Week 3: Build Web apps with Flask: Part1

- Configuring Flask Environment
- Template inheritance
- Database with Flask-sqlalchemy
- Creating summary model
- Adding values through REPL
- Displaying summaries in the frontend

Configuring Flask Environment

Until now, we have been configuring flask environment variables manually through CLI by writing something like this.

```
// Linux / MacOS
export FLASK_APP=app.py
export FLASK_ENV=development

// Windows
set FLASK_APP=app.py
set FLASK_ENV=development
```

However, we want to avoid the inconvenience of using the command line. We'll have this load automatically by putting it in one of our dot files.

We'll two dot files: `.env` and `.flaskenv`. We want to use `.flaskenv` for any Flask CLI configuration commands and use `.env` for our app configuration.

```
touch .flaskenv
// Inside .flaskenv add the these two lines
FLASK_APP=demo
FLASK_ENV=development
FLASK_RUN_PORT=8080
```

For this to work we need to install `python-dotenv`, lets install using pipenv

```
pipenv install python-dotenv
```

Template inheritance

Base Template

lets create `base.html`, that defines a simple HTML skeleton document for the project.The `base.html` file contains header, content area and footer

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% block head %}
 <link
    rel="stylesheet"
    href="{{url_for('static',filename='css/main.css')}}"/>
    <!-- <link rel="stylesheet" href="site/static/main.css"> -->
    <title>{% block title %}{% endblock title %} - Dhambaal</title>
    {% endblock head%}
</head>
<body>
    <!-- Header -->
    {% include "header.html" %}
    <div class="content">
        {% block content %}
        <!-- Content will be displayed here -->
        {% endblock content%}
    </div>
  <!-- Footer -->
  {% include "footer.html" %}
    </div>
</body>
</html>
```

In this example, the {% block %} tags define four blocks that child templates can fill in. All the block tag does is tell the template engine that a child template may override those placeholders in the template.

Child Template

A child template might look like this:

```
{% extends 'base.html'%}
{% block title %}Home{% endblock %}
{%block content%}
<div class="container">
  <h1>All Posts</h1>
  {% for post in posts %}
  <p>
    Title:{{ post.title }}<br />
    Description: {{ post.description }}<br />
    Source: {{ post.source }}<br />
    Date: {{post.created_at}}<br />
  </p>
  {%endfor%}
```

Database with Flask-sqlalchemy

Flask-SQLAlchemy is an extension for Flask that adds support for SQLAlchemy to your application. It aims to simplify using SQLAlchemy with Flask by providing useful defaults and extra helpers that make it easier to accomplish common tasks.

What is SQLAlchemy?

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.

It provides a full suite of well known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language.

To install flask-sqlalchemy

```
pipenv install flask-sqlalchemy
```

Creating summary model

For the common case of having one Flask application all you have to do is to create your Flask application, load the configuration of choice and then create the SQLAlchemy object by passing it the application.

Once created, that object then contains all the functions and helpers from both **sqlalchemy** and **sqlalchemy.orm**. Furthermore it provides a class called Model that is a declarative base which can be used to declare models:

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///dhambaal.db'
db = SQLAlchemy(app)


class Summary(db.Model):
    __tablename__ = "summary"
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(80), nullable=False)
    description = db.Column(db.Text)
    source = db.Column(db.String(129),nullable=True)
    created_at = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
    modified_at = db.Column(db.DateTime, nullable=False, default=datetime
.utcnow,onupdate=datetime.utcnow)

    def save(self):
        db.session.add(self)
        db.session.commit()

    def delete(self):
        db.session.delete()
```

Adding values through REPL

```
from app import db
from models.summary import Summary

// Create tables
db.create_all()
post1 = Summary(title="Title",description="description of the post",source="source of
the post")
post1.save()
```