Discussion 8

Author: Zhe Wang
01D Sit with your project group in sections by your assigned TA

<table>
<thead>
<tr>
<th>Front of Classroom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joon Young</td>
</tr>
<tr>
<td>Josh/Alex</td>
</tr>
<tr>
<td>Zach</td>
</tr>
<tr>
<td>Joyce</td>
</tr>
</tbody>
</table>
02D Sit with your project group in sections by your assigned TA

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Jason</td>
<td>Samy</td>
</tr>
<tr>
<td>Josh/Alex</td>
<td>Neel</td>
</tr>
<tr>
<td>Han</td>
<td>Justin</td>
</tr>
<tr>
<td>Samia</td>
<td>Konstantinos</td>
</tr>
</tbody>
</table>
**MVC Architecture**

- **Model**
  
  The central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.

- **View**
  
  Any representation of information such as a chart, diagram or table or web UI.

- **Controller**
  
  Accepts input and converts it to commands for the model or view.
Controller

- **Backend Endpoint**
  - Handling parsing HTTP requests send from frontend
  - Translate requests to SQL query
  - Send SQL query via connector
  - Translating database records to Data Access Objects (DAOs) (Python Objects)
  - Returning HTTP response response

```python
@bp.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('index.index'))
    form = LoginForm()
    if form.validate_on_submit():
        user = User.get_by_auth(form.email.data, form.password.data)
        if user is None:
            flash('Invalid email or password')
            return redirect(url_for('users.login'))
        login_user(user)
        next_page = request.args.get('next')
        if not next_page or url_parse(next_page).netloc != '':
            next_page = url_for('index.index')
        return redirect(next_page)
    return render_template('login.html', title='Sign In', form=form)
```
View

- Web UI
- Could be separate frontend like React, Vue.js, Angular or could be server side rendering returning HTML to browser to render such as Django
- Flask provides server side rendering with `render_template` that takes in Jinja template with context variables then returns a rendered HTML wrapped in HTTP Response
Model

- Usually represented by classes or objects depending on language
- Containing different methods for different business needs
- Translate between database table rows
- Provides an abstract interface to some type of database or other persistence mechanism

```python
class User(UserMixin):
    def __init__(self, id, email, firstname, lastname):
        self.id = id
        self.email = email
        self.firstname = firstname
        self.lastname = lastname

    @staticmethod
    def get_by_auth(email, password):
        rows = app.db.execute(""
        SELECT password, id, email, firstname, lastname
        FROM Users
        WHERE email = :email
        ", email=email)
        if not rows:  # email not found
            return None
        elif not check_password_hash(rows[0][0], password):
            # incorrect password
            return None
        else:
            return User(*([0][1:]))
```
SQLAlchemy

- SQL Connector and Object Relational Mapper (ORM)
- Open up connection to PostgreSQL
  - Execute SQL Query
  - Return Database Records
  - Map Database Records to Python Classes/Objects
Demo: A walk thru of an example project

- How to create table and load data
- Where is Model
- Where is Controller
  - What is FlaskForm ([flask-wtf](https://flask-wtf.readthedocs.io/en/0.12.0/))
  - What is API Endpoint, [URL Route Registration](https://flask.palletsprojects.com/en/1.1.x/routing/)
  - What is `render_template`
- Where is view
  - [Jinja template](https://jinja.palletsprojects.com/en/3.0.x/)
How does Flask Form Work?

- First, create a form class extending FlaskForm base class provided by flask_wtf
  - Add necessary form fields using classes from wtforms under the class
- Then in HTML template, insert a form tag block with jinja code block
  - Add all necessary fields and submit button if necessary
- Finally in the backend endpoint controller function, instantiate a new instance of the form then validate input and perform other logics with data parsed from the form

Creating Forms

Flask-WTF provides your Flask application integration with WTForms. For example:

```python
from flask_wtf import FlaskForm
from wtforms import StringField
from wtforms.validators import DataRequired

class MyForm(FlaskForm):
    name = StringField('name', validators=[DataRequired()])
```

Note:

From version 0.9.0, Flask-WTF will not import anything from wtforms, you need to import fields from wtforms.

In addition, a CSRF token hidden field is created automatically. You can render this in your template:

```html
<form method="POST" action="/">
    {{ form.csrf_token }}
    {{ form.name.label }}
    {{ form.name(size=20) }}
    <input type="submit" value="Go">
</form>
```

If your form has multiple hidden fields, you can render them in one block using `hidden_tag()`.

```html
<form method="POST" action="/">
    {{ form.hidden_tag() }}
    {{ form.name.label }}
    {{ form.name(size=20) }}
    <input type="submit" value="Go">
</form>
```

Validating Forms

Validating the request in your view handlers:

```python
@app.route('/submit', methods=['GET', 'POST'])
def submit():
    form = MyForm()
    if form.validate_on_submit():
        return redirect('/success')
    return render_template('submit.html', form=form)
```
How does HTTP work?

- The HTTP route for a service is the publicly-accessible directory path that maps to the root of your service. (xxxx.com/abc/1/0)
- HTTP Request has different type: GET, POST, PUT, PATCH, and DELETE
- The browser sends HTTP requests to the flask backend
- The flask backend sends HTTP response wrapping rendered HTML to the browser
How to define endpoint route in Flask

- Use `@app.route` or `blueprint.route`
- `<param_name>` for parameter called `param_name`
- `<int:param_name>` for enforcing parameter type
- Blueprint is used for grouping related requests/resources

The following converters are available:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>accepts any text without a slash (the default)</td>
</tr>
<tr>
<td>int</td>
<td>accepts integers</td>
</tr>
<tr>
<td>float</td>
<td>like <code>int</code> but for floating point values</td>
</tr>
<tr>
<td>path</td>
<td>like the default but also accepts slashes</td>
</tr>
<tr>
<td>any</td>
<td>matches one of the items provided</td>
</tr>
<tr>
<td>uuid</td>
<td>accepts UUID strings</td>
</tr>
</tbody>
</table>

Custom converters can be defined using `flask.Flask.url_map`.

Here are some examples:

```python
from flask import Blueprint, render_template, abort
from jinja2 import TemplateNotFound

simple_page = Blueprint('simple_page', __name__,
                         template_folder="templates")

@simple_page.route('/', defaults={'page': 'index'})
@simple_page.route('/<page>')
def show(page):
    try:
        return render_template(f'pages/{page}.html')
    except TemplateNotFound:
        abort(404)
```
What you need to implement end to end?

- Add new table in create.sql and load.sql for loading fake data
- Add new model under app/models/xxx.py (Example: app/models/user.py)
- Add new static method under the model class for each SQL query (Example: app/models/user.py::get (L92))
- Add new backend endpoint controller under app/xxx.py (Example: app/users.py::Login (L23))
- Add new HTML Jinja template under app/templates/xxx.html (Example: app/templates/login.html)