Introduction

Welcome to Module 5! This week, we will learn about creating middleware that is powered by a datastore. The datastore used is a document database named MongoDB. MongoDB databases are organized in collections that hold documents.

Topics and Learning Objectives

Topics

- MongoDB, Mongo Compass
- Domain modeling
- Working with ORM – MongoENgine
- Datastore, database
- Collection, documents
- MongoDB, MongoEngine
- find, findFirst, save, delete

Learning Objectives

By the end of the module, you should be able to:

1. Explain the concept of dynamic web applications.
2. Describe how to develop middleware powered by datastores.
3. Create middleware that provides personalized web pages with data retrieved from MongoDB.

Readings & Resources

Reading

Required

- Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python* (2nd ed.). O'Reilly Media, Inc.
    - Chapter 3: Templates
    - Chapter 4: Web Forms

- Bootstrap Documentation

Python and Flask

- Python is a programming language. Within Python there are many frameworks. One such framework is Flask. Flask is managed as a python module. That is you need to import it into your Python source code before using it.
- Flask is a micro-framework that is lightweight and provides essential components for creating web services. Flask provides the following capabilities: routing, request handling and session management. Flask aids Python developers in creating middleware applications.
- Flask implements WSGI (Web Server gateway Interface). WSGI is a universal interface between the web server and the web application.

WSGI

- Flask implements WSGI (Web Server Gateway Interface). WSGI is a universal interface that enables your middleware to have web server capabilities. Note: Flask is WSGI compliant and through Flask, WSGI is supported by Python 3.
- The Web Server Gateway Interface (WSGI) is a simple calling convention for web servers to forward requests to web applications or frameworks written in Python/Flask applications. This enables your middleware to handle HTTP requests using HTTP responses. There are two HTTP requests that are of interest ot us: GET and POST.
- WSGI is defined by PEP-3333.
- WSGI has two sides
  - The server/gateway side – this is running full web server software such as Apache or Nginx
  - The application side/framework side – a python callable supplied by a Python program

WSGI Middleware

- A WSGI middleware component is a Python callable that is itself a WSGI application. It may handle requests by delegating to other WSGI applications.
- A middleware component can perform functions as:
  - **Routing –** a requests is routed to different application objects based on the target URL
  - **Concurrency –** allowing multiple applications or frameworks to run side-by-side on the same server

  - **Load balancing –** forwards request to servers that have least usage or most resources available
  - **Post-processing –** formatting using templates

Installing Modules Using Pip

- Pip is the package installer for Python.
- Flask is a Python module and needs to be installed using pip before developing web applications using Python.
- To list available packaged use the following command:
  a. `pip list`
- To install Flask on your machine, use the following command:
  a. `pip install flask`
- Pip-review is a module that can be used to check available updates. Command (a) installs pip-review on your desktop. Command (b) lists all updates for your modules. Command (c) performs an upgrade of your Python modules. The interactive flag enables interaction mode which will prompt you where you would like to update each package:
  a. `pip install pip-review`
  b. `pip-review`
  c. `pip-review --interactive`

Creating a Flask Application

- Python provides tools to create a skeleton middleware application.
- You need to install flask-appbuilder, mongoengine and flask_mongoengine modules for Python/Flask from the command line. You can use the commands below:
  - `pip install flask-appbuilder`
  - `pip install mongoengine`
  - `pip install flask_mongoengine`
- A skeleton app may be created using the “flask” script available in $HOME/.local/bin
  - `flask fab create-app`
  - The above command asks a series of questions and creates the application file structure based on answers.

- Once the skeleton app has been created it may be executed using "`flask run`" command from the project directory.
- **Note:** if you do not want to use the command above "`flask run`" you can execute your application using "`python3 run.py`" from the command line.

- The "`flask fab create-app`" command generates several files.
- The file "run.py" is used to start your application (or middleware) to listen on the specified port to everyone or a specific ip.
  - Host value of 0.0.0.0 denotes every ip.
  - The default port is 8080. This port can be overwritten. The first 1024 ports are reserved. This means that they should not be used by any of your applications. As a developer you may use anything above 1024. That is port 8080 is acceptable, port 5000, port 5010, and so on.
  - Port usage is documented on the Wikipedia TCP and UDP Port Numbers page.
- The file "config.py" is used to configure your middleware environment. This is where we configure the server hosting our datastore, the port we use to access our datastore and any user accounts.
- The file "/app/views.py" is used to configure views and routes.
- Open /app/views.py and to the end of the file add the code below.

```
@appbuilder.app.route('/hello', methods=['GET'])
def hello_route():
    print('inside hello_route method')
    return 'Hello World!'
```

- Once the code has been entered, the application can be executed using the following command
  - "`Flask run`" or "`python3 run.py`"
  - To stop an application from running use CTRL + C on the terminal where your application or middleware is running
- To access the application a web browser is needed. The URL to enter in the browser window is http://localhost:8080/hello. Note the port, 8080 can be found in the file named config.py. You can change this port to another port if you would like. Port 8080 is traditionally used for middleware applications. A problem might arise if you have multiple middleware applications on your machine. Hence you should consider using ports such as 5010, 5020, 5030, and so on.

Debug Mode

- The skeleton app uses debug mode by default. This may change from version to version of flask-appbuilder.
- To place the application in debug mode you will have to edit run.py file and add the following parameter, "debug=True" to app.run().

  - `app.run(host='0.0.0.0', port=5010, debug=True)`
  - “`flask run`” may be executed afterwards to start your middleware

- To use the debugger a Debugger PIN will be displayed on the console or terminal where “flask run” was executed.
- Debug mode will result in the following:
  - Debug mode activates automatic reloader.
  - Debug mode activates the Python debugger.
  - Debug mode enables Flask application debugging.

Debugging Demonstration

- Open /app/views.py and modify hello_route(). Note “raise Exception(“…”)
- Raise Exception() will generate an exception and automatically invoke the debugger. See the code below.

```
@appbuilder.app.route('/hello', methods=['GET'])
 def hello_route():
    print('inside hello_route method')
    raise Exception('A custom exception to demo debugging')
    return 'Hello World!'
```

Debugging Pin

- When an exception has been generated by your application the web browser displays a stack trace.
- A stack trace is a report of the active stack frames at a certain point in time during the execution of a program. In this case the point in time where the exception has been generated.
- A stack trace shows which method and more specifically which line caused the exception.
- To view the stack trace a Debug PIN will be required. The Debug PIN may be obtained from the console where the Flask application was started.

Testing Flask Application

- Test-Driven Development (TDD) may be used for your project.

- A module PyTest may be used.
    - `pip install pytest`

- Inside the “app” directory/folder create “tests” directory/folder.
- Inside “tests” directory folder create a file test_hello.py and enter the code below.

```
from app import appbuilder
import pytest

@pytest.fixture
def client():
    """ A pytest fixture for test client """
    appbuilder.app.config["TESTING"] = True
    with appbuilder.app.test_client() as client:
        yield client

def test_hello(client):
    """ A test method to test view hello """
    resp = client.get("/hello", follow_redirects=True)
    assert 200 == resp.status_code
```

- Test_hello.py can be executed directly from the command prompt using the command below.
    - `python3 test_hello.py`

MongoDB

- MongoDB is a popular NoSQL database.
- MongoDB is a document database that stores JSON-like documents into collections.
- A MongoDB database can house multiple collections and each collection can store many documents.
- MongoDB does not require any languages for defining schemas and collections. The same is true for accessing documents.
- In MongoDB data (or documents) are represented by JSON. A user submits JSON requests and MongoDB returns JSON.

Using MongoDB

- On your VM, MongoDB 4.4.1 is installed
- MongoDB may be accessed by a GUI client named MongoDB Compass.
- MongoDB I Compass connects to MongoDB 4.4.1.
- The URI to connect to MongoDB is mongodb://localhost:27017
    - mongodb is the protocol
    - localhost is the server ip
    - 27017 is the port for accessing MongoDB
    - **Note:** this is a development server, hence there is no username or password

Data Models

- A data model represents a Python class that will mirror a document in MongoDB. That is a data model will be represented by Python code and its data will be persisted (stored) in a MongoDB document.
- The main function of the data model is to show which fields contain data that will be persisted. Each field has a value and a type.
- The Python code below represents a User that has two fields: name and email.

```
class User(db.Document):
    name = db.StringField()
    email = db.StringField()
```

- **Note:** data models may be created after server connections to MongoDB. See next slide for server connections using MongoEngine.

MongoEngine

- Python/Flask can not access MongoDB directly. A module named MongoEngine will enable to retrieve and persist models/documents to MongoDB.
- The Python code below exhibits MongoDB configuration setting and connection to database.

```
app.config['MONGODB_SETTINGS'] = {
    'db': 'ckcs145',
    'host': 'localhost',
    'port': 27017
}
db = MongoEngine()
db.init_app(app)
```

Data Retrieval

- Our data model is represented by class User (see Data Models slide).
- To query or retrieve data from MongoDB we can interact with our data model.
- To access our data model we can simply use the Python class that was created. This class is named User.
- To retrieve a specific user we can submit a query.
  - `User.objects(name="Mike Jones").first()`
  - **Note:** User is our Python class and "Mike Jones" is what is retrieved from MongoDB.
  - In our database there may be multiple "Mike Jones", hence we want to retrieve only the first occurrence.

Data Persistence

- Data may be persisted or stored in the database.
- To persist or save data from MongoDB we can interact with our data model.
- To access our data model we can simply use the Python class that was created. This class is named User.
- To persist a specific user we can submit a query.
  - `User.objects(name='Mike Jones', email='mjones@global.com').save()`
  - The save method persists "Mike Jones" and mjones@global.com to a document in MongoDB

Changing Data

- Deleting a user is a two part process. First find the user and then delete the user only if it is found. This is demonstrated below.

```
user = User.objects(name='Mike Jones').first

user.delete()
```

- Updating a user is a two process. First find the user and then you can update required fields. This is demonstrated below.

```
user = User.objects(name='Mike Jones').first

user.update(email='mike.jones@global')
```

Implementing Middleware with MongoDB Support

- The code below may be placed in /app/views.py.
- The code below may not work due to security configurations on your web server. HTTP requests are made using GET, PUT, POST and DELET. PUT and DELETE requests are disabled by default on servers. It is good practice to create middleware apps that only support GET and POST HTTP requests.
- The code below represents a single file application that provides your middleware that is backed by a MongoDB database. Note that there are two routes supporting a GET and POST request. The GET request retrieves all users and the POST request inserts a single user into MongoDB.

```
import json
from flask import Flask, request, jsonify
from flask_mongoengine import MongoEngine

app = Flask(__name__)
app.config['MONGODB_SETTINGS'] = {
     'db': 'CKCS145',
     'host': 'localhost',
     'port': 27017
}
db = MongoEngine()
db.init_app(app)

class User(db.Document):
     name = db.StringField()
     email = db.StringField()
     meta = { 'collection' : 'User', 'allow_inheritance': False }

@app.route('/', methods=['GET'])
def list_all_users():
     data = list(User.objects)  # Create a list
     return data

@app.route('/', methods=['POST'])
```

```
def create_user():

     data_name = request.form.get('Name')
     data_email  = request.form.get('Email')

     user = User(name=data_name, email=data_email)

     return user.to_json()

if __name__ == "__main__":
     app.run(debug=True)
```

Video

Please watch the following video, Introduction to MongoDB and MongoDB Compass. This video demonstrates how to work with a MongoDB service.

Introduction to MongoDB and MongoDB Compass
TMU Video

Summary

Middleware enables an application to provide business logic that is scalable and accessible over the internet. Middleware responds to GET and POST requests submitted by web pages running inside a web browser. Middleware that is backed by a database in our case MongoDB allows for the creation of business logic that can provide session contents for users' web pages. That means that each user can have information that is relevant to their session. Furthermore, sessions can be isolated based on users' requests.

Assessments

Each week, you'll find here reminders of assignments or labs that you should be working on. For Week 5:

- Lab 5: Creating a middleware application that uses a MongoDB database (5% of the final grade) is due in 7 days, i.e., the end of day, Friday, of Week 5.

Reminder

You have a weekly Zoom session. The date/timing for your Zoom session can be found in the Course Schedule in the Course Outline. The link to the Zoom session will be provided separately to you by your instructor. The Zoom session will consist of 20–30 minutes of lecture, followed by questions and answers (Q&A), followed by a lab period. You may ask questions about the previous or current week's content or labs.

Please bring any questions to the weekly Zoom session. However, if you have any questions during the week outside of the Zoom session, you may post them in the Discussion Board. Your instructor may respond in the Discussion board or during the weekly Zoom session.

Click-n-reveal: **Where is the Discussion board?**

Click on the "Communication" area at the top main course menu, and select Discussions from the dropdown menu.