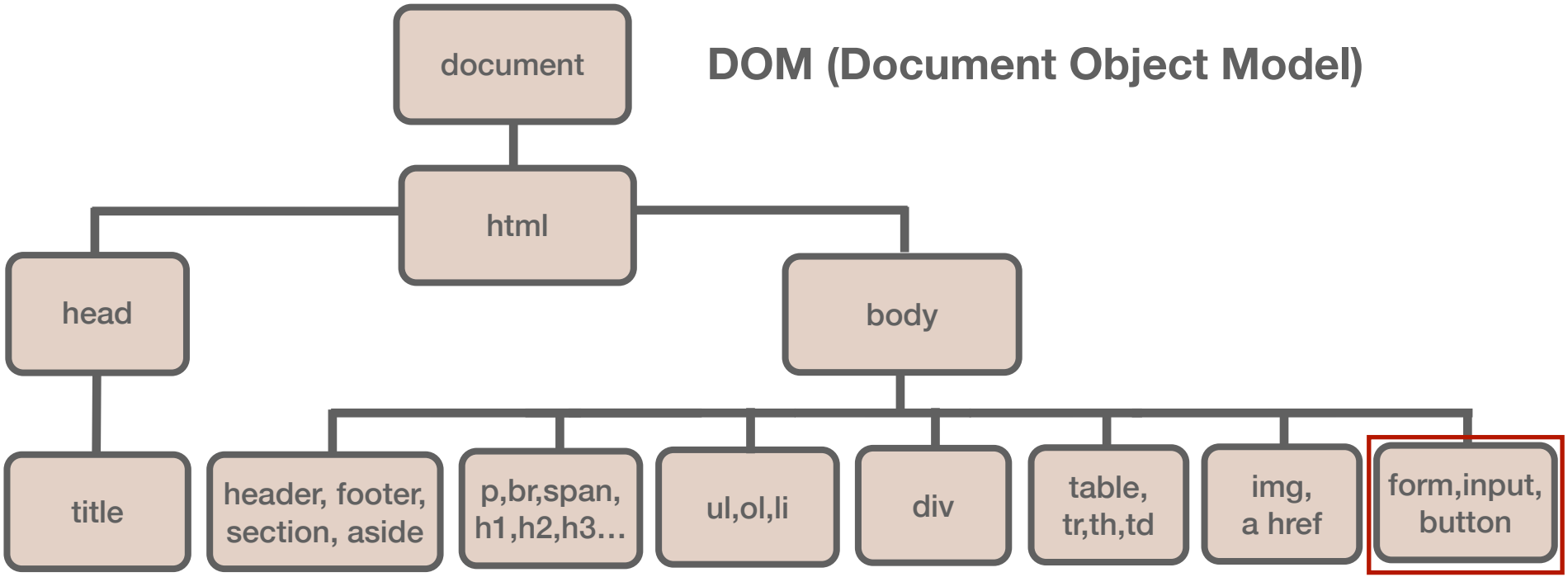
Source: https://themewagon.com/theme-framework/bootstrap-5/page/3/?swoof=1&really_curr_tax=28-pa_frameworks



https://technext.github.io/zay-shop/index.html

<img class=“…” src=“{{url_for(‘static', filename=‘….’)}}” alt=“”>

Introduction to standard HTML form object and Bootstrap form.



Two-way communications with the back-end.

https://www.runoob.com/htmldom/htmldom-tutorial.html

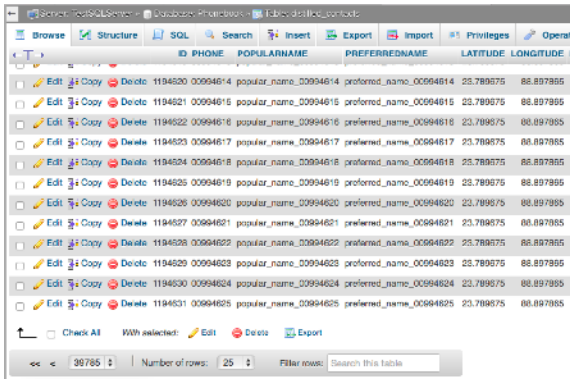
Browser
Screen Display

Web Server
HTML/CSS/JS

Database Server
Data

Dynamic Website (CMS)

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title> A Tiny HTML Document </title>
<link href ="styles.css" rel="stylesheet">
<script src="scripts.js"></script>
</head>

<body>
<p>Let's rock the browser, HTML5 style.</p>
</body>
</html>
```



Source: CMS Web Design: A Guide to Dynamic Content Applications

The HTML <form> Elements

The HTML <form> element can contain one or more of the following form elements:

<input>
<label>
<select><option>
<textarea>
<button>

Standard HTML form

HTML form, label, and input objects (basic text type).

Example

A form with input fields for text:

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname">
</form>
```

Try it Yourself »

This is how the HTML code above will be displayed in a browser:

First name:

Last name:

Source: https://www.w3schools.com/html/html_forms.asp

HTML form, label, and input objects (basic radio type).

Example

A form with radio buttons:

```
<p>Choose your favorite Web language:</p>

<form>
  <input type="radio" id="html" name="fav_language" value="HTML">
  <label for="html">HTML</label><br>
  <input type="radio" id="css" name="fav_language" value="CSS">
  <label for="css">CSS</label><br>
  <input type="radio" id="javascript" name="fav_language" value="JavaScript">
  <label for="javascript">JavaScript</label>
</form>
```

Try it Yourself »

This is how the HTML code above will be displayed in a browser:

Choose your favorite Web language:

- ○ HTML
- ○ CSS
- ○ JavaScript

Source: https://www.w3schools.com/html/html_forms.asp

HTML form, label, and input objects (basic checkbox type).

Example

A form with checkboxes:

```
<form>
  <input type="checkbox" id="vehicle1" name="vehicle1" value="Bike">
  <label for="vehicle1"> I have a bike</label><br>
  <input type="checkbox" id="vehicle2" name="vehicle2" value="Car">
  <label for="vehicle2"> I have a car</label><br>
  <input type="checkbox" id="vehicle3" name="vehicle3" value="Boat">
  <label for="vehicle3"> I have a boat</label>
</form>
```

Try it Yourself »

This is how the HTML code above will be displayed in a browser:

- [ ] I have a bike
- [ ] I have a car
- [ ] I have a boat

Source: https://www.w3schools.com/html/html_forms.asp

HTML form, label, and input objects (basic submit type).

Example

A form with a submit button:

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>
```

Try it Yourself »

This is how the HTML code above will be displayed in a browser:

First name:
John
Last name:
Doe

Submit

Source: https://www.w3schools.com/html/html_forms.asp

HTML textarea object.

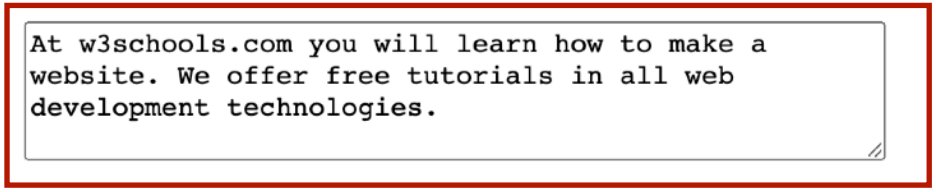
Example

A text area with a specified height and width:

```
<textarea rows="4" cols="50">
At w3schools.com you will learn how to make a website. We offer free tutorials in all web development technologies.
</textarea>
```

Try it Yourself »

The textarea rows and cols attributes



This textarea has 4 visible rows.

You can change that by changing the value of the "rows" attribute.

Source: https://www.w3schools.com/tags/att_textarea_rows.asp

HTML placeholder attribute.

Example

A text area with a placeholder text:

```
<textarea placeholder="Describe yourself here..."></textarea>
```

Try it Yourself »

Describe yourself here...

Source: https://www.w3schools.com/tags/att_textarea_placeholder.asp

Bootstrap form

Textarea

Comments:

Submit

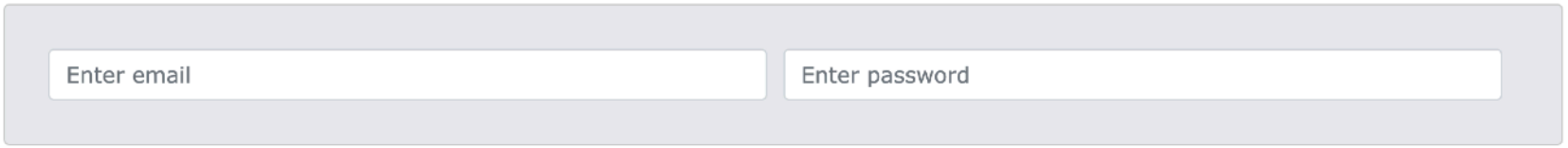
Example

```
<label for="comment">Comments:</label>
<textarea class="form-control" rows="5" id="comment" name="text"></textarea>
```

Try it Yourself »

Source: https://www.w3schools.com/bootstrap5/bootstrap_forms.php

Form Row/Grid (Inline Forms)



If you want your form elements to appear side by side, use `.row` and `.col`:

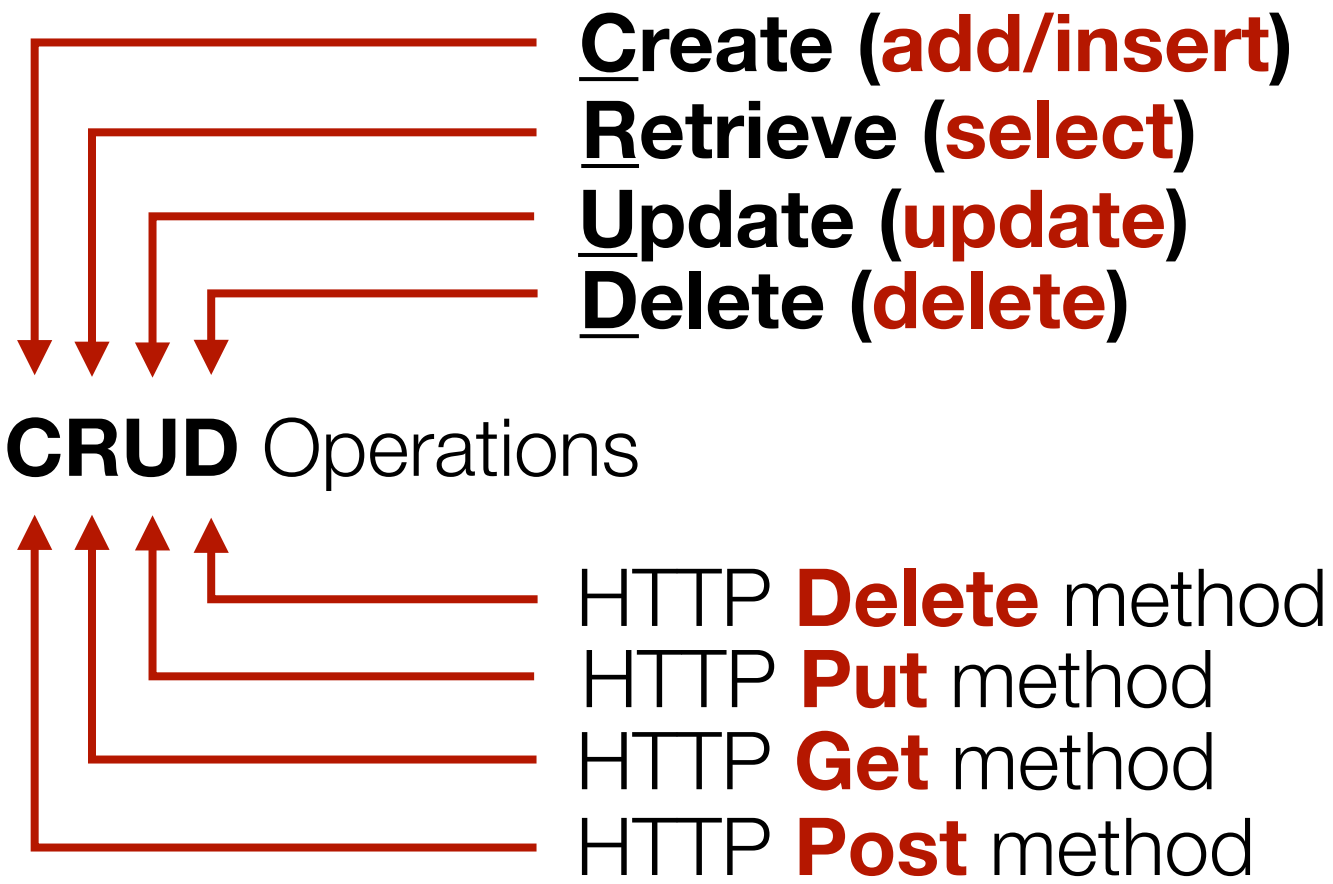
Example

```
<form>
  <div class="row">
    <div class="col">
      <input type="text" class="form-control" placeholder="Enter email" name="email">
    </div>
    <div class="col">
      <input type="password" class="form-control" placeholder="Enter password" name="pswd">
    </div>
  </div>
</form>
```

Try it Yourself »

Source: https://www.w3schools.com/bootstrap5/bootstrap_forms.php

Understanding HTTP request methods and CRUD database operations mapping.



https://www.w3schools.com/tags/ref_httpmethods.asp

The POST Method

POST is used to send data to a server to create/update a resource.

The GET Method

GET is used to request data from a specified resource.

The PUT Method

PUT is used to send data to a server to create/update a resource.

The DELETE Method

The DELETE method deletes the specified resource.

(C)reate MySQL Record Through SQLAlchemy Data Object

In [12]:

```
from flask import Flask                      # Import the Flask module from the flask package
from flask_sqlalchemy import SQLAlchemy      # Import the SQLAlchemy module from the flask_sqlalchemy package

app = Flask(__name__)  # Declare that this is a Flask app
app.config['SQLALCHEMY_DATABASE_URI']  = 'sqlite:///demo.db'    # Setup the database file "crud_demo.db"
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False                # Disable modification track
app.config['SECRET_KEY'] = 'mysecretkey'                            # Initialize SECRET_KEY used to encrypt your cookie
                                                                    # and save send them to the browser for session mar


db = SQLAlchemy(app)                                                # Initialize the SQLAlchemy flask database object
class Users(db.Model):                                               # Define User table class and call it 'users'
    __tablename__ = 'users'                                          # Associate it with the "users" table in the crud_
    id = db.Column(db.Integer, primary_key=True, autoincrement=True) # Define primary key id.
    username = db.Column(db.String(50))                              # Define other fields.
    email = db.Column(db.String(255))
    password = db.Column(db.String(80))
# Create entry
# product.productID, product.productCode, product.name, product.quantity, product.price,product.supplierID
user = Users(id=1003,username="kaki",email="kaki@cuhk.edu.hk",password='12345')
# Add user and commit entry
db.session.add(user)
db.session.commit()
```

Define target and create it (add).

```python
# Method I
from flask import Flask                    # Import the Flask module from the flask package
from flask_sqlalchemy import SQLAlchemy    # Import the SQLAlchemy module from the flask_sqlalchemy package

app = Flask(__name__)  # Declare that this is a Flask app
app.config['SQLALCHEMY_DATABASE_URI']  = 'sqlite:///demo.db'    # Setup the database file "crud_demo.db"
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False                # Disable modification track
app.config['SECRET_KEY'] = 'mysecretkey'                            # Initialize SECRET_KEY used to encrypt your cookie
                                                                    # and save send them to the browser for session mar

db = SQLAlchemy(app)                                                # Initialize the SQLAlchemy flask database object
class Users(db.Model):                                               # Define User table class and call it 'users'
    __tablename__ = 'users'                                          # Associate it with the "users" table in the crud_
    id = db.Column(db.Integer, primary_key=True, autoincrement=True) # Define primary key id.
    username = db.Column(db.String(50))                              # Define other fields.
    email = db.Column(db.String(255))
    password = db.Column(db.String(80))

users = Users.query.order_by(Users.id.desc()).all() # Setup users query (equivalent to SELECT * FROM Users
for user in users:                                  # Loop through all users entry by entry
    print(user.id, user.username, user.email)       # Print each one out
```

```
1003 kaki kaki@cuhk.edu.hk
```

```python
# Method II
from flask import Flask                     # Import the Flask module from the flask package
from flask_sqlalchemy import SQLAlchemy     # Import the SQLAlchemy module from the flask_sqlalchemy package

app = Flask(__name__)  # Declare that this is a Flask app
app.config['SQLALCHEMY_DATABASE_URI']  = 'sqlite:///demo.db'    # Setup the database file "crud_demo.db"
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False                # Disable modification track
app.config['SECRET_KEY'] = 'mysecretkey'                            # Initialize SECRET_KEY used to encrypt your cookie
                                                                    # and save send them to the browser for session mar


db = SQLAlchemy(app)                                                # Initialize the SQLAlchemy flask database object
class Users(db.Model):                                               # Define User table class and call it 'users'
    __tablename__ = 'users'                                          # Associate it with the "users" table in the crud_
    id = db.Column(db.Integer, primary_key=True, autoincrement=True) # Define primary key id.
    username = db.Column(db.String(50))                              # Define other fields.
    email = db.Column(db.String(255))
    password = db.Column(db.String(80))

result = db.engine.execute('select * from users')
for i in result:
    print(i)
```

```
(1003, 'kaki', 'kaki@cuhk.edu.hk', None, '12345')
```

```python
# Method III
from flask import Flask                    # Import the Flask module from the flask package
from flask_sqlalchemy import SQLAlchemy    # Import the SQLAlchemy module from the flask_sqlalchemy package

app = Flask(__name__)  # Declare that this is a Flask app
app.config['SQLALCHEMY_DATABASE_URI']  = 'sqlite:///demo.db'   # Setup the database file "crud_demo.db"
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False                # Disable modification track
app.config['SECRET_KEY'] = 'mysecretkey'                            # Initialize SECRET_KEY used to encrypt your cookie
                                                                    # and save send them to the browser for session mar


db = SQLAlchemy(app)                                                # Initialize the SQLAlchemy flask database object
class Users(db.Model):                                               # Define User table class and call it 'users'
    __tablename__ = 'users'                                          # Associate it with the "users" table in the crud_
    id = db.Column(db.Integer, primary_key=True, autoincrement=True) # Define primary key id.
    username = db.Column(db.String(50))                              # Define other fields.
    email = db.Column(db.String(255))
    password = db.Column(db.String(80))
```

```python
result = db.engine.execute('select * from users')
users = []
for i in result:          # Loop through SQL query result and add it to a users list
    users.append(i)
# print(users)
dataset = []              # Define a list called dataset
user_rec={}               # Define a dictionary called user_rect to store individual user record
for i in users:           # Loop through the users list to assign each list element to the individual user record
    user_rec['id'] = i[0]          # Field by field assignment for storing list element as dictionary value pair
    user_rec['username'] = i[1]    # Three value pairs id/id value, username/username value, email/email value
    user_rec['email'] = i[2]
    # print(i)                       # Print out each users list element
    # print(user_rec)                # Print out each user record value pair
    dataset.append(user_rec.copy())# Append each user record to the dataset list
print(dataset)
```

```
[{'id': 1003, 'username': 'kaki', 'email': 'kaki@cuhk.edu.hk'}]
```

(U)pdate MySQL Record Through SQLAlchemy Data Object

```
# Update entry
from flask import Flask                      # Import the Flask module from the flask package
from flask_sqlalchemy import SQLAlchemy      # Import the SQLAlchemy module from the flask_sqlalchemy package

app = Flask(__name__)  # Declare that this is a Flask app
app.config['SQLALCHEMY_DATABASE_URI']  = 'sqlite:///demo.db'    # Setup the database file "crud_demo.db"
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False                # Disable modification track
app.config['SECRET_KEY'] = 'mysecretkey'                            # Initialize SECRET_KEY used to encrypt your cookie
                                                                    # and save send them to the browser for session mar

db = SQLAlchemy(app)                                                # Initialize the SQLAlchemy flask database object
class Users(db.Model):                                               # Define User table class and call it 'users'
    __tablename__ = 'users'                                          # Associate it with the "users" table in the crud_
    id = db.Column(db.Integer, primary_key=True, autoincrement=True) # Define primary key id.
    username = db.Column(db.String(50))                              # Define other fields.
    email = db.Column(db.String(255))
    password = db.Column(db.String(80))

user = Users.query.filter_by(id=1003).first()
user.username = 'Kaki_Li'
user.email = 'kaki_li@cuhk.edu.hk'
db.session.commit()

result = db.engine.execute('select * from users')
users = []
for i in result:          # Loop through SQL query result and add it to a users list
    users.append(i)
# print(users)
dataset = []              # Define a list called dataset
user_rec={}               # Define a dictionary called user_rect to store individual user record
for i in users:           # Loop through the users list to assign each list element to the individual user record
    user_rec['id'] = i[0]          # Field by field assignment for storing list element as dictionary value pair
    user_rec['username'] = i[1]    # Three value pairs id/id value, username/username value, email/email value
    user_rec['email'] = i[2]
    # print(i)                       # Print out each users list element
    # print(user_rec)                # Print out each user record value pair
    dataset.append(user_rec.copy())# Append each user record to the dataset list
print(dataset)
```

Query to find target and update it.

```
[{'id': 1003, 'username': 'Kaki_Li', 'email': 'kaki_li@cuhk.edu.hk'}]
```

(D)elete MySQL Record Through SQLAlchemy Data Object

```
# Delete entry
from flask import Flask                     # Import the Flask module from the flask package
from flask_sqlalchemy import SQLAlchemy     # Import the SQLAlchemy module from the flask_sqlalchemy package

app = Flask(__name__)  # Declare that this is a Flask app
app.config['SQLALCHEMY_DATABASE_URI']  = 'sqlite:///demo.db'   # Setup the database file "crud_demo.db"
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False                # Disable modification track
app.config['SECRET_KEY'] = 'mysecretkey'                            # Initialize SECRET_KEY used to encrypt your cookie
                                                                    # and save send them to the browser for session mar

db = SQLAlchemy(app)                                                # Initialize the SQLAlchemy flask database object
class Users(db.Model):                                               # Define User table class and call it 'users'
    __tablename__ = 'users'                                          # Associate it with the "users" table in the crud_
    id = db.Column(db.Integer, primary_key=True, autoincrement=True) # Define primary key id.
    username = db.Column(db.String(50))                              # Define other fields.
    email = db.Column(db.String(255))
    password = db.Column(db.String(80))

user = Users.query.filter_by(id=1003).first()
user.username = 'Kaki_Li'
user.email = 'kaki_li@cuhk.edu.hk'

user = Users.query.filter_by(id=1003).first()
db.session.delete(user)
db.session.commit()

result = db.engine.execute('select * from users')
users = []
for i in result:         # Loop through SQL query result and add it to a users list
    users.append(i)
# print(users)
dataset = []             # Define a list called dataset
user_rec={}              # Define a dictionary called user_rect to store individual user record
for i in users:          # Loop through the users list to assign each list element to the individual user record
    user_rec['id'] = i[0]          # Field by field assignment for storing list element as dictionary value pair
    user_rec['username'] = i[1]    # Three value pairs id/id value, username/username value, email/email value
    user_rec['email'] = i[2]
    # print(i)                       # Print out each users list element
    # print(user_rec)                # Print out each user record value pair
    dataset.append(user_rec.copy())# Append each user record to the dataset list
print(dataset)
```

Query to find target and delete it.

```
[]
```

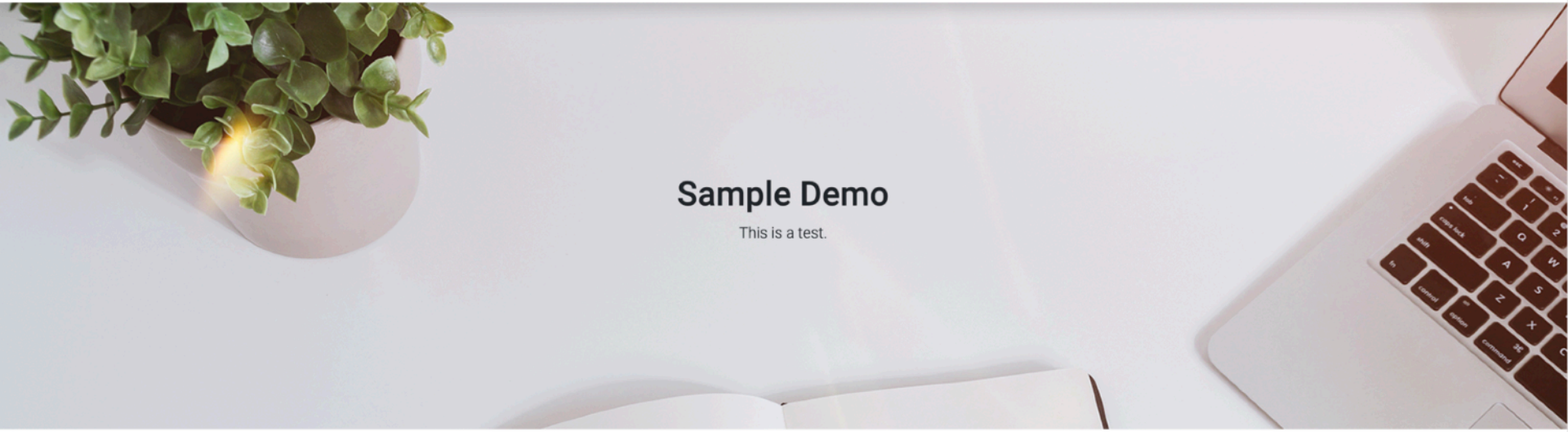
Introduction to Flask-login Manager

info@company.com 010-020-0340

Zay

Home About Sample Signup Login Profile Logout

7 +99



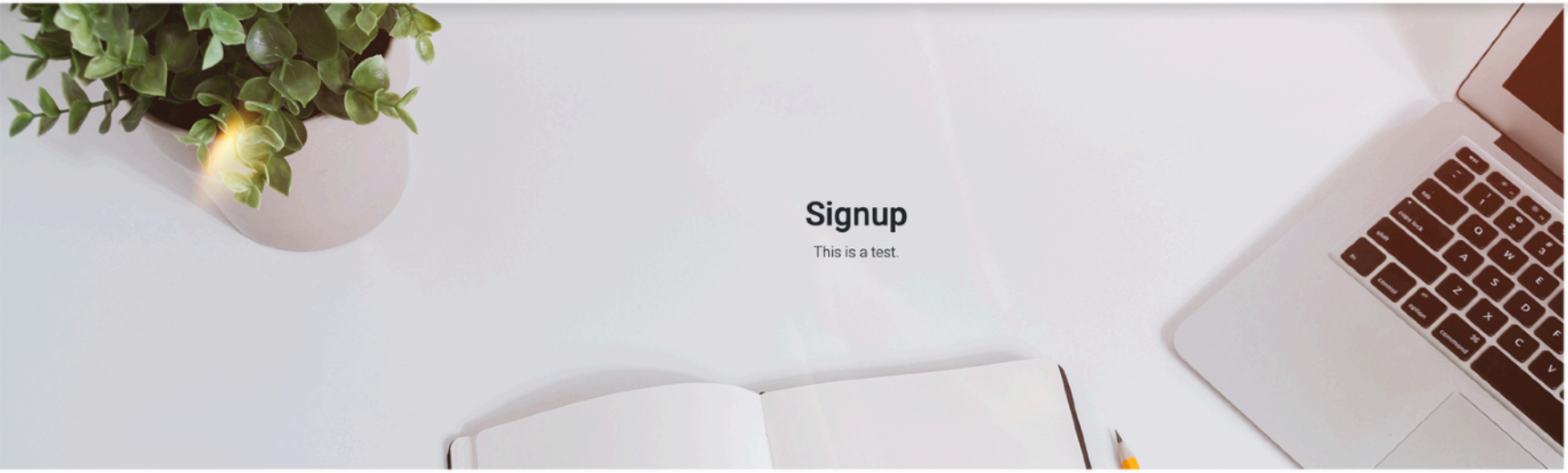
Sample Content

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod Lorem ipsum dolor sit amet.

Zay Shop Products Further Info

Zay

Home About Sample Signup Login Profile Logout



Signup

This is a test.

User name

Enter username

Email address

Enter email

Password

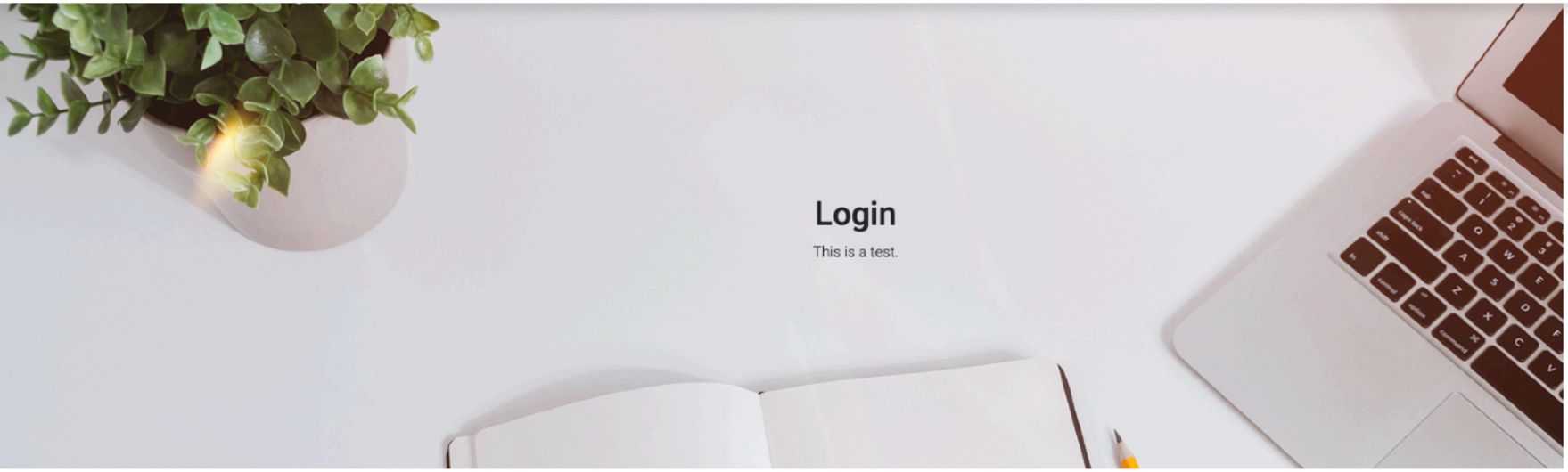
Password

Submit

Zay

Home About Sample Signup Login Profile Logout

7 +99



Login

This is a test.

User name

Enter username

Password

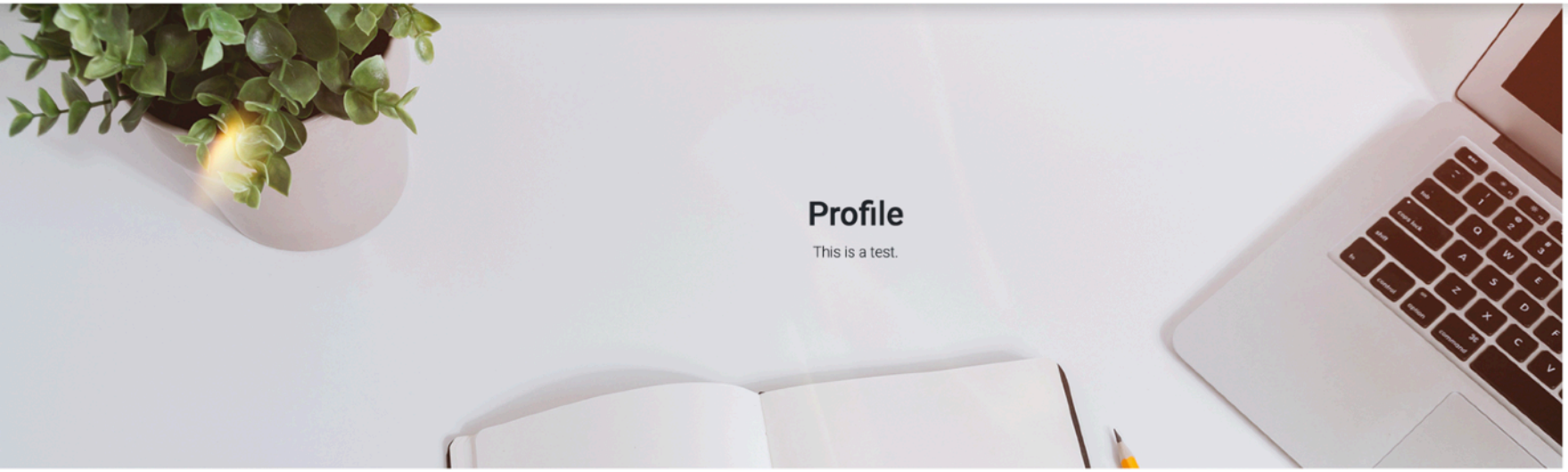
Password

Submit

Zay

Home About Sample Signup Login Profile Logout

7 +99



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod Lorem ipsum dolor sit amet.

Hello admin!

Successfully logged in.

```python
from flask import Flask, render_template, redirect, url_for, request, flash
from flask_sqlalchemy import SQLAlchemy     # Import the SQLAlchemy module from the flask_sqlalchemy package
from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user, current_user

app = Flask(__name__)
db = SQLAlchemy(app)                                    # Initialize the SQLAlchemy flask database object
app.config['SQLALCHEMY_DATABASE_URI']  = 'sqlite:///demo.db'    # Setup the database file "demo.db"
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False            # Disable modification track
app.config['SECRET_KEY'] = 'mysecretkey'                        # Initialize SECRET_KEY used to encrypt your cookie
                                                                # and save send them to the browser for session man
db = SQLAlchemy(app)
login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_message = "You need to login again."
login_manager.refresh_view = 'login'
login_manager.login_view = 'login'
```

Setup Flask login manager

```python
class Users(UserMixin, db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    username = db.Column(db.String(50))
    email = db.Column(db.String(255))
    password = db.Column(db.String(80))

@login_manager.user_loader
def load_user(user_id):
    return Users.query.get(int(user_id))
```

Load login user

```python
@app.route('/login', methods=["GET", "POST"])
def login():
    message=''
    status = "login"
    if request.method == "POST":
        username = request.form['username']
        password = request.form['password']
        print(username,password)

        user = Users.query.filter_by(username=username).first()
        if not user:
            print("Wrong username or username not found.")
            flash("Wrong username or username not found.")
            return redirect(url_for('login'))
        else:
            if password == user.password:
                print("Successfully logged in.")
                login_user(user)
                flash("Successfully logged in.")
                return redirect(url_for('profile'))
            else:
                print("Wrong password.")
                flash("Wrong password.")
                return redirect(url_for('login'))
    else:
        return render_template('login.html',status=status)
```

login_user(user) will load login user function.

```python
@app.route("/signup", methods=["GET", "POST"])
def signup():
    status = "signup"
    if request.method == "POST":
        username = request.form['username']
        email = request.form['email']
        password = request.form['password']
        print(username,email,password)
        new_user = Users(username=username,email=email,password=password
        try:
            db.session.add(new_user)
            db.session.commit()
            print("Registered!")
            return redirect(url_for('index'))
        except Exception as e:
            print("Exception occurred.",e)
            return 'Fail to add user.'
    else:
        return render_template('signup.html',status=status)
```

Add new user to database Users table.

```python
@app.route("/signup", methods=["GET", "POST"])
def signup():
    status = "signup"
    if request.method == "POST":
        username = request.form['username']
        email = request.form['email']
        password = request.form['password']
        print(username,email,password)
        new_user = Users(username=username,email=email,password=password)
        try:
            db.session.add(new_user)
            db.session.commit()
            print("Registered!")
            return redirect(url_for('index'))
        except Exception as e:
            print("Exception occurred.",e)
            return 'Fail to add user.'
    else:
        return render_template('signup.html',status=status)

@app.route("/profile")
@login_required
def profile():
    status = "profile"
    return render_template('profile.html',username=current_user.username, status=status)

@app.route("/logout")
@login_required
def logout():
    print('logout user.')
    flash('logged out.', current_user.username)
    logout_user()
    return redirect(url_for('index'))
```

Protect the route for login users only.

Protect the route for login users only.

Introduction to CRUD operations through the web interface.

Man must explore, and this is exploration at its greatest

Problems look mighty small from 150 miles up

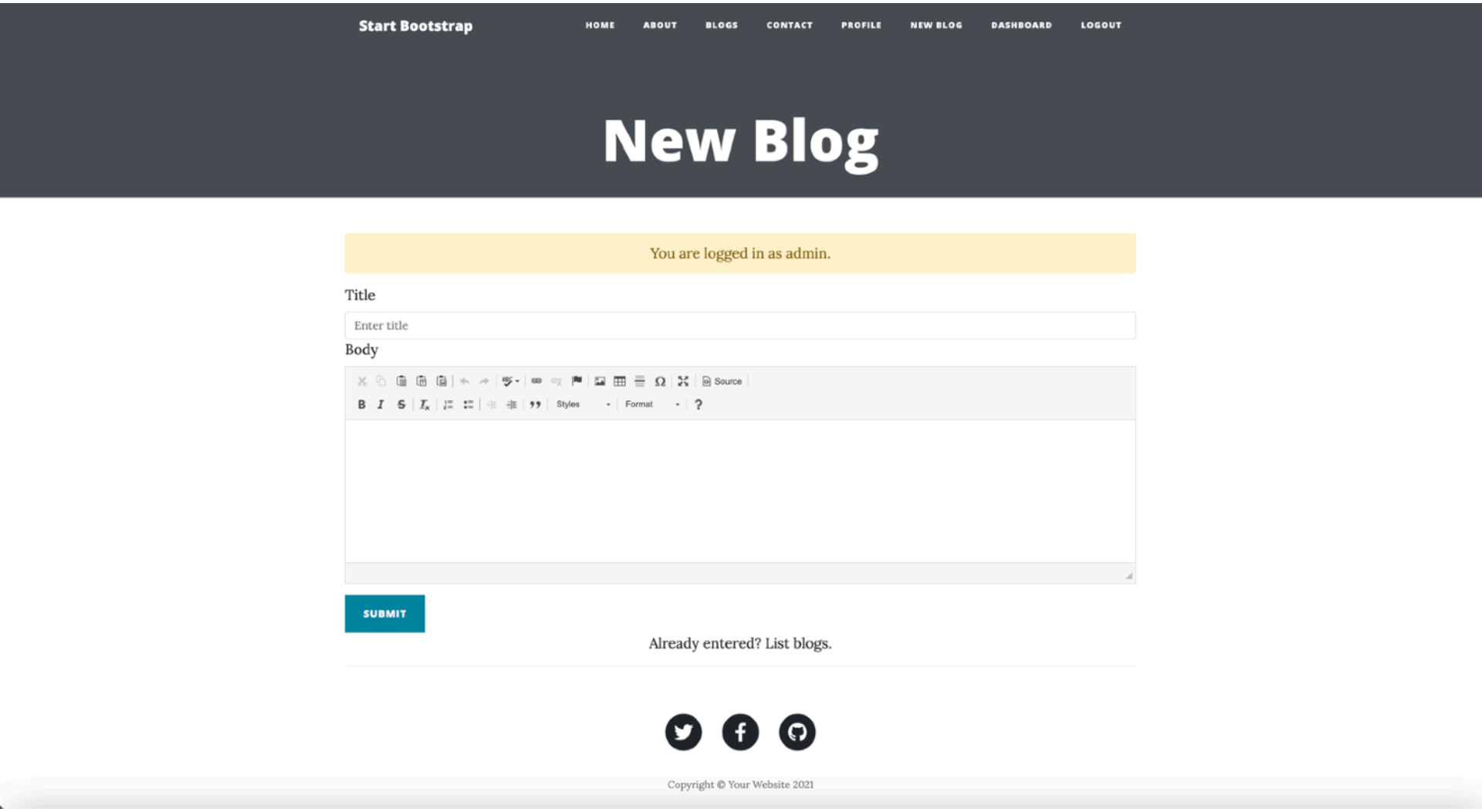
Posted by **Start Bootstrap** *on September 24, 2021*

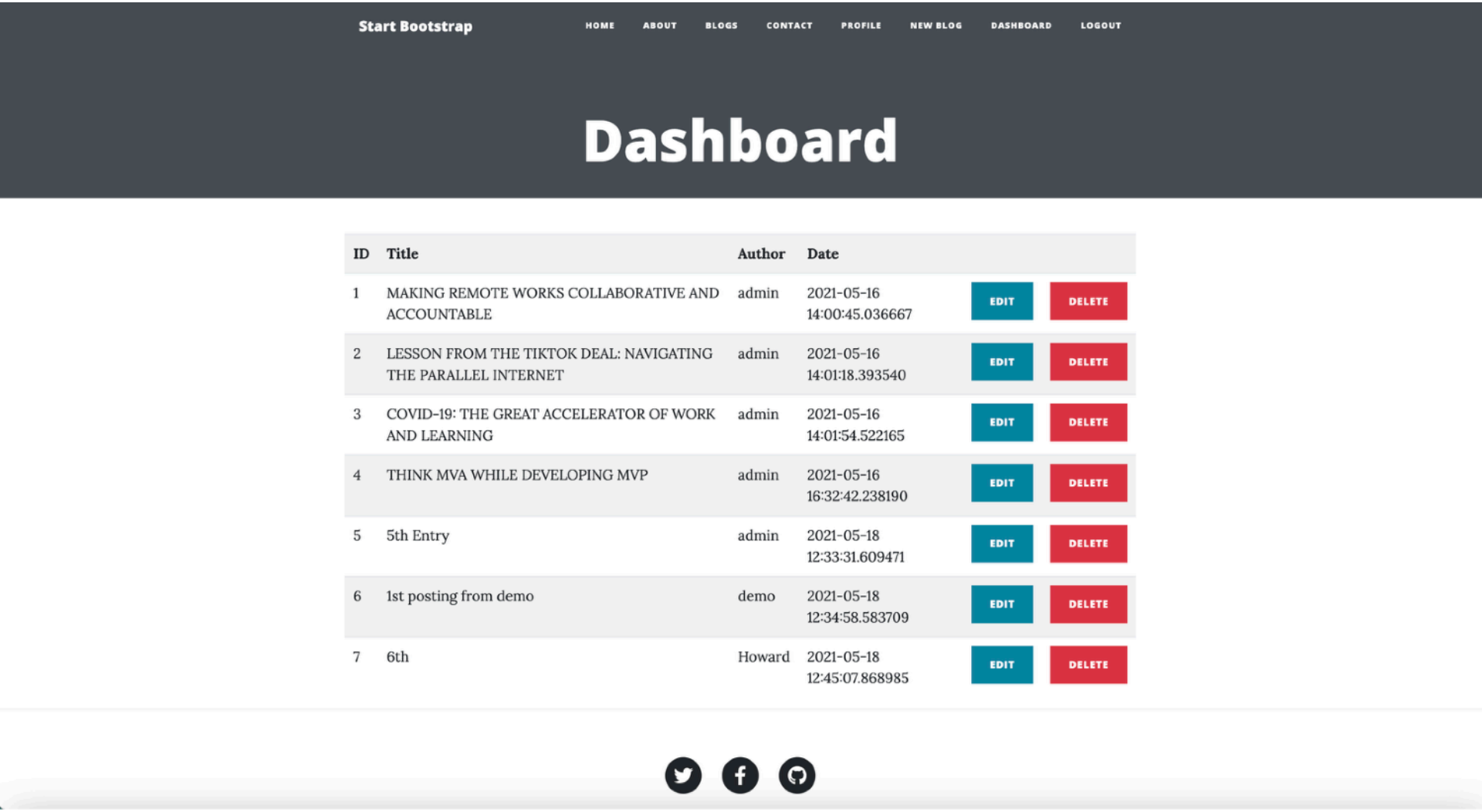
I believe every human has a finite number of heartbeats. I don't intend to waste any of mine.

Posted by **Start Bootstrap** *on September 18, 2021*

Science has not yet mastered prophecy

We predict too much for the next year and yet far too little for the next ten.


Start Bootstrap
HOME
ABOUT
BLOGS
CONTACT
PROFILE
NEW BLOG
DASHBOARD
LOGOUT
New Blog
You are logged in as admin.
Title
Enter title
Body
Source
Styles
Format
SUBMIT
Already entered? List blogs.
Copyright © Your Website 2021


Start Bootstrap
HOME
ABOUT
BLOGS
CONTACT
PROFILE
NEW BLOG
DASHBOARD
LOGOUT
Dashboard
ID
Title
Author
Date
1
MAKING REMOTE WORKS COLLABORATIVE AND ACCOUNTABLE
admin
2021-05-16 14:00:45.036667
EDIT
DELETE
2
LESSON FROM THE TIKTOK DEAL: NAVIGATING THE PARALLEL INTERNET
admin
2021-05-16 14:01:18.393540
EDIT
DELETE
3
COVID-19: THE GREAT ACCELERATOR OF WORK AND LEARNING
admin
2021-05-16 14:01:54.522165
EDIT
DELETE
4
THINK MVA WHILE DEVELOPING MVP
admin
2021-05-16 16:32:42.238190
EDIT
DELETE
5
5th Entry
admin
2021-05-18 12:33:31.609471
EDIT
DELETE
6
1st posting from demo
demo
2021-05-18 12:34:58.583709
EDIT
DELETE
7
6th
Howard
2021-05-18 12:45:07.868985
EDIT
DELETE

```
#########################################################################################################

# EXTERNAL MODULES TO BE USED

#########################################################################################################

from flask import Flask, flash, render_template, request, redirect, url_for, session
#from flask_session import Session
from flask_session.__init__ import Session
from flask_bcrypt import Bcrypt
from functools import wraps
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime

app = Flask(__name__)
bcrypt = Bcrypt(app)

#########################################################################################################

# APP CONFIGURATION

#########################################################################################################

# app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql://root:root@localhost/workshop'
app.config['SQLALCHEMY_DATABASE_URI'] = app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///workshop.db'
app.config['SESSION_TYPE'] = 'filesystem'
app.config['SECRET_KEY'] = 'thisismysecret'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
Session(app)
```

Server-side session management and encryption features added.

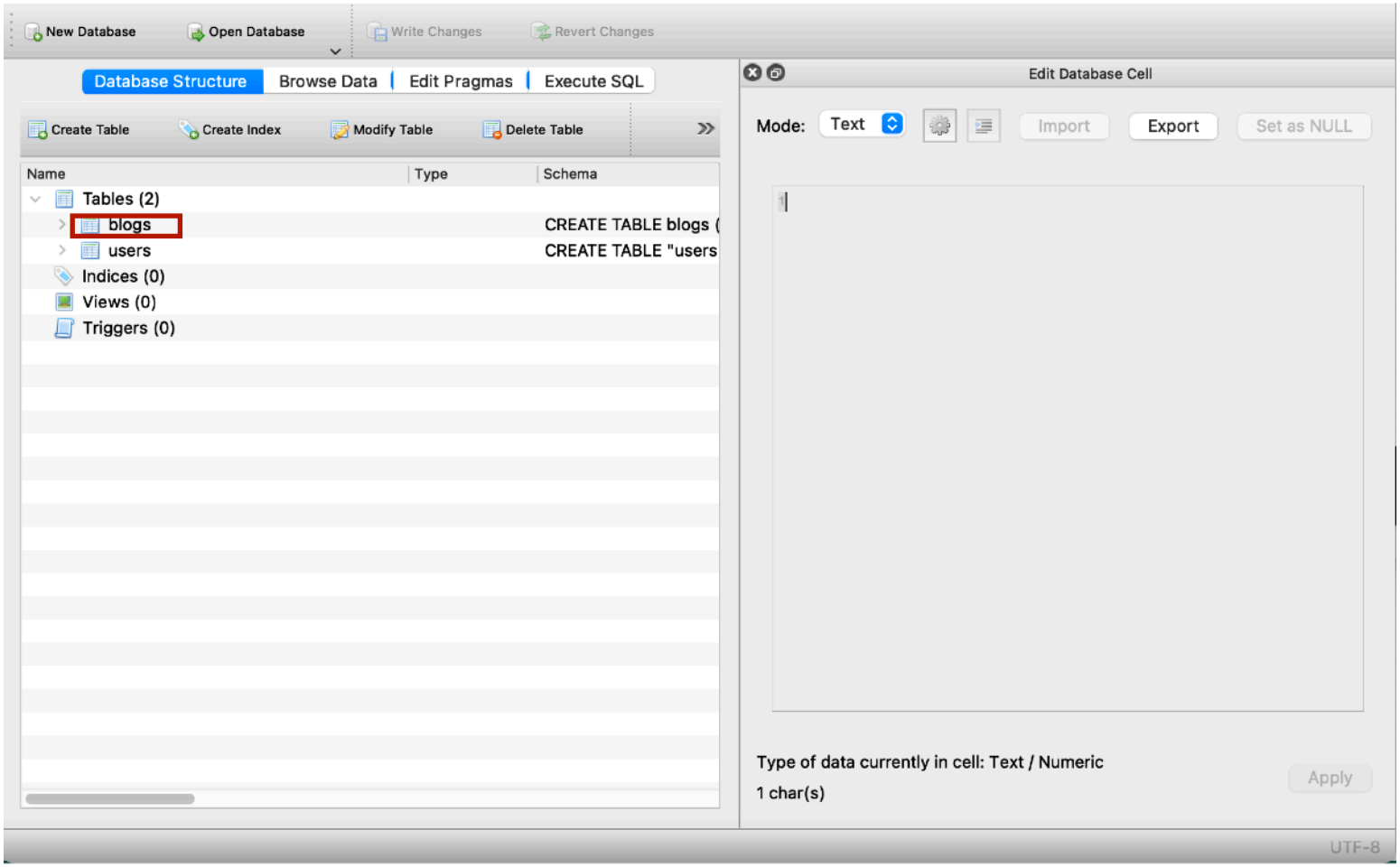

jupyter
Quit
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
0
/ jupyternotebooks / Classes / com5961 / 2022 / Lesson_9 / 05_boostrap5_ flask_crud_demo1
Name
Last Modified
File size
..
seconds ago
flask_session
32 minutes ago
Server-side session
static
a year ago
templates
a year ago
app.ipynb
Running 32 minutes ago
27.6 kB
run_db_init.ipynb
an hour ago
865 B
db_init.py
a year ago
1.3 kB
workshop.db
3 hours ago
24.6 kB

```
##########################################################################################################################

# DECORATORS

##########################################################################################################################

def login_required(func):
    @wraps(func)
    def wrap(*args, **kwargs):
        if 'logged_in' in session:
            return func(*args, **kwargs)
        else:
            flash('You need to login first.','danger')
            return redirect(url_for('login'))
    return wrap
```

Protection setup for login requirement

```
##########################################################################################################

# DATA MODELS

##########################################################################################################

db = SQLAlchemy(app)

class Blog(db.Model):
    __tablename__ = 'blogs'
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(255))
    body = db.Column(db.Text, nullable=False)
    author = db.Column(db.String(255))
    post_date = db.Column(db.TIMESTAMP, default=datetime.utcnow, nullable=False)

class User(db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(50))
    email = db.Column(db.String(255))
    password = db.Column(db.String(80))
    bio = db.Column(db.Text, nullable=False)
    admin = db.Column(db.Boolean)
    active = db.Column(db.Boolean)
```

New table defined.


New Database
Open Database
Write Changes
Revert Changes
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Create Table
Create Index
Modify Table
Delete Table
Name
Type
Schema
Tables (2)
blogs
CREATE TABLE blogs (
users
CREATE TABLE "users
Indices (0)
Views (0)
Triggers (0)
Edit Database Cell
Mode:
Text
Import
Export
Set as NULL
Type of data currently in cell: Text / Numeric
1 char(s)
Apply
UTF-8

```
##########################################################################################################################

# WEB ROUTES FOR CONTROLLING ACCESS TO TEMPLATE VIEWS

##########################################################################################################################

@app.route("/")
def index():
    return render_template('index.html')

@app.route("/about")
def about():
    return render_template('about.html')

@app.route("/post")
def post():
    return render_template('post.html')

@app.route("/contact")
def contact():
    return render_template('contact.html')
```

Login route

```
@app.route("/login", methods=["GET","POST"])
def login():
    msg = None
    if 'username' in session:
        msg = 'You are logged in as ' + session['username'] + '.'

    if request.method == "POST":
        username = request.form['username']
        password = request.form['password']
        user = User.query.filter_by(username=username).first()
        if not user:
            print("Account does not exist!")
            msg = "Account does not exist!"
        else:
            if bcrypt.check_password_hash(user.password, password):
                session['logged_in'] = True
                session['username'] = user.username
                session['admin'] = user.admin
                print("Welcome!")
                msg = "Welcome!"
                return redirect(url_for('profile', username=username))
            msg = "Wrong password!"

    return render_template('login.html', msg = msg)

@app.route("/logout")
@login_required
def logout():
    session.clear()
    return render_template('logout.html')
```

Receive submitted form field “username” through POST method.

Test submitted form field “username”.

Test encrypted password.

If ok, redirect to “profile” page.

```python
@app.route("/register", methods=["GET","POST"])
def register():
    if request.method == "POST":
        username = request.form['username']
        email = request.form['email']
        bio = request.form['bio']
        password = request.form['password']
        password = bcrypt.generate_password_hash(password)
        admin = 0
        user = User(username=username,email=email,bio=bio,password=password,admin=admin)
        db.session.add(user)
        db.session.commit()
        print("You have been registered!")

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

Registration route

Receive submitted form fields through POST method.

Encrypt password before saving.

Save new user as registered.

Dashboard route (login_required)

Test to see if user = admin

```python
@app.route("/dashboard")
@login_required
def dashboard():
    dataset = []
    if session['admin'] > 0:
        # sql = "SELECT * FROM blogs"
        blog_list = Blog.query.all()
    else:
        # sql = "SELECT * FROM blogs WHERE author = '"+session['username']+"'"
        blog_list = Blog.query.filter_by(author=session['username']).all()

    for blog in blog_list:
        dataset.append({'id': blog.id, 'title': blog.title, 'body': blog.body,
                        'author': blog.author, 'post_date':blog.post_date})
    return render_template('dashboard.html', blogs=dataset)
```

Admin will see all blog edits in dashboard.

```python
@app.route("/blogs")
def blogs():
    dataset = []
    blog_list = Blog.query.all()
    for blog in blog_list:
        dataset.append({'id': blog.id, 'title': blog.title, 'body': blog.body,
                        'author': blog.author, 'post_date':blog.post_date})
    return render_template('blogs.html', blogs=dataset)
```

Display blog listings.

```python
@app.route("/blogs")
def blogs():
    dataset = []
    blog_list = Blog.query.all()
    for blog in blog_list:
        dataset.append({'id': blog.id, 'title': blog.title, 'body': blog.body,
                        'author': blog.author, 'post_date':blog.post_date})
    return render_template('blogs.html', blogs=dataset)
```

New blog route (login_required)

```python
@app.route("/blog", methods=["GET","POST"])
@login_required
def blog():
    msg = None
    if 'username' in session:
        msg = 'You are logged in as ' + session['username'] + '.'

    if request.method == "POST":
        title = request.form['title']
        body = request.form['body']
        author = session['username']
        blog = Blog(title=title,body=body,author=author)
        db.session.add(blog)
        db.session.commit()
        msg = "New Blog posted!"
        print("New Blog posted!")

    return render_template('newblog.html', msg = msg)
```

Get form entries submitted to create new post.

Author code is saved to ensure who owns the posted entry.

```python
@app.route("/blog/<string:id>", methods=["GET","POST"])
@login_required
def blogone(id):
    dataset = []
    if request.method == "GET":
        msg = "Edit"
        dataset = []
        result = Blog.query.filter_by(id=id).first()

    if request.method == "POST":
        print('blogone update entry ok')
        title = request.form['title']
        body = request.form['body']
        del_flag = request.form['_method']
        result = Blog.query.filter_by(id=id).first()
        if result is None:
            msg = "Not found!"
            print("Not found!")
            return redirect(url_for('blogone'))
        else:
            if del_flag == 'DELETE':
                print("Deleted.")
                msg = "Deleted!"
                blog = Blog.query.filter_by(id=id).first()
                db.session.delete(blog)
                db.session.commit()
                return redirect(url_for('dashboard'))
            else:
                print("Updated")
                msg = "Updated!"
                blog = Blog.query.filter_by(id=id).first()
                blog.title = title
                blog.body = body
                db.session.commit()

    dataset.append(result)
    print(dataset)
    print(type(dataset))
    return render_template('blog.html', entries=dataset, msg=msg, id=id)
```

delete

update

Get individual blog post for edit use.

Post individual blog post for update and delete use.

```python
@app.route("/blog_details/<string:id>", methods=["GET"])
def blog_details(id):
    dataset = []
    blog = Blog.query.filter_by(id=id).first()
    dataset.append({"title":blog.title, "body":blog.body, "author":blog.author, "post_date":blog.post_date})
    return render_template('blog_details.html', blogs=dataset)
```

Display detailed page for edit.

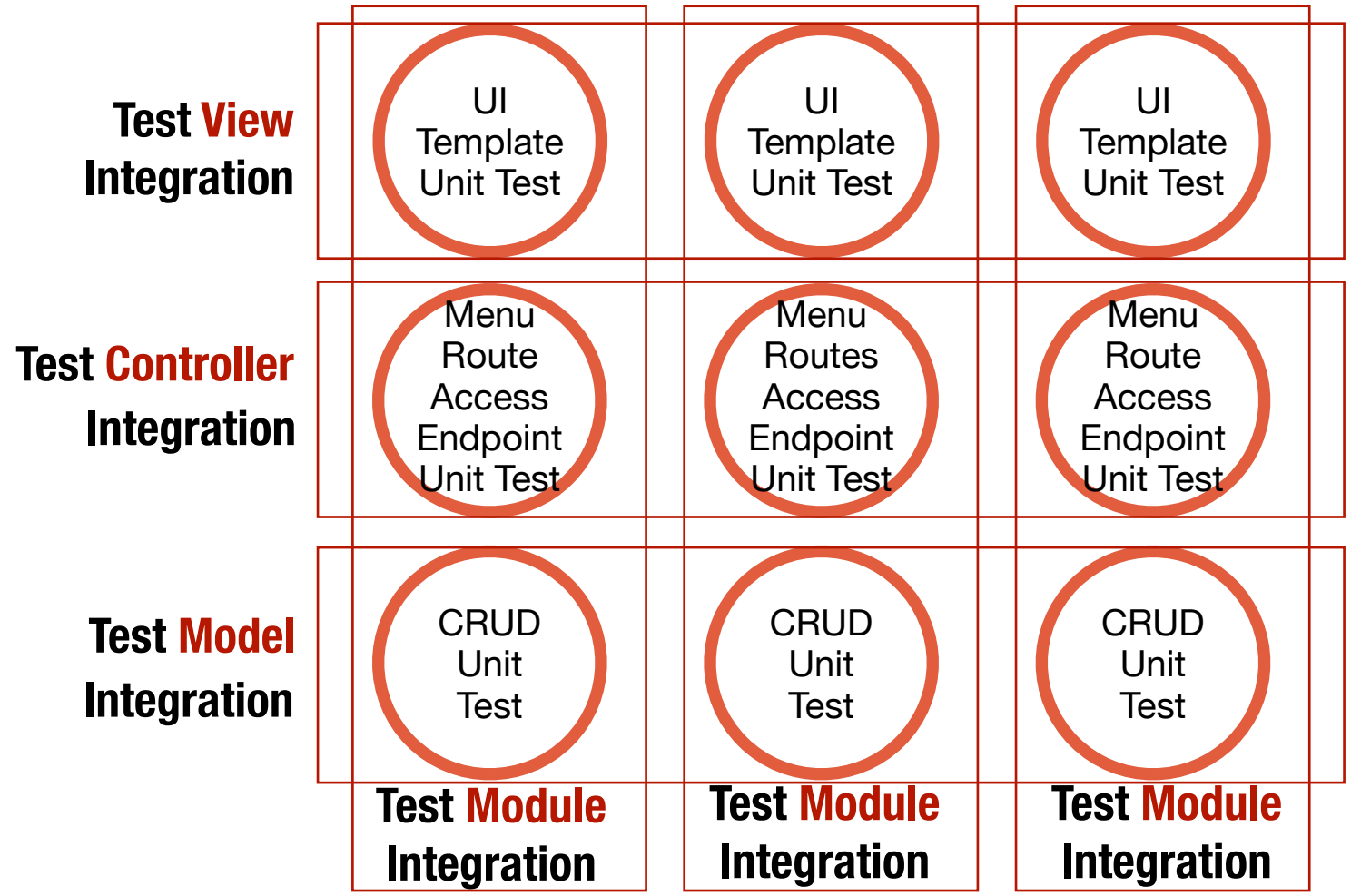
```python
@app.route("/profile/", defaults={"username": "nobody"})
@app.route("/profile/<string:username>")
@login_required
def profile(username):
    dataset = []
    user = User.query.filter_by(username=username).first()
    dataset.append({'username': user.username,'email': user.email,'bio': user.bio})

    return render_template('profile.html', entries=dataset)
```

Display profile page for edit.

Unit Testing, Integration Testing, and User Acceptance Testing (UAT)

Add Testing and Team Works into the Picture

Test Total System Integration / Test User Acceptance



UX/UI

DESIGN

DEVELOPMENT

- **Content**
- **Usability Test**
- **Wire-Frames**
- **SiteMap(IA)**
- **Requirements**

e.g. Registration Module, Discussion/Comment Module, Product Module, Search Location Module

https://www.youtube.com/watch?v=QYCaaNz8emY

Use of stubs and drivers to deal with completion and integration time lags.

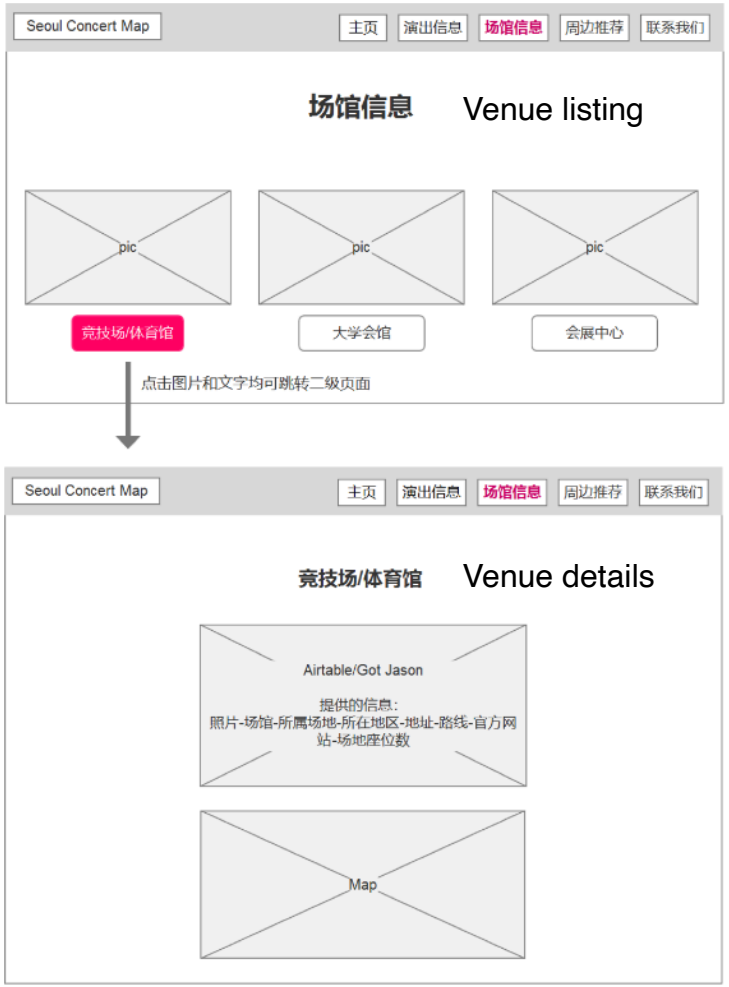


Figure 5. Wireframe-Venue information

Venue listing module completed by "A" (without screen display content coming from the data table). Need link to detail page for testing. So ask "B" to provide a stub for temporary use.

Venue detail module not done by "B". So in this case "B" can provide static template (stub) page with hard-coded content to simulate the detail page when linked from the looping list.

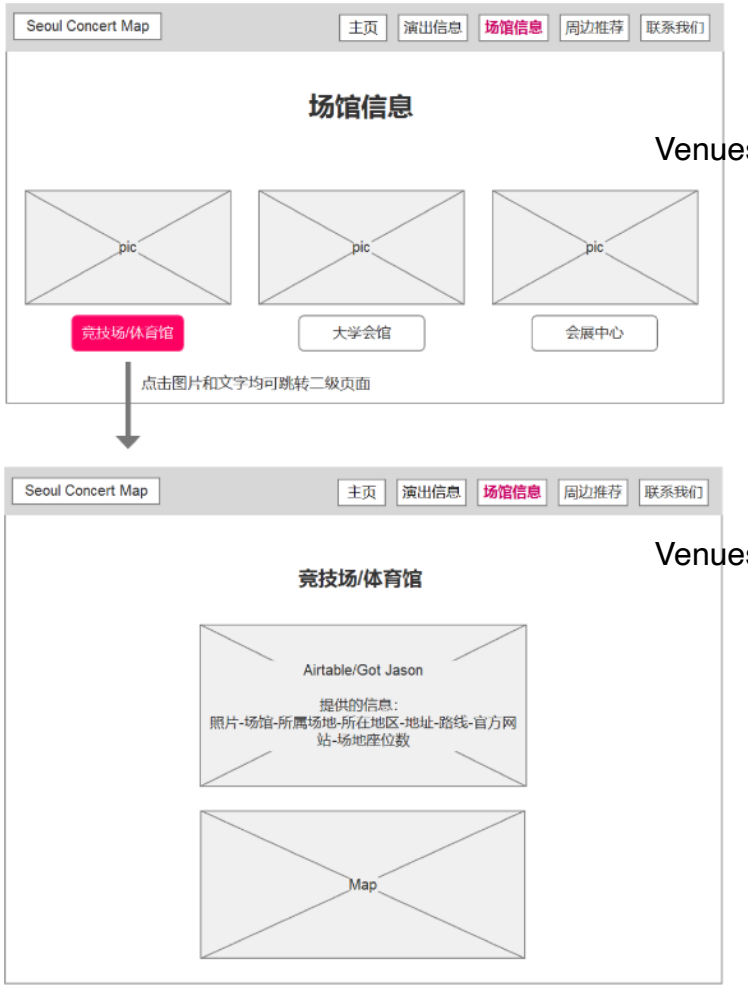
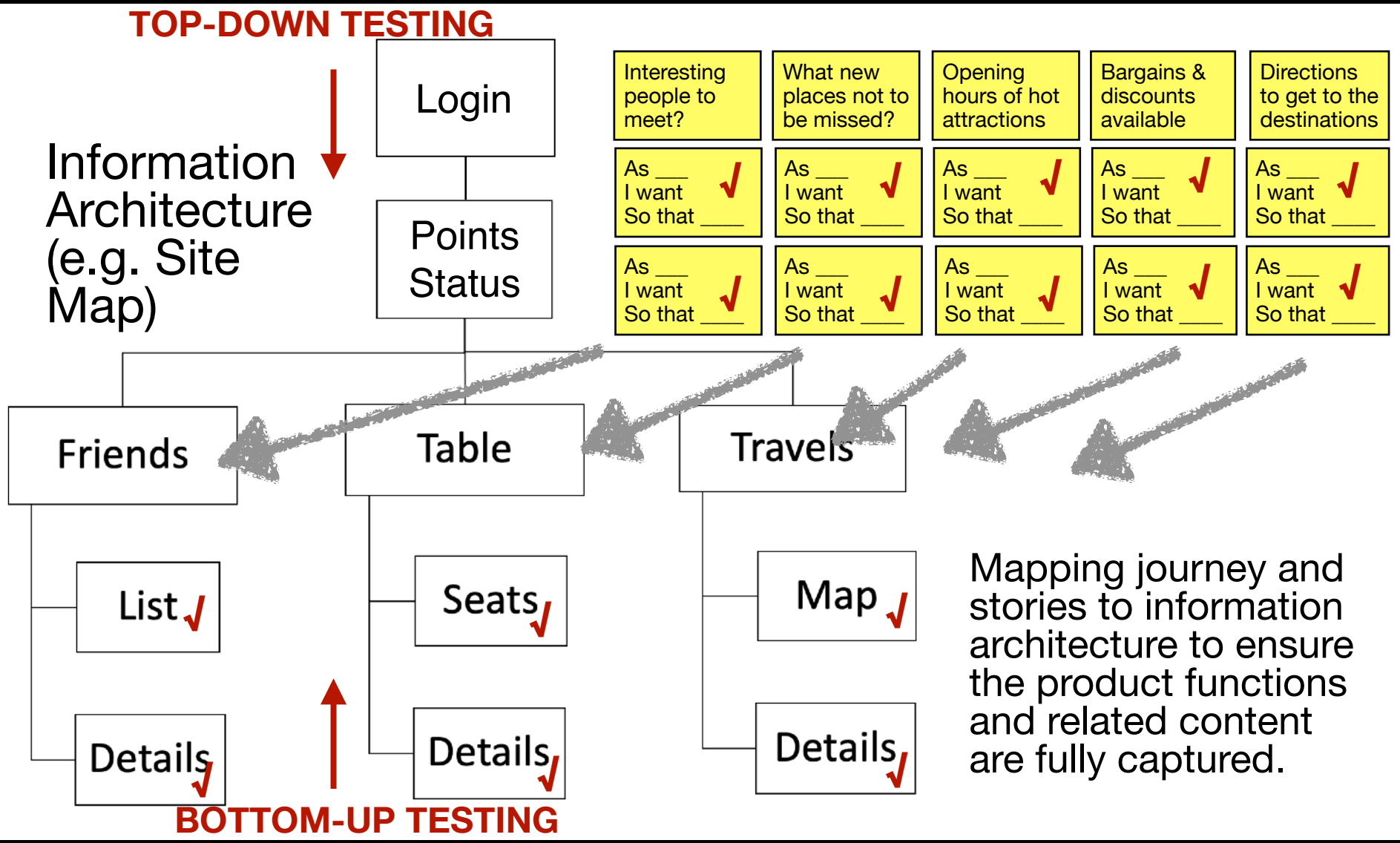


Figure 5. Wireframe-Venue information

Venue listing module not completed by “A” (without screen display content coming from the table). “A” can provide a list template page (driver) with hard coded link to the detail page for testing by “B”.

Venue detail module completed by “B”. But the listing module has not been completed by “A", so “B” has to ask “A” to provide a driver.

Recommendations for Team Development


TOP-DOWN TESTING
Login
Information Architecture (e.g. Site Map)
Points Status
Interesting people to meet?
What new places not to be missed?
Opening hours of hot attractions
Bargains & discounts available
Directions to get to the destinations
As ___ I want ___ So that ___
Friends
Table
Travels
List
Seats
Map
Details
Details
Details
BOTTOM-UP TESTING
Mapping journey and stories to information architecture to ensure the product functions and related content are fully captured.

1. Establish design standards (header, footer, information architecture, database structure-ER diagram, navigation logic)
2. Set up identical local environment for performing unit testing (maintain different versions locally)
3. Assign one person to be the integrator to perform integration testing. Once integration test is completed for the given sprint, distribute the update to the team for subsequent unit test.
4. When user acceptance test is involved, make sure the users' comments are gathered and distributed to every member on the team.

Thank you for your time!